Philipp Schiele

On Robust Asset Allocation

Dissertation an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Eingereicht am 12.01.2024

Philipp Schiele

On Robust Asset Allocation

Dissertation an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Eingereicht am 12.01.2024

Erster Berichterstatter: Prof. Stefan Mittnik, PhD
Zweite Berichterstatterin: Prof. Dr. Sandra Paterlini
Dritter Berichterstatter: Prof. Dr. Svetlozar Rachev
Tag der Disputation: 21.05.2024

Acknowledgments

This dissertation would not have been possible without the continued support of my advisor, Stefan Mittnik, who believed in me and gave me the freedom to explore my own ideas, yet always provided guidance and support when needed. I am also grateful for the unwavering support of Stephen Boyd, for inviting me to Stanford and giving me the opportunity to work with him and his group. Furthermore, I thank Sandra Paterlini and Svetlozar Rachev for agreeing to be part of my dissertation committee without hesitation, and Christian Heumann for chairing the committee.

At the Chair of Financial Econometrics, I not only found colleagues but also great friends. Christoph Berninger, Dennis Mao, Henry Port, and Maria Sprincenatu, thank you for the warm welcome and the great camaraderie. Additionally, Martina Brunner's meticulous attention to detail on all administrative matters made my life as a doctoral student much easier. The lively atmosphere at the LMU statistics institute was always enjoyable, with the summer retreat and a nightly swim in the Ammersee being a particular highlight. David Rügamer's mentorship was instrumental during my first project, and his guidance at any time of day or night was invaluable.

I would also like to thank all my co-authors of the papers included in this dissertation, namely, Christoph Berninger, Stephen Boyd, Kasper Johansson, Eric Luxenberg, Ronald Kahn, David Rügamer, and Thomas Schmelzer, as well as everyone who provided feedback on my papers, for pushing me to do my best work and for the many interesting discussions we had.

Neither the time at LMU nor the time at Stanford would have been nearly as joyful without the many friends I made along the way, as well as the ones that were there from the very beginning. I would like to thank each and every one of you for the many great memories we made together.

To my dear friend Eric Luxenberg, I cannot describe how lucky I feel to have met you. From the countless late night sessions working on our projects together to spending Thanksgiving with your parents, you have felt like a brother to me from the very beginning.

Contributing to the open-source project CVXPY was not only a great way to procrastinate on my dissertation, but it brought immense joy to my life, and for every contribution I made, the rewards were tenfold. I did not envision speaking about self-landing rockets in Texas when I submitted my first line of code. I am delighted to work with many talented people on this project, in particular, Steven Diamond, Akshay Agrawal, Riley Murray, Bartolomeo Stellato, and Parth Nobel.

My colleagues at Scalable Capital also deserve a special mention, in particular my mentor Nikolay Robinzonov, as well as the entire Financial Engineering team. Working with them not only gave me the opportunity to apply the knowledge I gained at university in practice, but provided me with a great environment to learn and grow as a professional.

Finally, the unconditional love and support I received from my parents Helmut and Erika as well as from my sister Elena, my brother-in-law Joel, and my nephew Avi has been a cornerstone for me not only during my time as a doctoral student but throughout my entire life. I would not be where I am today without you, and I am eternally grateful to have you in my life. I also extend my deepest gratitude to God, as it is through His grace that I have been able to achieve this milestone in my life.

This journey, filled with growth, learning, ups and downs, and many great memories, was made possible by your collective support. To all of you, I am deeply grateful, carrying the lessons and memories into my future endeavors with great joy and excitement.

Summary

This dissertation offers a multifaceted examination of robust asset allocation strategies in the context of financial portfolio management, a subject of paramount importance and great interest to investors seeking to navigate the intricacies and complexities of modern financial markets, with significant implications for both academic research and practical applications. The field intersects various disciplines, including finance, economics, mathematics, statistics, computer science, and behavioral psychology. As such, it presents a unique and compelling opportunity for innovative and multidisciplinary approaches to portfolio management, opening avenues for new research and applications.

The initial chapter provides an introduction to the topic, defining the term robustness in the context of financial portfolio management as the ability of a portfolio to perform well under a variety of perturbations. It also highlights the various facets of robust asset allocation, including statistical robustness, computational robustness, model robustness, and more, each being highlighted for its critical importance in the broader context of effective portfolio management. This chapter also sets the stage for the dissertation by providing a comprehensive introduction for each essay included, describing the motivation, notation, and key contributions of each research article.

The body of the dissertation comprises several contributing essays. The first essay revisits Harry Markowitz's pioneering portfolio construction model. It demonstrates how extending this foundational model with practically relevant constraints and an explicit formulation of uncertainty in forecasting return statistics can address many of the alleged shortcomings of the original model. The second essay introduces an innovative domain-specific language, designed to simplify solving convex-concave saddle problems. This language has direct and significant applications in the realm of robust optimization, offering a new tool for researchers and practitioners alike. The third essay focuses on the area of robust bond portfolio construction, providing advanced methods for assessing and optimizing bond portfolios under a variety of market conditions, adding depth to this important area of portfolio management. The fourth essay concerns itself with computational methods in the field of behavioral finance. It presents novel approaches for portfolio optimization, using cumulative prospect theory utility to effectively integrate investor behavior and preferences into portfolio construction, thus bridging the gap between theory and practical investor behavior. The final essay serves as a bridge between traditional statistical methods and contemporary deep learning techniques. It introduces the Autoregressive Moving Average (ARMA) cell as a novel component for neural autoregressive modeling, thereby significantly expanding the toolkit available for robust forecasting in various applications, including financial forecasting.

In summary, this dissertation contributes to multiple dimensions in the field of robust asset allocation. It provides not only comprehensive theoretical insights and practical applications but also fosters a collaborative environment for continued innovation and development in this dynamically evolving field. The inclusion of open-source software with each research contribution greatly enhances the practical utility of the work. This ensures that the insights and methodologies developed can be readily applied and extended in both academic and industry settings, and thus have the potential to make a lasting impact on the field.

Zusammenfassung

Diese Dissertation stellt eine umfassende Untersuchung von robusten Anlagestrategien im Kontext des Finanzportfolio-Managements dar, einem Thema von höchster Bedeutung und großem Interesse für Investoren im Hinblick auf die Herausforderungen und Komplexitäten moderner Finanzmärkte. Dies hat bedeutende Implikationen sowohl für die akademische Forschung als auch für praktische Anwendungen. Das Feld überschneidet sich mit verschiedenen Disziplinen, darunter Finanz- und Wirtschaftswissenschaften, Mathematik, Statistik, Informatik und Verhaltenspsychologie, und bietet somit eine einzigartige Gelegenheit für innovative und multidisziplinäre Ansätze im Portfolio-Management, die neue Wege für Forschung und Anwendungen eröffnen.

Das einleitende Kapitel bietet eine Einführung in das Thema und definiert den Begriff der Robustheit im Kontext des Finanzportfolio-Managements als die Fähigkeit eines Portfolios, unter einer Vielzahl von Marktveränderungen hinreichend gute Renditecharakteristiken zu erzielen. Es beleuchtet auch die verschiedenen Facetten der robusten Vermögensallokation, wie statistische Robustheit, rechnerische Robustheit, Modellrobustheit und mehr, wobei die kritische Bedeutung jedes Aspekts im Gesamtkontext eines effektiven Portfoliomanagements hervorgehoben wird. Zudem setzt dieses Kapitel den Rahmen für die gesamte Dissertation, indem es eine umfassende Einführung in jeden einzelnen Aufsatz bietet und die Motivation, Notation und die wesentlichen Beiträge beschreibt.

Der Hauptteil der Dissertation setzt sich aus den beitragenden Aufsätzen zusammen. Der erste Aufsatz knüpft an das einflussreiche Portfoliokonstruktionsmodell von Harry Markowitz an und zeigt auf, wie praktisch relevante Nebenbedingungen sowie eine explizite Modellierung der Unsicherheiten in der Renditeprognose die vermeintlichen Schwächen des Ansatzes adressieren können. Der zweite Aufsatz führt eine domänenspezifische Sprache zur Vereinfachung der Lösung von konvex-konkaven Sattelpunktproblemen ein, welche direkte Anwendungen im Bereich der robusten Optimierung findet und ein neues Werkzeug für Industrie und Forschung darstellt. Der dritte Aufsatz widmet sich dem Bereich des robusten Anleihenportfolio-Managements und entwickelt innovative Methoden zur Bewertung und Optimierung von Anleihenportfolios unter verschiedenen Marktbedingungen. Der vierte Aufsatz beschäftigt sich mit computergestützten Methoden in der verhaltensorientierten Finanzwissenschaft. Er präsentiert neuartige Ansätze für die Portfoliooptimierung, die auf der cumulative prospect theory basieren, wodurch das Verhalten und die Präferenzen von Investoren effektiv in die Portfoliokonstruktion integriert werden und somit einen Beitrag zur Annäherung von Theorie und praktischem Investorenverhalten leisten. Der letzte Aufsatz vereint traditionelle statistische Methoden und moderne Deep-Learning-Techniken. Er führt die Autoregressive Moving Average (ARMA)-Zelle als eine neue Komponente für neuronales autoregressives Modellieren ein, was die verfügbaren Methoden für robuste Prognosen in verschiedenen Anwendungen, einschließlich der Finanzprognose, wesentlich erweitert.

Zusammenfassend leistet diese Dissertation Beiträge zu mehreren Aspekten der robusten Vermögensallokation. Sie bietet nicht nur umfassende theoretische Einsichten und praktische Anwendungen, sondern fördert auch eine kollaborative Umgebung für weitere Innovationen und Entwicklungen in diesem dynamischen Feld. Die Bereitstellung von Open-Source-Software zu jedem Forschungsbeitrag erhöht die praktische Relevanz der Arbeit erheblich, sodass neue Erkenntnisse und entwickelte Methodologien problemlos sowohl in akademischen als auch in industriellen Umgebungen angewendet und erweitert werden können, und somit das Potenzial haben, nachhaltige Auswirkungen auf das Feld zu haben.

Contents

Part I.

Introduction and Background

1. General introduction

The field of robust asset allocation is both fascinating and challenging, with many opportunities for innovation and advancement that can have a significant impact on the financial industry. This dissertation aims to contribute to the current understanding of robustness in asset allocation, by exploring and developing various theoretical frameworks, computational methods, and practical applications. To give the reader some context, this chapter starts by providing a brief motivation that highlights the importance of robustness in portfolio management, before discussing various forms of robustness relevant to asset allocation.

1.1. Motivation

Throughout the past century, much effort has been devoted to the development of portfolio management strategies that seek to provide superior risk-adjusted returns to their investors, dating back to the seminal work of Graham and Dodd (1934). Despite these efforts, the financial markets remain a thorny landscape to thrive in, with countless examples of failed portfolio management strategies. One such case is Long-Term Capital Management (LTCM), a hedge fund that, although founded by Nobel laureates and staffed with some of the brightest minds in finance, collapsed in 1998 after losing $4.6 billion in less than four months (Lowenstein, 2001). The collapse of LTCM was a wake-up call for the financial industry, highlighting the need for more robust portfolio management strategies. Other crises, such as the 2007-2008 financial crisis or the COVID-19 pandemic, further underscored the importance of robustness in portfolio management, as many portfolios suffered significant losses during these events. To this day, robust asset allocation strategies remain less well understood, and implementing them in practice remains a challenge, leaving room for further research and innovation.

At its core, robustness in portfolio management entails designing strategies that excel not just in expected scenarios but also maintain their effectiveness in the face of unexpected events, or at least, do not suffer catastrophic losses. This is a challenging task, as the financial world is complex and interconnected, with many sources of hard to quantify uncertainty. The comprehensive understanding of risk, extending beyond traditional financial metrics to include market, credit, operational, and systemic risks, is crucial in developing robust asset allocation strategies. Thus, any attempt to design robust asset allocation strategies must navigate these layers of complexity, ensuring portfolios are optimized for anticipated conditions while remaining resilient against the unknown.

Furthermore, the concept of robustness in asset allocation offers a unique intersection of disciplines, ranging from finance and economics to mathematics, statistics, computer science, and even behavioral psychology. This multidisciplinary nature of robustness in asset allocation makes it a fascinating and challenging field, with many opportunities for new research and innovation, some of which are explored in this dissertation.

This dissertation thus investigates the diverse interpretations of robustness in asset allocation, and explores various methods and techniques to design robust portfolio management strategies. From optimizing portfolio returns and managing risk to adapting to changing market conditions and investor preferences, the essays in this dissertation offer a multifaceted perspective on robustness in financial portfolio management.

Special attention is also given to the practical implementation of each contribution in this dissertation, with each essay accompanied by a software implementation that is publicly available as open-source code, with the link provided alongside the essay. The software implementations are designed to not only allow for reproducibility of the results but also to serve as a starting point for future research and innovation in the field of robust asset allocation.

1.2. Forms of robustness in asset allocation

In this dissertation, the concept of robustness is pivotal, and accordingly, we start by providing a definition of robustness in this setting. One interpretation of robustness is the ability of a system to maintain feature persistence when subjected to a wide range of perturbations (Jen, 2003). In asset allocation, feature persistence may refer to the ability of a portfolio to maintain positive returns, or to maintain a certain risk level, across a wide range of market regimes. This definition, albeit not universally agreed upon, serves as the basis for the following discussion of robustness in asset allocation. As robustness can take on various forms, each targeting specific facets of portfolio management, our discussion now turns to these pertinent aspects of robustness. These aspects are not mutually exclusive, and in fact, many of them are closely related, as will be discussed in the following.

Statistical robustness. Focusing on the resilience of statistical methods against model assumption violations, and in particular, deviations from the assumed distribution of the data or outliers (Huber and Ronchetti, 2009), statistical robustness is immediately relevant to portfolio management. For instance, a loss that would happen once in a thousand years according to normally distributed asset returns may be observed every few years in practice, motivating the use of more robust statistical methods, such as using a heavy-tailed distribution (Mittnik and Rachev, 1993). Robust statistics is a well-established field, with many methods and techniques, such as robust regression, robust covariance estimation, and robust testing (Huber and Ronchetti, 2009). One immediate application of robust statistics in the context of portfolio management is thus to use robust methods to estimate the expected returns and covariance matrix of the assets, which are crucial inputs to portfolio optimization models.

Robust optimization. Addressing uncertainties in input parameters within the model, robust optimization stands in contrast to traditional optimization, which relies on precise estimates of these parameters - often a challenging task in practice. Instead, robust optimization considers a range of plausible values for the input parameters, and seeks to optimize for various moments of the objective, such as the expected value or the worst-case value (Shapiro et al., 2021; Beyer and Sendhoff, 2007; Ben-Tal et al., 2009; El Ghaoui and Lebret, 1997). Theoretical and computational advancements in robust optimization have made the approach increasingly practical, with portfolio optimization being a prominent example, as it is known to be sensitive to input parameters.

Robust optimization is closely related to statistical robustness, as it can be used to address model assumption violations, and much of the literature on robust asset allocation focuses on robust optimization methods (*e.g.*, Fabozzi et al., 2007; Tütüncü and Koenig, 2004).

Computational robustness. The resilience of computational methods against numerical or algorithmic errors is an essential aspect of computational robustness. In portfolio management, this involves ensuring that the numerical methods used to solve the optimization problem are stable and reliable and that the results are not affected by numerical errors. Moreover, it may also involve following best practices in software engineering, such as using version control, automated testing, and continuous integration, to ensure the reliability of the software implementation of the portfolio management strategy. This process is simplified by the advancement of higher level programming languages and domain-specific languages, which allow for concise and expressive code that is easy to read and understand, and thus, less susceptible to errors.

Model robustness. As market dynamics evolve, shifts in the underlying assumptions of the portfolio management model may occur, rendering the model obsolete. Model robustness refers to the degree to which a model can adapt to changing market conditions, or to the degree to which the model remains valid in the face of structural changes in the market. For example, a model that exploits a small inefficiency in the market may become obsolete if the inefficiency is corrected, whereas a model that trades based on long-term economic trends may remain valid for a longer period of time.

Other forms of robustness. In addition to the forms of robustness discussed above, there are many other forms of robustness that are relevant to portfolio management. For example, operational robustness refers to the resilience of the portfolio management against process failures, such as human errors or technical failures, and is often closely related to computational robustness for quantitative strategies. A prominent example is the Knight Capital Group trading error in 2012, where faulty software and a lack of control systems led to a series of erroneous trades which cost the company $440 million (Davidson, 2012). Further, regulatory robustness aims to ensure that changes in regulatory requirements, such as capital requirements or trading restrictions, do not adversely affect the portfolio management strategy. Ethical robustness investigates the ethical implications of the strategy, ensuring that the strategy is aligned with the ethical values of the investors, with the recent rise of environmental, social, and governance (ESG) investing being a notable example. Finally, considering behavioral robustness is a crucial aspect of portfolio management, as evidently investors exhibit behavioral biases, such as loss aversion and overconfidence, which can lead to suboptimal investment decisions if not properly accounted for (Barber and Odean, 2013).

1.3. Outline of the dissertation

This dissertation comprises five essays, addressing various aspects of robustness in asset allocation. The remainder of this chapter provides a brief overview of each essay, highlighting its contributions and relevance to the field of robust asset allocation, as well as the notation and terminology used.

The subsequent chapters then include the essays in their entirety. The five essays that comprise this dissertation are:

1. **Markowitz Portfolio Construction at Seventy:** Revisiting Harry Markowitz's pioneering work, this essay extends his original model by incorporating practical constraints and addressing uncertainties in the expected returns and covariance matrix of the assets. It exemplifies robust optimization and model robustness in modern portfolio management.

2. **Disciplined Saddle Programming:** Shifting focus to convex-concave saddle point problems, this essay introduces a novel domain-specific language for specifying and solving such problems, with natural use cases being robust optimization in finance and related fields.

3. **Robust Bond Portfolio Construction via Convex-Concave Saddle Point Optimization:** Focusing on the bond market, this essay explores methods for computing the minimum value of a bond portfolio under various adverse yield curve and spread environments, as well as methods to construct portfolios that perform well in these worst-case settings.

4. **Portfolio Optimization with Cumulative Prospect Theory Utility via Convex Optimization:** This essay tackles the challenge of maximizing non-concave cumulative prospect theory utility (CPT) in portfolio selection. It contributes to behavioral robustness by providing a computational framework that can incorporate investor preferences and empirically observed behavioral patterns into portfolio construction.

5. **ARMA Cell: A Modular and Effective Approach for Neural Autoregressive Modeling:** Bridging the gap between traditional statistical methods, in this case autoregressive moving average (ARMA) models, and modern deep learning techniques, this essay introduces the ARMA cell, a novel neural network component that can effectively model a wide range of time series. It contributes to robust forecasting by providing a flexible and robust framework for modeling time series, which can be used in portfolio management and beyond.

2. Modern portfolio theory

A central challenge in portfolio management is the construction of robust portfolios that can consistently deliver high returns in the face of uncertainties. The seminal work by Harry Markowitz in 1952 laid the groundwork for modern portfolio theory (MPT), introducing an optimization framework that balances expected return against risk, defined as the variance of portfolio returns (Markowitz, 1952). Before Markowitz, portfolio construction was largely driven by heuristics and intuition, with little theoretical foundation and rigor. Markowitz had the vision of an optimization-based approach to portfolio management, which he realized through the development of the mean-variance optimization model. Over the decades, this framework has evolved significantly, adapting to advancements in computational power, optimization techniques, and practical investment constraints. Still, the method is commonly criticized, with a prime concern being its sensitivity to input data, especially the estimated return statistics (Michaud and Michaud, 2008). The essay revisits Markowitz's original model and shows how contemporary advancements in optimization and computational techniques can be used to address these concerns, thereby enhancing the robustness of the model. Building on the foundations of Markowitz's work, the model introduced in this essay is therefore termed "Markowitz++." In the scope of the dissertation, this essay serves to provide some historical context to portfolio optimization, highlighting the evolution of the field and the relevance of Markowitz's original insight in the context of modern portfolio management. Additionally, it introduces terminology and notation that will be used in most of the other essays.

2.1. Background and notation

The aim of portfolio construction methods is to determine the optimal holdings of a portfolio according to some objective, typically involving maximizing the expected return, and constraints. To construct a portfolio, we start with a set of n assets, called the investment universe.

Holdings and trades. The portfolio that we are allocating has a positive value V, and we can invest in these assets by allocating a fraction w_i of the portfolio value to the i-th asset, and thus $w \in \mathbf{R}^n$ is the vector of asset weights. Any remaining fraction of the portfolio value is held in cash, denoted by c, leading to the identity

$$\mathbf{1}^T w + c = 1,$$

where $\mathbf{1}$ is the vector of all ones of the appropriate dimension. When $w_i > 0$, we say that the portfolio has a long position in the i-th asset, and when $w_i < 0$, we say that the portfolio has a short position in the i-th asset. Similarly, when $c > 0$, the portfolio is holding cash and is referred

to as *diluted* and when $c < 0$, the portfolio is borrowing cash, and is referred to as *margined*. Closely related is the notion of *leverage*, which we take to be

$$L = \sum_{i=1}^{n} |w_i| = \|w\|_1,$$

noting that other definitions of leverage are also used.

We change the portfolio holdings by executing *trades*

$$z = w - w^{\text{pre}},$$

where w^{pre} represents the pre-trade weights. We use *turnover*, to quantify the volume of the trades, which we define as

$$T = \frac{1}{2}\sum_{i=1}^{n} |z_i| = \frac{1}{2}\|z\|_1.$$

Costs and constraints. One way of making the portfolio robust is by making the model more realistic, *i*.e., by taking into account practical considerations. In practice, there are several implicit and explicit costs associated with our allocation and trading decisions. We can therefore model holding and trading costs, ϕ^{hold} and ϕ^{trade}, where ϕ^{hold} should take into account the cost of shorting assets and borrowing cash, and ϕ^{trade} should take into account the bid-ask spread of the assets, as well as the *market impact*, *i*.e., the change in price caused by the trade itself. There are also constraints that portfolio managers are facing, which could be external, such as regulatory constraints, or internal, such as risk limits. The model should thus allow for constraints on the portfolio holdings and trades, such as limits on the leverage, the turnover, or bounds on the individual asset weights, cash holdings, or trades. The semantics of constraints are that they are *hard*, *i*.e., no violation is tolerated. However, instead of constraints, we can also add penalization terms to the objective to disincentive allocations with undesirable properties, while keeping the problem feasible. In some cases we only want to penalize after a certain threshold, prompting the use of *soft* constraints, where instead of the constraint

$$f \leq f^{\max},$$

we add the term

$$\gamma(f - f^{\max})_+,$$

to the objective, where γ is a positive scalar, and $(x)_+ = \max\{x, 0\}$. Conveniently, if f is a convex function, then so is $(f - f^{\max})_+$ by the composition rules for convex functions (Boyd and Vandenberghe, 2004, §3.2.4). The semantics of soft constraints are therefore that they add no penalty if $f \leq f^{\max}$, and a penalty tunable by the parameter γ if $f > f^{\max}$.

Risk and return. The Markowitz method aims to trade off the expected return and risk of the portfolio. The gross portfolio return, *i*.e., before deducting any trading and holding costs, is given by $R = r^T w + r^{\text{rf}} c$, where $r \in \mathbf{R}^n$ is the vector of asset returns, and r^{rf} is the risk-free rate. While the returns of assets are not random, but rather determined by the orders of the market participants, it is often useful to model them as a multivariate random variable, with expected value $\mu \in \mathbf{R}^n$ and covariance matrix $\Sigma \in \mathbf{S}^n_{++}$, *i*.e., a symmetric positive definite matrix. The

portfolio return is then a random variable with expected value $\bar{R} = \mu^T w + r^{\text{rf}} c$ and variance $\sigma^2 = w^T \Sigma w$. The essay makes no attempt at suggesting a model for the expected returns, as it is by the nature of the problem the most difficult task, with successful models often being closely guarded secrets. While the covariance matrix is comparatively easier to estimate, it is still a challenging task, with many models and techniques available (Johansson et al., 2023). One common approach, especially with large universes, is to use a factor model, where the return of the n assets is expressed in terms of a smaller number of underlying factors. This relation is expressed by the factor loading matrix $F \in \mathbf{R}^{n \times k}$. One way of using the factor approach is to model the first two moments of the asset returns as

$$\mu = F\bar{f} + \bar{\epsilon} \quad \text{and} \quad \Sigma = F\Sigma^f F^T + D,$$

with $\bar{f} \in \mathbf{R}^k$ and $\Sigma^f \in \mathbf{S}^k_{++}$ being the expected factor returns and the factor covariance matrix, respectively. This model allows expressing idiosyncratic risk and return contributions through the vectors $\bar{\epsilon} \in \mathbf{R}^n$ and $D \in \mathbf{S}^n$, where the latter is a diagonal matrix. Using a factor model can not only improve the out of sample performance of the model (Feng et al., 2020), but also greatly speeds up solving the optimization problem, as demonstrated in the essay.

Robust optimization. The estimation of μ and Σ is subject to considerable uncertainty, which can be amplified by the optimization process. These challenges can be addressed by using robust optimization techniques, which seek to construct portfolios that perform well, either measured by the expected value or the worst-case of the objective, across a spectrum of plausible outcomes, expressed as a set of scenarios or as a probability distribution (Shapiro et al., 2021; Beyer and Sendhoff, 2007; Ben-Tal et al., 2009; El Ghaoui and Lebret, 1997). In this essay, we follow Boyd et al. (2017) and use the worst-case over a convex set of possible values, defined as

$$R^{\text{wc}} = \min\{(\mu + \delta)^T w \mid |\delta| \leq \rho\}, \tag{2.1}$$

for the expected portfolio return, and

$$(\sigma^{\text{wc}})^2 = \max\{w^T(\Sigma + \Delta)w \mid |\Delta_{ij}| \leq \varrho(\Sigma_{ii}\Sigma_{jj})^{1/2}\}, \tag{2.2}$$

for the portfolio variance, where $\delta \in \mathbf{R}^n$ and $\Delta \in \mathbf{S}^n$ are perturbations to the expected return and covariance matrix, respectively, and $\rho \in \mathbf{R}^n_+$ and $\varrho \in (0, 1]$ are parameters that control the size of the perturbations. These specific formulations have analytical solutions, making them easy to implement. Anthropomorphizing the optimization problem, the semantics are that we can choose the weights of the assets, but an adversary can choose the perturbations, so we wish to maximize the return in the worst-case scenario.

2.2. The Markowitz model

In his original paper, Markowitz geometrically illustrated the risk-return trade-off characterizing the optimization problem

$$\begin{array}{ll} \text{maximize} & \mu^T w \\ \text{subject to} & w^T \Sigma w \leq (\sigma^{\text{tar}})^2, \\ & \mathbf{1}^T w = 1, \end{array}$$

where the portfolio return is maximized subject to a variance and budget constraint. The Markowitz++ model, as introduced in this essay, extends the original model by introducing additional constraints and costs, leading to a much more robust portfolio. Still, we can easily implement and solve it in practice by modeling it as a convex optimization problem.

Convex optimization. All objective terms and constraints in the Markowitz++ model are convex. Far from being a mere technicality, this is a crucial property that allows us to solve the model efficiently and reliably. When Markowitz first introduced his model, the simplex algorithm for linear programs (LPs) was just introduced (Dantzig, 1951), with the first solver for quadratic programs (QPs) being developed a few years later (Wolfe, 1959). Today, convex optimization is a mature field, encompassing not only LPs and QPs but also more general convex optimization problems like second-order cone programming (SOCP) and semidefinite programming (SDP). These problem classes allow us to model a wide range of practical applications, including portfolio optimization, where we use SOCPs to directly model the volatility instead of the variance, and SDPs can be used to model covariance matrices. The theory surrounding convex optimization is also well-developed, ranging from sensitivity analysis via duality theory to convergence guarantees (Boyd and Vandenberghe, 2004). Moreover, reliable solvers and algorithms are readily available as both commercial (*e.g.*, Gurobi Optimization, LLC, 2023; MOSEK ApS, 2020; Cplex, IBM ILOG, 2009) and open-source (*e.g.*, Goulart and Chen, 2024; Domahidi et al., 2013; O'Donoghue et al., 2016; Stellato et al., 2020) software.

Domain-specfic languages. Once the problem is formulated mathematically, we can turn to implementing it in software, where we leverage the power of modern programming languages and domain-specific languages (DSLs). Using a DSL like CVXPY (Diamond and Boyd, 2016), the software used throughout this dissertation to specify convex problems, we can express the problem in code that closely resembles the mathematical formulation. For example, even without previous experience with the Python programming language or CVXPY, the following code snippet can easily be understood to solve the original Markowitz model for `n` assets, given expected return vector `mu` and covariance matrix `Sigma`, and a volatility limit `sigma_tar`.

```python
import cvxpy as cp

w = cp.Variable(n)
objective = cp.Maximize(mu.T @ w)
constraints = [w.T @ Sigma @ w <= sigma_tar**2, cp.sum(w) == 1]
problem = cp.Problem(objective, constraints)
problem.solve()
```

In addition to the conciseness of the code, leveraging the properties of convex optimization allows us to solve the problem without worrying about the algorithmic details of the solver in most cases, as indicated by the `solve` method not requiring any arguments. DSLs for convex optimization also exist in other programming languages, such as Convex.jl (Udell et al., 2014) for Julia, CVXR (Fu et al., 2017) for R, and CVX (Grant and Boyd, 2014) for MATLAB.

3. Saddle programming

Convex optimization is a powerful tool in portfolio management, allowing us to model a broad range of practical problems, as shown in the previous essay. It also became apparent that we can improve the portfolio construction process by incorporating robust optimization techniques. In this essay, we explore convex-concave saddle problems, a class of optimization problems that is widely applicable in finance, machine learning, game theory, and other fields. One natural use case of saddle problems is robust optimization, as saddle problems can be used to model worst-case optimization problems. While specialized methods exist to solve some classes of saddle problems, one common approach is to dualize the problem, *i*.e., to transform the saddle problem into a single convex optimization problem. Traditionally, this is a manual expertise-driven process that is often tedious and prone to errors. This essay introduces a novel approach to automate and simplify this process, encapsulating it within the framework of *disciplined saddle programming* (DSP). This approach leverages recent advances in the theory of conic representable saddle functions by Juditsky and Nemirovski (2022), and introduces a DSL that can parse saddle problems, automatically dualize them, solve the resulting convex problem, and return the solution to the original problem. The fact that it is disciplined implies that a small set of rules needs to be followed that provide sufficient conditions for the representability of the problem. The essay also contributes insights that are relevant for solving saddle problems in practice, as DSP ensures that even in cases where certain technical conditions (like compactness) are not explicitly verified, the solutions obtained are feasible and valid. This approach underscores the robustness and reliability of DSP in solving a broad spectrum of saddle problems.

3.1. Background and notation

To understand the motivation behind DSP, we first need to understand the structure of saddle problems, and how they can be solved using convex optimization. We start by introducing the notation used in this essay.

Saddle problems. A convex-concave saddle problem is an optimization problem that involves a *saddle function*, *i*.e., a function that is convex in one set of variables and concave in another set of variables, either in the objective or in the constraints. Formally, a saddle function is a function $f : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, with $\mathcal{X} \subseteq \mathbf{R}^n$ and $\mathcal{Y} \subseteq \mathbf{R}^m$, that is a convex function in $x \in \mathcal{X}$ for any fixed $y \in \mathcal{Y}$, and a concave function in $y \in \mathcal{Y}$ for any fixed $x \in \mathcal{X}$. The domain of the saddle function is the Cartesian product $\mathcal{X} \times \mathcal{Y}$, which we require to be a nonempty closed convex set. A *saddle point* is a point $(x^*, y^*) \in \mathcal{X} \times \mathcal{Y}$ satisfying

$$f(x^\star, y) \leq f(x^\star, y^\star) \leq f(x, y^\star) \text{ for all } x \in \mathcal{X},\ y \in \mathcal{Y}.$$

That is, choosing any $y \in \mathcal{Y}$ that is not y^* can only yield a value that is less or equal than $f(x^*, y^*)$, and similarly for x. Since we are interested in minimizing over x and maximizing over y, the components of (x^*, y^*) are the corresponding best responses to each other.

A *saddle point problem* is to find a saddle point of a saddle function over a convex feasible set. The more general class of *saddle problems* additionally includes problems involving functions defined as the partial supremum or infimum of a saddle function, *i.*e., a *saddle extremum function*. The saddle max function $G : \mathcal{X} \to \mathbf{R} \cup \{\infty\}$ and the saddle min $H : \mathcal{Y} \to \mathbf{R} \cup \{-\infty\}$ are defined as

$$G(x) = \sup_{y \in \mathcal{Y}} f(x, y) \quad \text{and} \quad H(y) = \inf_{x \in \mathcal{X}} f(x, y),$$

respectively. The saddle max function is convex, and the saddle min function is concave (Boyd and Vandenberghe, 2004, §3.2.3), and we can therefore use them in the objective or constraints of a convex optimization problem.

Conic optimization. A conic form problem is an optimization of the (standard) form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & x \succeq_K 0, \\ & Ax = b, \end{array} \tag{3.1}$$

where $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{p \times n}$, $b \in \mathbf{R}^p$, and $K \subseteq \mathbf{R}^n$ is a proper cone (Boyd and Vandenberghe, 2004, §4.6.1), with proper cones being defined as being closed, convex, pointed, and with nonempty interior (Boyd and Vandenberghe, 2004, §2.4.1). Conic form problems are a subclass of convex optimization problems and look similar to linear programs, and indeed, when $K = \mathbf{R}^n_+$, we recover the standard form of linear programs. However, conic form problems are much more general, allowing us to model a wide range of practical problems, including QPs, SOCPs, and SDPs. In most practical cases, the cone K is a Cartesian product of primitive cones, *i.*e., $K = K_1 \times \cdots \times K_n$, where each K_i is a proper cone. As an example, we can model a Markowitz portfolio optimization problem with a volatility instead of a variance constraint and a long-only constraint as a conic optimization problem. We start with the problem

$$\begin{array}{ll} \text{maximize} & \mu^T w \\ \text{subject to} & \|L^T w\|_2 \leq \sigma^{\text{tar}}, \\ & \mathbf{1}^T w = 1, \\ & w \geq 0, \end{array}$$

where $\mu \in \mathbf{R}^n$ is the expected return vector, $L \in \mathbf{R}^{n \times n}$ is the Cholesky factor of the covariance matrix, and $\sigma^{\text{tar}} \in \mathbf{R}_+$ is the target volatility. The inequality for the long-only constraint here is elementwise. We can rewrite the problem in standard form by introducing auxiliary variables $t \in \mathbf{R}$ and $y \in \mathbf{R}^n$ and instantiating (3.1) with

$$c = \begin{bmatrix} -\mu \\ 0 \\ \mathbf{0} \end{bmatrix}, \quad A = \begin{bmatrix} \mathbf{0}^T & 1 & \mathbf{0}^T \\ -L^T & 0 & I_n \\ \mathbf{1}^T & 0 & \mathbf{0}^T \end{bmatrix}, b = \begin{bmatrix} \sigma^{\text{tar}} \\ \mathbf{0} \\ 1 \end{bmatrix} \quad \text{and} \quad x = \begin{bmatrix} w \\ t \\ y \end{bmatrix},$$

where $\mathbf{0}$ is the vector of all zeros of appropriate dimension. The cone K is given $K = \mathbf{R}^n_+ \times \mathcal{Q}^{n+1}$, where $\mathcal{Q}^{n+1}$ is the second-order cone in $\mathbf{R}^{n+1}$, defined as

$$\mathcal{Q}^{n+1} = \{(t, y) \in \mathbf{R} \times \mathbf{R}^n \mid \|y\|_2 \leq t\}.$$

While forming the standard form manually has a didactic value, it is rarely necessary in practice, as DSLs like CVXPY automate this process. The generality of conic optimization makes it useful as an interface between solvers and modeling frameworks, as it allows users to express a wide range of problems that eventually can all be reduced to a conic standard form problem. Conic solvers can then be used to solve the problem using efficient algorithms that work across different cone types, such as interior point methods (Potra and Wright, 2000). Finally, the conic form can be used to generate theoretical insights, such as differentiating the optimal value of the problem with respect to parameters (Amos, 2019; Agrawal et al., 2019), or deriving a dual reduction for conic representable saddle functions (Juditsky and Nemirovski, 2022), which is the basis of DSP.

It is therefore not surprising that conic optimization has become the dominant paradigm in convex optimization, with many solvers and DSLs available.

Dual reduction. The idea behind dual reduction is to transform a saddle problem into a single convex optimization problem, which goes back to Morgenstern and Von Neumann (1953). As an example, we consider the problem of maximizing the worst-case return of a portfolio (2.1), subject to a budget constraint, given by

$$\begin{array}{ll} \text{maximize} & \inf_{\delta \in \mathcal{D}} \delta^T w + \mu^T w \\ \text{subject to} & \mathbf{1}^T w = 1, \end{array}$$

with $\mathcal{D} = \{\delta \in \mathbf{R}^n \mid |\delta| \leq \rho\}$ and variables $w \in \mathbf{R}^n$ and $\delta \in \mathbf{R}^n$. For any w, the saddle extremum function in the objective evaluates to the optimal value of the LP

$$\begin{array}{ll} \text{minimize} & w^T \delta \\ \text{subject to} & -\delta \leq \rho, \\ & \delta \leq \rho, \end{array}$$

with variable $\delta \in \mathbf{R}^n$. The dual of this LP is given by

$$\begin{array}{ll} \text{maximize} & \rho^T \lambda_1 + \rho^T \lambda_2 \\ \text{subject to} & \lambda_1 - \lambda_2 = w, \\ & \lambda_1 \geq 0, \lambda_2 \geq 0, \end{array}$$

with dual variables $\lambda_1, \lambda_2 \in \mathbf{R}^n$. Assuming strong duality holds, the optimal value of both problems is equal, and we can therefore replace the saddle extremum function with the dual objective, leading to the single convex optimization problem

$$\begin{array}{ll} \text{maximize} & \rho^T \lambda_1 + \rho^T \lambda_2 + \mu^T w \\ \text{subject to} & \lambda_1 - \lambda_2 = w, \\ & \lambda_1 \geq 0, \lambda_2 \geq 0, \\ & \mathbf{1}^T w = 1, \end{array}$$

with variables w, λ_1, λ_2. While it is easy to derive the dual reduction for this simple example by hand, the process quickly becomes tedious for complex problems, motivating the development of DSP.

3.2. Disciplined saddle programming

We now turn to the main contribution of this essay, DSP, a novel approach to automate the process of dualizing saddle problems. We refer to the DSL, as well as its concrete implementation in Python, as DSP.

Atoms and composition. The core idea behind DSP is to automate the process of dualizing saddle problems, encapsulating it within a DSL. The DSL is based on the concept of *atoms*, representing convex-concave saddle functions. Those functions include bilinear and bi-affine functions, weighted norms, quadratic forms, and others. These atoms can be composed to form more complex functions, which are also convex-concave saddle functions. For example, scaling a saddle function with a positive scalar yields a convex-concave saddle function, as does adding two saddle functions. When negating a saddle function, we obtain a saddle function with the roles of the convex and concave variables switched.

Developing a DSL. DSP is inspired by the principles of disciplined convex programming (DCP), extending its scope to encompass saddle problems. Just like DSLs for convex optimization (such as CVXPY) abstract away the intricacies of canonicalizing convex programming models to a standard form, DSP automates the dualization process, thereby greatly simplifying the specification of saddle problems. As an example, we can easily express (2.1) in DSP as shown below, with the full problem and the exact syntax (*e.g.*, the notion of local variables) being explained in detail in the essay. While for this specific case, an analytic expression is available, the DSP formulation is much more general, allowing for arbitrary DCP representable constraint sets.

```
import cvxpy as cp
import dsp

w = cp.Variable(n)
delta = dsp.LocalVariable(n)

f = dsp.inner(w, mu + delta)
constraints = [cp.abs(delta) <= rho]

R_wc = dsp.saddle_max(f, constraints)
```

In this example, `R_wc` is a convex function that can be used in any CVXPY problem, underlining the tight integration of DSP with CVXPY. The implementation also introduces a new Problem class `dsp.SaddlePointProblem`, which can be used to solve saddle point problems directly, without the need to formulate the objective as a saddle max or saddle min function.

4. Bond portfolio construction

While equities often take the spotlight in the discourse around portfolio management, bonds are a crucial component of many investment strategies. Indeed, the bond market even surpasses equities in size, with the global bond market being valued at $130 trillion in 2022, compared to $101 trillion for the global equity market (Kolchin et al., 2023).

While commonly perceived as less risky, constructing bond portfolios comes with its own set of challenges. A recent example highlighting the importance of robust bond portfolio construction is the collapse of the Silicon Valley Bank, which, among other factors, invested in longer term bonds to increase its yield, but was ultimately forced to sell its assets at a loss to meet its obligations when US interest rates increased in 2022 (Yousaf and Goodell, 2023). This essay contributes to the field of bond portfolio management by introducing a framework that not only allows for an assessment of the worst-case of a given bond portfolio over a set of yield curves and credit spreads, but also provides a methodology for constructing robust bond portfolios that maximize the worst-case value of the portfolio.

4.1. Background and notation

We first introduce the notation used in this essay, in particular, the definition of bonds and bond portfolios, yields, and spreads.

Bond portfolios. We define a *bond* as a financial contract defining a series of specified payments over time, represented by a vector $c \in \mathbf{R}^T$, where T denotes the number of periods. Each component c_t of this vector corresponds to the net cash flow in period t to the bondholder. These payments may include coupon payments, which are periodic interest payments, and the principal payment, which is the final payment at maturity, *i*.e., the last period in which a cash flow occurs.

A *bond portfolio* is a collection of n bonds, with each bond having a maturity of at most T. We represent holdings in a bond portfolio by a vector $h \in \mathbf{R}^n$, where h_i denotes the number of units of bond i held in the portfolio. We assume that the portfolio is long-only, *i*.e., $h_i \geq 0$ for all bonds. The prices of the bonds are collected in a vector $p \in \mathbf{R}^n_+$, resulting in the total value of the portfolio being $V = p^T h$.

Yield curves and spreads. The price of a bond can be modeled as the discounted value of its cash flows, *i*.e.,

$$p_i = \sum_{t=1}^{T} c_{i,t} \exp(-t(y_t + s_i)), \quad i = 1, \ldots, n,$$

where we use continuous compounding for simplicity, but the results can be applied to periodic compounding as well. Here, we observe that the discount factor is a function of the yield curve $y \in \mathbf{R}^T$ and the credit spread $s \in \mathbf{R}^n_+$, that is, the current value of the future cash flow depends on a time component as well as an idiosyncratic component for each bond. The yield curve y is derived from traded bonds, with a common approach being to fit a smooth curve to the observed yields, *e*.g., using a Nelson-Siegel model (Nelson and Siegel, 1987), but more complex models are also used (Filipović et al., 2022). The credit spread s is a measure of the risk associated with the individual bond, and depends on factors such as the credit rating of the issuer or the liquidity of the bond.

4.2. Robust bond portfolios

The main contribution of the essay is to develop worst-case evaluation and robust bond portfolio construction methods. For this, we first note that the value of a bond portfolio

$$V = p^T h = \sum_{i=1}^{n} \sum_{t=1}^{T} h_i c_{i,t} \exp(-t(y_t + s_i))$$

is a convex function of the yield curve y and the credit spread s for fixed holdings h. It also is an affine, and thus concave, function of the holdings h for fixed y and s. We often work with the logarithm of the portfolio value for convenience, and indeed, this composition remains convex in y and s and concave in h. This property also trivially holds for the linearized change in the portfolio value based on the first-order Taylor approximation, a common approach used in practice that serves as a reference for the exact methods.

Having established these convexity properties, we can use convex optimization to solve the worst-case evaluation problem, and saddle point optimization to solve the robust portfolio construction problem.

Worst-case portfolio value. The worst-case portfolio problem minimizes the portfolio value over a set of yield curves and credit spreads, *i*.e.,

$$\begin{array}{ll} \text{minimize} & \log(V/V^{\text{nom}}) \\ \text{subject to} & (y, s) \in \mathcal{U}, \end{array}$$

where V^{nom} is the nominal portfolio value, *i*.e., the value of the portfolio at the current yield curve and credit spreads, and $\mathcal{U}$ is the convex uncertainty set of yield curves and credit spreads. This set can be derived from historical data, forecasts, uncertainty quantification of the yield curve and credit spread models, or other sources. The worst-case portfolio value is a convex function of the yield curve and credit spread. It can readily be modeled using only a few lines of code in CVXPY, as shown in the essay, and solved efficiently even for large portfolios.

Robust bond portfolio construction. A simple bond portfolio construction problem would be to maximize the worst-case portfolio value over a set of feasible portfolios $\mathcal{H} \subset \mathbf{R}^n_+$. While this might be intended in some cases, it can lead to portfolios that are extremely conservative, *e*.g., by holding only money market instruments or cash. Instead, we often have a nominal objective

function $\phi : \mathbf{R}^n_+ \to \mathbf{R}$ that we wish to minimize, which can include the negative expected return, the variance, tracking error with respect to a benchmark, or other components. We can then trade off the nominal objective with the worst-case portfolio value by introducing a regularization parameter $\lambda > 0$, leading to the robust bond portfolio construction problem

$$\begin{array}{ll} \text{minimize} & \phi(h) - \lambda \min_{(y,s)\in\mathcal{U}} \log(V/V^{\text{nom}}) \\ \text{subject to} & h \in \mathcal{H}. \end{array}$$

This problem is a convex-concave saddle problem, and can either be solved by explicit dualization for a given uncertainty set $\mathcal{U}$, or by using the DSP framework introduced in the previous essay.

5. Cumulative prospect theory optimization

One aspect of portfolio optimization that has received increasing attention in recent years is the incorporation of behavioral aspects into the modeling process. While traditional models assume rational investor behavior, behavioral models suggest that investors often behave irrationally in the face of uncertainty, leading to suboptimal decisions. Cumulative prospect theory (CPT), developed by Tversky and Kahneman (1992), is a pivotal development in this area, offering a more realistic depiction of investor behavior by incorporating aspects like loss aversion and the overweighting of low-probability events. This essay presents novel computational methods for portfolio optimization using CPT utility, which, unlike other approaches, accommodates the complexities and idiosyncrasies of human decision-making in financial markets.

While simple utility functions such as quadratic utility, solved by the Markowitz model, are easy to optimize by today's standards, CPT utility still poses computational challenges due to its nonconvexity. This essay addresses these challenges by introducing efficient optimization methods that leverage the subtle convexity properties of CPT utility, enabling the construction of portfolios that align more closely with actual investor preferences. As such, the essay contributes to robust portfolio optimization by allowing to incorporate behavioral aspects into the modeling process, providing a richer choice of utility functions.

5.1. Background and notation

Before introducing the optimization methods, we first provide the necessary background on CPT and its application to portfolio optimization. CPT diverges from the conventional von Neumann-Morgenstern (VNM) utility theory (von Neumann et al., 1944) by incorporating a value function that is concave for gains and convex for losses ("S-shaped"), reflecting the phenomenon of loss aversion. Moreover, it employs a probability weighting function that overweighs small probabilities, capturing the human tendency to overreact to low-probability events.

CPT utility. We begin by defining the CPT utility function. For this, we first define a *prospect theory* utility function u as a function of wealth relative to a reference point that has a greater marginal utility loss for losses than marginal utility gain for gains and is concave for gains and convex for losses. In the context of this essay, we use

$$u^{\text{prosp}}(x) = \begin{cases} 1 - \exp(-\gamma_+ x) & \text{if } x \geq 0 \\ -1 + \exp(\gamma_- x) & \text{otherwise} \end{cases},$$

which is prospect theory utility function when $\gamma_- > \gamma_+ > 0$. The second component of CPT utility is the probability weighting function that assigns a higher weight to extreme outcomes. An extreme outcome is one where our portfolio of n assets with weights $w \in \mathbf{R}^n$ has a return

that is either very high or very low. Letting $r_1, \ldots, r_N \in \mathbf{R}^n$ denote the asset returns across N periods, we obtain portfolio returns for each period as $r_i^T w$, $i = 1, \ldots, N$, or as the vector Rw, where $R \in \mathbf{R}^{N \times n}$ is the matrix of returns. We can partition these returns into N_- negative and N_+ nonnegative returns. The vectors $\pi_+ \in \mathbf{R}_+^N$ and $\pi_- \in \mathbf{R}_+^N$ jointly represent the probability of each observation occurring, with $\pi_+^T \mathbf{1} + \pi_-^T \mathbf{1} = 1$. The vectors are constructed as $\pi_+ = (0_{N_-}, \pi'_+)$ and $\pi_- = (0_{N_+}, \pi'_-)$, where both π'_+ and π'_- are nonnegative and nondecreasing, with the exact definition deferred to the essay. To give the most extreme outcomes a higher weight, we obtain the weighted sum by applying the function

$$f_\pi(x) = \sum_{i=1}^{N} \pi_i x_{(i)}$$

where $x_{(i)}$ is the i-th smallest component of x, which is a convex function. The CPT utility function is then defined as

$$U^{\text{cpt}}(w) = \underbrace{f_{\pi_+}(\phi_+(}_{\text{convex}} \underbrace{1 - \exp(-\gamma_+ Rw)}_{\text{concave}})) - \underbrace{f_{\pi_-}(\phi_-(}_{\text{convex}} \underbrace{-1 + \exp(\gamma_- Rw)}_{\text{convex}})),$$

with $\phi_+ = \max(x, 0)$ and $\phi_- = -\min(x, 0)$. Recognizing that this is a difference of two convex functions, each composed with a concave function is crucial for the development of the optimization methods.

CPT utility optimization. We define the CPT utility portfolio optimization problems as

$$\begin{array}{ll} \text{maximize} & U^{\text{cpt}}(w) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array}$$

where U^{cpt} is the CPT utility function, $w \in \mathbf{R}^n$ are the portfolio weights, and $\mathcal{W}$ is the set of feasible portfolios, which we require to be convex and DCP representable. Because the CPT utility function is non-convex, this problem is non-convex as well.

5.2. Optimization methods

The central contribution of this essay is the development of optimization methods that effectively handle the non-convexity of the CPT utility function in portfolio optimization. These methods are grounded in the recognition that, despite its non-convexity, the CPT utility function possesses convexity properties that can be exploited to construct efficient optimization algorithms.

Minorization-maximization method. The first method presented is the Minorization-Maximization (MM) algorithm, which iteratively constructs a concave lower bound (minorant) for the CPT utility and maximizes it. This approach bypasses the direct maximization of the non-convex CPT utility, instead focusing on a surrogate problem that is more tractable yet closely approximates the original problem. The MM algorithm can handle arbitrary DCP-compliant constraints and is therefore well-suited for complex optimization problems.

Convex-concave method. The second method, the Convex-Concave procedure (CC), leverages the convex-concave structure of the CPT utility. It iteratively linearizes the convex terms of the CPT utility and maximizes this local approximation of the function, allowing for efficient optimization even in the presence of complex portfolio constraints. Indeed, like the MM algorithm, the CC procedure also handles arbitrary DCP-compliant constraints.

Gradient ascent. Finally, the Gradient Ascent (GA) method is introduced for large-scale problems. This method utilizes gradient-based optimization techniques that are well-suited for problems with a large number of assets. The GA method is implemented using automatic differentiation and modern computational frameworks, in this case, PyTorch (Paszke et al., 2019). It is particularly well-suited for scenarios where the portfolio constraints are relatively simple.

In conclusion, this essay not only introduces novel computational methods to tackle the challenge of portfolio optimization under CPT utility but also significantly contributes to the broader field of financial decision-making, where nonconvex optimization problems frequently arise. The methodologies developed here have the potential to be extended to other areas of computational finance and economics, where similar challenges are encountered.

6. Neural network based time series forecasting

Recent advancements in time series analysis have seen a surge in the adoption of deep learning techniques, from fields like finance and economics (Li and Ma, 2010) to meteorology (Hsieh and Tang, 1998) and retail demand forecasting (Kochak and Sharma, 2015). However, despite the progress, traditional linear autoregressive models like the ARMA (Autoregressive Moving Average) model still hold significant relevance, especially in scenarios involving smaller datasets or lower signal-to-noise ratios. In such contexts, complex models like recurrent neural networks (RNNs), while powerful and expressive, often fail to outperform simpler alternatives, either due to overfitting or training difficulties. This essay introduces the ARMA cell, a novel neural network component that integrates the simplicity and effectiveness of traditional ARMA models with the flexibility and scalability of deep learning frameworks. The ARMA cell offers a more nuanced approach to time series modeling, balancing complexity with practicality and opening new avenues for efficient forecasting in various applications, demonstrated in a series of numerical experiments in the essay.

6.1. Background and notation

Before introducing the ARMA cell, we first provide the necessary background by defining the notation used in this essay, as well as introducing the ARMA model and recurrent neural networks. Time series data can be broadly categorized into univariate ($x_t \in \mathbf{R}$), multivariate ($\mathbf{x}_t \in \mathbf{R}^k$), and tensor-variate ($\mathbf{X}_t \in \mathbf{R}^{N_1 \times \ldots \times N_d}$) types, with observations indexed by $t = 1, \ldots, T$. There are a variety of models that can be used to describe the dynamics of time series data, including the ARMA model and recurrent neural networks, which we introduce below.

ARMA and VARMA models. The ARMA(p, q) model, going back to Box et al. (2015), describes a univariate time series as a linear combination of its past values and past error terms. For $p, q \in \mathbf{N}_0$, the model is defined as

$$x_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j} + \varepsilon_t,$$

with α, β_i, and γ_j being the model parameters, and ε_t denoting the independent and identically distributed (i.i.d.) error terms with variance σ^2. This model captures both autoregressive and moving average dynamics, making it versatile for various forecasting scenarios. Extending this concept, the Vector Autoregressive Moving Average (VARMA) model adapts the ARMA principles to multivariate time series, introducing matrix-valued parameters to model interactions between multiple time series components. The VARMA(p, q) model is defined correspondingly as

$$\mathbf{x}_t = \boldsymbol{\alpha} + \sum_{i=1}^{p} \mathbf{B}_i \mathbf{x}_{t-i} + \sum_{j=1}^{q} \boldsymbol{\Gamma}_j \boldsymbol{\varepsilon}_{t-j} + \boldsymbol{\varepsilon}_t,$$

with parameters $\boldsymbol{\alpha} \in \mathbf{R}^k$, $\mathbf{B}_i \in \mathbf{R}^{k \times k}$, $\boldsymbol{\Gamma}_j \in \mathbf{R}^{k \times k}$, and error terms $\boldsymbol{\varepsilon}_t \in \mathbf{R}^k$ again being i.i.d. with covariance matrix $\boldsymbol{\Omega}$. It is further possible to extend the family of linear autoregressive moving average models to tensor-variate time series, although less common in practice. A crucial assumption of these models is that the time series is stationary, *i*.e., the mean and variance of the time series do not change over time. This assumption is often violated in practice, motivating the development of models that can handle non-stationary time series, like the ARIMA model, which aims to achieve stationarity by applying differencing operations to the time series.

Recurrent neural networks. Recurrent neural networks (RNNs) are a class of neural networks that are designed to process sequential data, making them particularly well-suited for applications in natural language processing, speech recognition, and time series forecasting. RNNs are characterized by their recurrent structure, which allows them to access their internal state from a previous time step and thus capture the temporal dependencies in the data. However, RNNs are notoriously difficult to train, with the vanishing gradient problem being a common challenge. While mitigated by the introduction of gated architectures like LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014), RNNs still remain comparatively difficult to train, as demonstrated in the numerical experiments, with the simpler ARMA cell based model showing more robust performance.

6.2. The ARMA cell

Building on the foundation of the ARMA model, the ARMA cell is a unique adaptation designed for integration into neural networks, particularly RNN architectures. It is based on a reformulation of the ARMA model dependent on previous observations and previous forecasts, instead of previous observations and previous errors. We show that this reformulation is indeed only a reparametrization of the ARMA model that is equivalent to the original formulation, but allows for a more natural integration into existing deep learning frameworks.

Multi-unit and stacked cells. Much like a single unit in a neural network can be viewed as a linear model with non-linear activation that, when combined with additional randomly initialized units can approximate any continuous function (Cybenko, 1989), the ARMA cell is a linear time series model with non-linear activation that, when combined accordingly, can model complex time series dynamics. This key advantage is unlocked by implementing the ARMA cell as a neural network component, allowing for easy integration into existing deep learning frameworks. Additionally, using hyperparameter tuning techniques, this approach can be used to find an architecture that is optimal for a given dataset, ranging from a prediction of only the intercept (ARMA(0,0) model), to a model with multiple layers and units and hundreds or even thousands of parameters.

Implementation and extensions. The ARMA cell is implemented in Python using the TensorFlow (Abadi et al., 2016) framework, and naturally handles uni-, multi-, and tensor-variate time series data. It can be combined with other neural network components, such as convolutional layers, dropout, batch normalization, and others, and can be trained using standard optimization

techniques like (variants of) stochastic gradient descent. The implementation is available as open-source software, with a syntax similar to that of TensorFlow's native RNN cells, allowing for a seamless integration into existing models.

In summary, this essay presents the ARMA cell, bridging the gap between traditional statistical methods and modern deep learning techniques. It not only offers a model that is both effective in its simplicity and versatile in application, but also exhibits robust performance across a variety of numerical experiments.

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 265–283.

A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi. 2019. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2):107–115.

B. Amos. 2019. *Differentiable Optimization-Based Modeling for Machine Learning*. Ph.D. thesis, Carnegie Mellon University.

B. Barber and T. Odean. 2013. *The Behavior of Individual Investors*, pages 1533–1570. Elsevier.

A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. 2009. *Robust optimization*, volume 28. Princeton University Press.

H. Beyer and B. Sendhoff. 2007. Robust optimization—a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218.

G. Box, G. Jenkins, G. Reinsel, and G. Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth. 2017. Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76.

S. Boyd and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Cplex, IBM ILOG. 2009. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157.

G. Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

G. Dantzig. 1951. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 13:339–347.

C. Davidson. 2012. A dark knight for algos. *Risk*, 25(9):32.

S. Diamond and S. Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5.

References

A. Domahidi, E. Chu, and S. Boyd. 2013. ECOS: An SOCP solver for embedded systems. In *2013 European control conference*, pages 3071–3076. IEEE.

L. El Ghaoui and H. Lebret. 1997. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064.

F. Fabozzi, P. Kolm, D. Pachamanova, and S. Focardi. 2007. *Robust portfolio optimization and management*. John Wiley & Sons.

G. Feng, S. Giglio, and D. Xiu. 2020. Taming the factor zoo: A test of new factors. *Journal of Finance*, 75(3):1327–1370.

D. Filipović, M. Pelger, and Y. Ye. 2022. Stripping the discount curve-a robust machine learning approach. *Swiss Finance Institute Research Paper No. 22-24*.

A. Fu, B. Narasimhan, and S. Boyd. 2017. CVXR: An R package for disciplined convex optimization. *arXiv preprint arXiv:1711.07582*.

P. Goulart and Y. Chen. 2024. Clarabel documentation. Accessed on 2023-09-07.

B. Graham and D. Dodd. 1934. *Security analysis: Principles and technique*. McGraw-Hill.

M. Grant and S. Boyd. 2014. CVX: MATLAB software for disciplined convex programming, version 2.1.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

W. Hsieh and B. Tang. 1998. Applying neural network models to prediction and data analysis in meteorology and oceanography. *Bulletin of the American Meteorological Society*, 79(9):1855–1870.

P. Huber and E. Ronchetti. 2009. *Robust Statistics*. Wiley.

E. Jen. 2003. Stable or robust? what's the difference? *Complexity*, 8(3):12–18.

K. Johansson, G. Ogut, M. Pelger, T. Schmelzer, and S. Boyd. 2023. A simple method for predicting covariance matrices of financial returns. *Foundations and Trends in Econometrics*, 12(4):324–407.

A. Juditsky and A. Nemirovski. 2022. On well-structured convex–concave saddle point problems and variational inequalities with monotone operators. *Optimization Methods and Software*, 37(5):1567–1602.

A. Kochak and S. Sharma. 2015. Demand forecasting using neural network for supply chain management. *International journal of mechanical engineering and robotics research*, 4(1):96–104.

K. Kolchin, J. Podziemska, and D. Hadley. 2023. Sifma research – capital markets fact book, 2023. `https://www.sifma.org/resources/research/fact-book/`. Accessed: [2023-12-27].

References

Y. Li and W. Ma. 2010. Applications of artificial neural networks in financial economics: a survey. In *2010 International symposium on computational intelligence and design*, volume 1, pages 211–214. IEEE.

R. Lowenstein. 2001. *When genius failed: The rise and fall of Long-Term Capital Management*. Random House trade paperbacks.

H. Markowitz. 1952. Portfolio selection. *Journal of Finance*, 7(1):77–91.

R. Michaud and R. Michaud. 2008. *Efficient asset management: a practical guide to stock portfolio optimization and asset allocation*. Oxford University Press.

S. Mittnik and S. Rachev. 1993. Modeling asset returns with alternative stable distributions. *Econometric Reviews*, 12(3):261–330.

O. Morgenstern and J. Von Neumann. 1953. *Theory of games and economic behavior*. Princeton University Press.

MOSEK ApS. 2020. MOSEK modeling cookbook.

C. Nelson and A. Siegel. 1987. Parsimonious modeling of yield curves. *Journal of Business*, 60(4):473 – 489.

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. 2016. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169:1042–1068.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., New York.

F. Potra and S. Wright. 2000. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281–302. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

A. Shapiro, D. Dentcheva, and A. Ruszczynski. 2021. *Lectures on stochastic programming: Modeling and theory*. SIAM.

B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. 2020. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672.

R. Tütüncü and M. Koenig. 2004. Robust asset allocation. *Annals of Operations Research*, 132(1–4):157–187.

A. Tversky and D. Kahneman. 1992. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, 5(4):297–323.

M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. 2014. Convex optimization in Julia. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28. IEEE.

References

J. von Neumann, O. Morgenstern, and A. Rubinstein. 1944. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press, Princeton, NJ.

P. Wolfe. 1959. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398.

I. Yousaf and J. Goodell. 2023. Responses of us equity market sectors to the silicon valley bank implosion. *Finance Research Letters*, 55:103934.

Part II.

Markowitz Portfolio Construction at Seventy

This chapter is a reprint of:

Boyd, S., Johansson, K., Kahn, R., Schiele, P., Schmelzer, T. (2024). Markowitz Portfolio Construction at Seventy. *Available at https://arxiv.org/abs/2401.05080*. To appear in the *Journal of Portfolio Management*.

Author contributions:

The project was initiated by Stephen Boyd, Kasper Johansson, and Thomas Schmelzer, with Philipp Schiele joining them shortly thereafter. The majority of the essay was collaboratively written by Philipp Schiele, Kasper Johansson, Thomas Schmelzer, and Stephen Boyd. Philipp Schiele made significant contributions across all sections, particularly in the numerical experiments, and was responsible for the initial reference implementation of the model. This model was further developed by Philipp Schiele, Kasper Johansson, and Thomas Schmelzer. Throughout the project, both Ronald Kahn and Stephen Boyd provided continuous feedback and insights.

Supplementary material available at: https://github.com/cvxgrp/markowitz-reference

Markowitz Portfolio Construction at Seventy

Stephen Boyd* Kasper Johansson Ronald Kahn Philipp Schiele
Thomas Schmelzer

January 11, 2024

Abstract

More than seventy years ago Harry Markowitz formulated portfolio construction as an optimization problem that trades off expected return and risk, defined as the standard deviation of the portfolio returns. Since then the method has been extended to include many practical constraints and objective terms, such as transaction cost or leverage limits. Despite several criticisms of Markowitz's method, for example its sensitivity to poor forecasts of the return statistics, it has become the dominant quantitative method for portfolio construction in practice. In this article we describe an extension of Markowitz's method that addresses many practical effects and gracefully handles the uncertainty inherent in return statistics forecasting. Like Markowitz's original formulation, the extension is also a convex optimization problem, which can be solved with high reliability and speed.

*Alphabetical order.

Contents

1 Introduction

Harry Markowitz's 1952 paper *Portfolio Selection* [Mar52] was a true breakthrough in our understanding of and approach to investing. Before Markowitz there was (almost) no mathematical approach to investing. As a 25-year-old graduate student, Markowitz founded modern portfolio theory, and methods inspired by him would become the most widely used portfolio construction practices over the next 70 years (and counting).

Before Markowitz, diversification and risk were fuzzy concepts. Investors loosely connected risk to the probability of loss, but with no analytical rigor around that connection. Ben Graham, who along with David Dodd wrote *Security Analysis* [GD09], once commented that investors should own "a minimum of ten different issues and a maximum of about thirty" [Gra73].

There were a few precursors, such as an article by de Finetti, that contained some similar ideas before Markowitz; see [dF40, Rub06] for a discussion and more of the history of mathematical formulation of portfolio construction. Another notable precursor is John Burr Williams' 1938 *Theory of Investment Value* [Wil38]. He argued that the value of a company was the present value of future dividends. His book is full of mathematics, and Williams predicted that "mathematical analysis is a new tool of great power, whose use promises to lead to notable advances in investment analysis". That prediction came true with Markowitz's work. Indeed, Markowitz considered Williams' book as part of his inspiration. According to Markowitz, "the basic concepts of portfolio theory came to me one afternoon in the library while reading John Burr Williams' *Theory of Investment Value*".

For many years, the lack of data and accessible computational power [Mar19] rendered Markowitz's ideas impractical, despite his pragmatic approach. In 1963, William Sharpe published his market model [Sha63], designed to speed up the Markowitz calculations. This model was a one-factor risk model (the factor was the market return), with the assumption that all residual returns are uncorrelated. His paper stated that solving a 100-asset problem on an IBM 7090 computer required 33 minutes, but his simplified risk model reduced it to 30 seconds. He also commented that computers could only handle 249 assets at most with a full covariance matrix, but 2000 assets with the simplified risk model. Today such a problem can be solved in microseconds; we can routinely solve problems with tens of thousands of assets and substantially more factors in well under one second.

Markowitz portfolio construction has thrived for many years in spite of claims of various alleged deficiencies. These have included the method's sensitivity to data errors and estimation uncertainty, its single-period nature to handle what is fundamentally a multi-period problem, its symmetric definition of risk, and its neglect of higher moments like skewness and kurtosis. We will address these alleged criticisms and show that standard techniques in modern approaches to optimization effectively deal with them without altering Markowitz's vision for portfolio selection.

In 1990 Markowitz was awarded the Nobel Memorial Prize in Economics for his work on portfolio theory, shared with Merton Miller and William Sharpe. For more light on the fascinating historic details we recommend an interview with Markowitz [Mar19], his acceptance speech for the Nobel Prize [Mar23], and his remarks in the introduction to the

Handbook of Portfolio Construction [Gue10].

1.1 The original Markowitz idea

Markowitz identified two steps in the portfolio selection process. In a first step, the investor forms beliefs about the expected returns of the assets, expressed as a vector μ, and their covariances, expressed as a covariance matrix Σ, which gives the volatilities of asset returns and the correlations among them. These beliefs are the core inputs for the second step, which is the optimization of the portfolio based on these quantities.

He introduced the expected returns–variance of returns (E–V) rule, which states that an investor desires to achieve the maximum expected return for a portfolio while keeping its variance or risk below a given threshold. Convex programming was not a well developed field at that time, and Markowitz used a geometric interpretation in the space of portfolio weights [Mar52] to solve the problem we would now express as

$$\begin{array}{ll} \text{maximize} & \mu^T w \\ \text{subject to} & w^T \Sigma w \leq (\sigma^{\text{tar}})^2, \\ & \mathbf{1}^T w = 1, \end{array} \tag{1}$$

with variable $w \in \mathbf{R}^n$, the set of portfolio weights, where $\mathbf{1}$ is the vector with all entries one. The data in the problem are $\mu \in \mathbf{R}^n$, the vector of expected asset returns, and Σ, the $n \times n$ covariance matrix of asset returns. The positive parameter σ^{tar} is the target portfolio return standard deviation or volatility. (We define the weights and describe the problem more carefully in §2.)

There are many other ways to formulate the trade-off of expected return and risk as an optimization problem [BV04, ApS23]. One very popular method maximizes the *risk-adjusted return*, which is the expected portfolio return minus its variance, scaled by a positive *risk-aversion parameter*. This leads to the optimization problem [GK00]

$$\begin{array}{ll} \text{maximize} & \mu^T w - \gamma w^T \Sigma w \\ \text{subject to} & \mathbf{1}^T w = 1, \end{array} \tag{2}$$

where γ is the risk-aversion parameter that controls the trade-off between risk and return. Both problems (1) and (2) give the full trade-off curve of Pareto optimal weights, as σ^{tar} or γ vary from 0 to ∞ (although (1) can be infeasible when σ^{tar} is too small). One advantage of the first formulation (1) is that the parameter σ^{tar} that controls the volatility is interpretable as, simply, the target risk level. The risk-aversion parameter γ appearing in (2) is less interpretable. We will have more to say about the parameters that control trade-offs in portfolio construction in §4.

Both problems (1) and (2) have analytical solutions. For example the solution of (2) is given by

$$w^\star = \frac{1}{2\gamma}\Sigma^{-1}(\mu + \nu^\star \mathbf{1}), \qquad \nu^\star = \frac{2\gamma - \mathbf{1}^T\Sigma^{-1}\mu}{\mathbf{1}^T\Sigma^{-1}\mathbf{1}}.$$

(The scalar $\nu^\star$ is the optimal dual variable [BV04, Chap. 5].) We note here the appearance of the inverse covariance matrix. To compute $w^\star$ we would not compute the inverse, but rather solve two sets of equations to find $\Sigma^{-1}\mu$ and $\Sigma^{-1}\mathbf{1}$ [BV18]. Still, the appearance of Σ^{-1} in the expressions for the solutions give us a hint that the method can be sensitive to the input data when the covariance matrix Σ is nearly singular. These analytical formulas can also be used to back out so-called implied returns, *i.e.*, the mean μ for which a given portfolio is optimal. For example the *market implied return* μ^{mkt} is the return for which the optimal weights are the market weights, *i.e.*, proportional to asset capitalization.

Both formulations (1) and (2) are referred to as the basic Markowitz problem, or *mean-variance optimization*, since they both trade off the mean and variance of the portfolio return. In his original paper Markowitz also noted that additional constraints can be added to the problem, specifically the constraint that $w \geq 0$ (elementwise), which means the portfolio is long-only, *i.e.*, it does not contain any short positions. With this added constraint, the two problems above do not have simple analytical solutions. But the formulation (2), with the additional constraint $w \geq 0$, is a quadratic program (QP), a type of convex optimization problem for which numerical solvers were developed already in the late 1950s [Wol59]. In that early paper on QP, solving the Markowitz problem (2) with the long-only constraint $w \geq 0$ was listed as a prime application. Today we can solve either formulation reliably, with essentially any set of convex portfolio constraints.

Since the 1950s we have seen a truly stunning increase in computer power, as well as the development of convex optimization methods that are fast and reliable, and high-level languages that allow users to express complex convex optimization problems in a few lines of clear code. These advances allow us to extend Markowitz's formulation to include a large number of practical constraints and additional terms, such as transaction cost or the cost incurred when holding short positions. In addition to directly handling a number of practical issues, these generalizations of the basic Markowitz method also address the issue of sensitivity to the input data μ and Σ. This paper describes one such generalization of the basic Markowitz problem, that works well in practice.

Out of respect for Markowitz, and because the more generalized formulation we present here is nothing more than an extension of his original idea, we will refer to these more complex portfolio construction methods also as Markowitz methods. When we need to distinguish the extension of Markowitz's portfolio construction that we recommend from the basic Markowitz method, we refer to it as Markowitz++. (In computer science, the post-script ++ denotes the successor.)

1.2 Alleged deficiencies

The frequent criticism of Markowitz's work is a testament to its importance. These criticisms usually fall into one or more of the following (related) categories.

It's sensitive to data errors and estimation uncertainty. The sensitivity of Markowitz portfolio construction to input data is well documented [Mul93, MM08, SH13, Bra10, CY16],

and already hinted at by the inverse covariance that appears in the analytical solutions of the basic Markowitz method. This sensitivity, coupled with the challenge of estimating the mean and covariance of the return, leads to portfolios that exacerbate errors or deficiencies in the input data to find unrealistic and poorly performing portfolios. Some authors argue that choosing a portfolio by optimization, as Markowitz's method does, is essentially an estimation-error maximization method. This is still a research topic that draws much attention. In the recent papers [GPS22, Shk23] the authors quantify how the (basic) Markowitz portfolio is affected by estimation errors in the covariance matrix.

This criticism is justified, on the surface. Markowitz portfolio construction can perform poorly when it is naïvely implemented, for example by using empirical estimates of mean and covariance on a trailing window of past returns. But the critical practical issues of taming sensitivity and gracefully handling estimation errors are readily addressed using techniques such as regularization and robust optimization, described in more detail in §1.3.

It implicitly assumes risk symmetry. Markowitz portfolio construction uses variance of the portfolio return as its risk measure. With this risk measure a portfolio return well above the mean is just as bad as one that is well below the mean, whereas the former is clearly a good event, not a bad one. This observation should at least make one suspicious of the formulation, and has motivated a host of proposed alternatives, such as defining the risk taking into account only the downside [Mar59, Chap. IX]. This criticism is also valid, on the surface. But when the parameters are chosen appropriately, and the data are reasonable, portfolios constructed from mean-variance optimization do not suffer from this alleged deficiency.

We should maximize expected utility. A more academic version of the previous criticism is that portfolios should be constructed by maximizing the expected value of a concave increasing utility function of the portfolio return [VNM47]. The utility in mean-variance optimization (with risk-adjusted return objective) is $U(R) = R - \gamma R^2$, where R is the portfolio return. This utility function is concave, but only increasing for $R < 1/(2\gamma)$; above that value of return, it decreases, putting us in the awkward position of seeming to prefer smaller returns over larger ones.

This criticism is also valid, taken at face value; the quadratic utility above is indeed not increasing. Markowitz himself addressed the issue in a 1979 paper with H. Levy that argued that while mean-variance optimization does not appear to be the same as maximizing an expected utility, it is a very good approximation; see [LM79] and [MB14, Chap. 2]. But in fact it turns out that Markowitz portfolio construction *does* maximize the expected value of a concave increasing utility function. Specifically if we model the returns as Gaussian, and use the exponential utility $U(R) = 1 - \exp(-\gamma R)$, then the expected utility is the risk-adjusted return, up to an additive constant [LB23]. In other words, Markowitz portfolio construction *does* maximize expected utility of portfolio return, for a specific concave increasing utility function and a specific asset return distribution.

It considers only the first and second moments of the return. Mean-variance optimization naturally only considers the first two moments of the distribution. It would seem that taking higher moments like skewness and kurtosis into account might better describe investor preferences [Caj22, ZP21]. This, coupled with the fact that the tails of asset returns are not well modeled by a Gaussian distribution [Fam65], suggests that portfolio construction should consider higher moments than the first and second.

While it is possible to construct small academic examples where mean-variance optimization does poorly due to its neglect of higher moments, simple mean-variance optimization does very well on practical problems. In [LB23] the authors extend Markowitz by maximizing exponential utility, but with a more complex Gaussian mixture model of asset returns. Such a distribution is general, in that it can approximate any distribution. Their method evidently handles higher moments, but empirically gives no boost in performance on practical problems.

Markowitz himself addressed the common misconception that he labeled the "Great Confusion" [Mar19, MB14, Mar99, Mar09], stating that Gaussian returns are merely a sufficient but not a necessary condition on the return distribution for mean-variance optimization to work well and that mean and variance are good approximations for expected utility.

It's a greedy method. Portfolios are generally not just set up and then held for one investment period; they are rebalanced, and sometimes often. Problems in which a sequence of decisions are made, based on newly available information, are more accurately modeled not as simple optimization problems, but instead as stochastic control problems, also known as sequential decision making under uncertainty [Koc15, KWW22, Bel66, Ber12]. In the context of stochastic control, methods that take into account only the current decision and not future ones are called *greedy*, and in some cases can perform very poorly. This criticism is also, on its face, valid. Using Markowitz portfolio construction repeatedly, as is always done in practice, is a greedy method.

We can readily counter this criticism. First, in the special case with risk-adjusted return and quadratic transaction costs, and no additional constraints, the stochastic optimal policy can be worked out, and coincides with a single-period Markowitz portfolio [GK20, BB21]. This suggests that when other constraints are present, and the transaction cost is not quadratic, the (greedy) Markowitz method should not be too far from stochastic optimal.

Second, there are extensions of Markowitz portfolio construction, called *multi-period* methods, that plan a sequence of trades over a horizon, and then execute only the first trade; see, *e.g.*, [BBD+17, LUM22]. These multi-period methods can work better than so-called single-period methods, for example when a portfolio is transitioning between two managers, or being set up or liquidated over multiple periods. But in almost all other cases, single-period methods work just as well as multi-period ones.

The third response to this criticism more directly addresses the question. In the paper *Performance Bounds and Suboptimal Policies for Multi-Period Investment* [BMOW13], the authors develop bounds on how well a full stochastic control trading policy can do, and show empirically that single-period Markowitz trading essentially does as well as a full stochastic

control policy (which is impractical if there are more than a handful of assets). So while there are applications where greedy policies do much more poorly than a true stochastic control policy, it seems that multi-period trading is not one of them.

1.3 Robust optimization and regularization

Here we directly address the question of sensitivity of Markowitz portfolio construction to the input data μ and Σ. As mentioned above, the basic methods are indeed sensitive to these parameters. But this sensitivity can be mitigated and tamed using techniques that are widely used in other applications and fields, robust optimization and regularization.

Robust optimization. Modifying an optimization-based method to make it more robust to data uncertainty is done in many fields, using techniques that have differing names. When optimization is used in almost any application, some of the data are not known exactly, and solving the optimization problem without recognizing this uncertainty, for example by using some kind of mean or typical values of the parameters, can lead to very poor practical performance. *Robust optimization* is a subfield of optimization that develops methods to handle or mitigate the adverse effects of parameter uncertainty; see, *e.g.*, [BTEGN09, TK04, GMT14, BTN02, BBC11, Lob00]. These methods tend to fall in one of two approaches: statistical or worst-case deterministic. In a statistical model, the uncertain parameters are modeled as random variables and the goal is to optimize the expected value of the objective under this distribution, leading to a *stochastic optimization problem* [SDR21], [BV04, Chap. 6.4.1]. A worst-case deterministic uncertainty model posits a set of possible values for parameters, and the goal is to optimize the worst-case value of the objective over the possible parameter values [BS07], [BV04, Chap. 6.4.2]. Another name for worst-case robust optimization is *adversarial optimization*, since we can model the problem as us choosing values for the variables to obtain the best objective, after which an adversary chooses the values of the parameters so as to achieve the worst possible objective. Worst-case robust optimization has many variations and goes by many names. For example when the set of possible parameter values is finite, they are called *scenarios* or *regimes*, and optimizing for the worst-case scenario is called *worst-case scenario optimization*. While these general approaches sound quite different, they often lead to very similar solutions, and both can work well in applications. Robust optimization methods work by modifying the objective or constraints to model the possible variation in the data.

One very successful application of robust optimization is in *robust control*, where a control system is designed so that the control performance is not too sensitive to changes in the system dynamics [ZD98, KDG96]. So-called linear quadratic optimal control was developed around 1960, and used in many applications. Its occasional sensitivity to the data (in this case, the dynamic model of the system being controlled) was noted then; by the early 1990s robust control methods were developed, and are now very widely used.

Regularization. Regularization is another term for methods that modify an optimization problem to mitigate sensitivity to data. It is almost universally used in statistics and machine learning when fitting models to data. Here we fit the parameters of a model to some given training data, accounting for the fact that the training data set could have been different [TA77, HTF09]. This process of regularization can be done explicitly by adding a penalty term to the objective, and also implicitly by adding constraints to the problem that prevent extreme outcomes. Regularization can often be interpreted as a form of robust optimization; see, *e.g.*, [BV04, Chap. 6.3–6.4].

The high level story. Robust optimization and regularization both follow the same high level story, and both can be applied to the Markowitz problem. The story starts with a basic optimization-based method that relies on data that are not known precisely. We then modify the optimization problem, often by adding additional objective terms or constraints. Doing this *worsens* the in-sample performance. But if done well, it *improves* out-of-sample performance. Roughly speaking, robustification and regularization tell the optimizer to not fully trust the data, and this serves it well out-of-sample.

In portfolio construction a long-only constraint can be interpreted as a form of regularization [JM03]. A less extreme version is to impose a leverage limit, which can help avoid many of the data sensitivity issues. We will describe below some effective and simple robustification methods for portfolio construction.

Regularization can (and should) also be applied to the forecasting of the mean and covariance in Markowitz portfolio construction. The Black-Litterman approach to estimating the mean returns regularizes the estimate toward the market implied return [BL90]. A return covariance estimate can be regularized using *shrinkage*, another term for regularized estimation in statistics [LW04].

1.4 Convex optimization

Over the same 70-year period since Markowitz's original work, there has been a parallel advance in mathematical optimization, and especially convex optimization, not to mention stunning increases in available computer power. Roughly speaking, convex optimization problems are mathematical optimization problems that satisfy certain mathematical properties. They can be solved reliably and efficiently, even when they involve a very large number of variables and constraints, and involve nonlinear, even nondifferentiable, functions [BV04].

Shortly before Markowitz published his paper on portfolio selection, George Dantzig developed the simplex method [Dan51], which allowed for the efficient solution of linear programs. In 1959, Wolfe [Wol59] extended the simplex method to QP problems, citing Markowitz's work as a motivating application. This close connection between portfolio construction and optimization was no coincidence, since Dantzig and Markowitz were colleagues at RAND.

Since then, the field of convex optimization has grown tremendously. Today, convex optimization is a mature field with a large body of theory, algorithms, software, and applications [BV04]. Being able to solve optimization problems reliably and efficiently is crucial

for portfolio construction, especially for back-testing or simulating a proposed method on historical or synthesized data, where portfolio construction has to be carried many times. Thus, any extension of the Markowitz objective or additional constraints should be convex to ensure tractability. As we will see, this is hardly a limitation in practice.

Solvers. The dominant convex optimization problem form is now the *cone program*, a generalization of linear programming that handles nonlinear objective terms and constraints [NN92, BV04, LVBL98a, VB96]. There are now a number of reliable and efficient solvers for such problems, including open-source ones like ECOS [DCB13], Clarabel [GC24], and SCS [OCPB16], and commercial solvers such as MOSEK [ApS20], GUROBI [Gur23], and CPLEX [Cpl09]. A recent open-source solver for QPs is OSQP [SBG+20].

Domain-specific languages. Convex optimization is also now very accessible to practitioners, even those without a strong background in the mathematics or algorithms of convex optimization, thanks to high-level domain-specific languages (DSLs) for convex optimization, such as CVXPY [AVDB18, DB16], CVX [GB14], Convex.jl [UMZ+14], CVXR [FNB17], and YALMIP [Lof04]. These DSLs make it easy to specify complex, but convex, optimization problems in a natural, human readable way. The DSLs transform the problem from the human readable form to a lower level form (often a cone program) suitable for a solver. These DSLs make it easy to develop convex optimization based methods, as well as to modify, update, and maintain existing ones. As a result, CVXPY is used at many quantitative hedge funds today, as well as in many other applications and industries. The proposed extension of Markowitz's portfolio construction method that we describe below is a good example of the use of CVXPY. It is a complex problem involving nonlinear and nondifferentiable functions, but its specification in CVXPY takes only a few tens of lines of clear readable code, given in appendix B. The overhead of translating the human readable problem specification into a cone program is typically small. Additionally, in some DSLs, such as CVXPY, problems can be parametrized [AAB+19], such that they can be solved for a range of values of the parameters, making the translation overhead negligible. Related to DSLs are modeling layers provided by some solver, such as MOSEK's Fusion API [ApS20], which provides a high-level interface to the solver. Less focused on convex optimization, there are other modeling languages such as JuMP [LDD+23] and Pyomo [HWW11, BHH+21] that do not verify convexity, but provide flexibility in modeling a wide range of optimization problems, including nonconvex ones.

Code generators. Code generators like CVXGEN [MB12] and CVXPYgen [SBD+22] are similar to DSLs. They support high level specification of a problem (family) but instead of directly solving the problem, they generate custom low level code (typically C) for the problem that is specified. This code can be compiled to a very fast and totally reliable solver, suitable for embedded real-time applications. For example, CVXGEN-generated code guides all of SpaceX's Falcon 9 and Falcon Heavy first stages to their landings [Bla16].

1.5 Previous work

The literature on portfolio construction is vast, and focusing on the practical implementation of Markowitz's ideas, we do not attempt to survey it here in detail. Instead, we highlight only a few major developments that are relevant to our work. For a detailed overview see, *e.g.*, [GK00, Chap. 14], [Nar09, Chap. 6], and [CT06, KTF14].

Building on Markowitz's framework, the field of portfolio construction has undergone substantial evolution. Notable contributions include Sharpe's Capital Asset Pricing Model [Sha63] and the Black-Litterman model [BL90]. A pivotal figure in bringing the field to the forefront of the industry was Barr Rosenberg, whose research evolved to become the Barra risk model [Ros84, She96], first used for risk modeling and then in portfolio optimization. The introduction of risk parity models [MRT10] brought a focus on risk distribution. Additionally, hierarchical risk parity, a recent advancement, offers a more intricate approach to risk allocation, considering the hierarchical structure of asset correlations [DP16]. These developments reflect the field's dynamic adaptation to evolving market conditions and analytical techniques.

Software. Dedicated software helped practitioners access the solvers and DSLs mentioned earlier, and has facilitated the wide acceptance of Markowitz portfolio construction. A wealth of software packages have been developed for portfolio optimization, many (if not most) with Python interfaces, both open-source and commercial. Examples range from simple web-based visualization tools to complex trading platforms. Here we mention only a few of these software implementations.

On the simpler end Portfolio Visualizer [Glo23] is a web-based tool that allows users to back-test and visualize various portfolio strategies. PyPortfolioOpt [Mar21] and Cvxportfolio [BBD+17] are Python packages offering various portfolio optimization techniques. PyPortfolioOpt includes mean-variance optimization, Black-Litterman allocation [BL90], and more recent alternatives like the Hierarchical Risk Parity algorithm [DP16], while Cvxportfolio [BBD+17] supports multi-period strategies. The skfolio [Del23] package offers similar functionality to PyPortfolioOpt, with a focus on interoperability with the scikit-learn [PVG+11] machine learning library. Another Python implementation is proposed in [SXD20], where the authors introduce an approach to multicriteria portfolio optimization. Quantlib [The23] is an alternate open-source software package for modeling, trading, and risk management.

The list of commercial software is also extensive. MATLAB's Financial Toolbox [Bra13, Mat23] includes functions for mathematical modeling and statistical analysis of financial data, including portfolio optimization. Another example is Axioma, which on top of its popular risk model offers a portfolio optimizer [Qon23].

Other software packages include Portfolio123 [Por23a], PortfoliosLab [Por23b], and PortfolioLab by Hudson & Thames [Tha23]. Additionally, many solvers, such as MOSEK [ApS20, ApS23], provide extensive examples of portfolio optimization problems, making them easy to use for portfolio optimization.

1.6 This paper

Our goal is to describe an extension of the basic Markowitz portfolio construction method that includes a number of additional objective terms and constraints that reflect practical issues and address the issue of sensitivity to inevitable forecasting errors. We give a minimal formulation that is both simple and practical; we make no attempt to list all possible extensions that a portfolio manager (PM) might wish to add.

While the resulting optimization problem might appear complex, containing nonlinear nondifferentiable functions, it is convex, which means it can be solved reliably and efficiently. It can also be specified in a DSL such as CVXPY in just a few tens of lines of clear simple code. We can solve even large instances of the optimization problem very quickly, making it practical to carry out extensive back-testing to predict performance or adjust parameter values. One additional advantage of our formulation is that parameters that need to be specified are generally more interpretable than those appearing in basic formulations. For example a PM specifies a target risk and a target turnover instead of some parameters that are less directly related to them.

Most of the material in this paper is not new but scattered across many sources, in different formats, and indeed in different application fields. Some of our recommendations are widely accepted and industry standard, but others are rarely discussed in the literature and even less commonly used in practice.

The authors bring a diverse set of backgrounds to this paper. Some of us have applied Markowitz portfolio construction day-to-day in research, writing, and real portfolios. Others approach Markowitz's method from the perspective of optimization and control in engineering. Control systems engineering has a long history and is widely applied in essentially all engineering applications. Most applications of control engineering use methods based on models that are either wrong or heavily simplified. While naïve implementations of these methods do not work well (or worse), simple sensible modifications, similar to the ones we describe later in this paper, work very well in practice.

These different backgrounds together can provide a new perspective and bring modern tools to the endeavor Markowitz began. These techniques have made Markowitz's method even more applicable and useful to investors.

Software. We have created two companion software packages. One is designed for pedagogical purposes, uses limited parameter testing and checking, and very closely follows the terminology and notation of the paper. It is available at

https://github.com/cvxgrp/markowitz-reference.

The second package is a robust and flexible implementation, which is better suited for practical use. It is available at

https://github.com/cvxgrp/cvxmarkowitz.

Outline. In §2 we set up our notation, define weights and trades, and describe various objective terms and constraints. Return and risk forecasts are covered in §3. In §4 we pull together the material of the previous two sections to define the (generalized) Markowitz trading problem, which we refer to as Markowitz++. In §5 we present some simple numerical experiments that illustrate how the extra terms robustify the basic Markowitz trading policy, and how parameters are tuned via back-testing to improve good performance.

2 Portfolio holdings and trades

This section introduces the notation and terminology for portfolio holdings, weights, and trades, fundamental objects in portfolio construction independent of the trading strategy. We follow the notation of [BBD+17], with the exceptions of handling the cash weight separately and dropping the time period subscript.

2.1 Portfolio weights

Universe. We consider a portfolio consisting of investments (possibly short) in n assets, plus a cash account. We refer to the set of assets we might hold as the *universe* of assets, and n as the size of the universe. These assets are assumed to be reasonably liquid, and could include, for example, stocks, bonds, or currencies.

Asset and cash weights. To describe the portfolio investments, we work with the *weights* or fractions of the total portfolio value for each asset, with negative values indicating short positions. We denote the weights for the assets as w_i, $i = 1, \ldots, n$, and collect them into a portfolio weight vector $w = (w_1, \ldots, w_n) \in \mathbf{R}^n$. The weights are readily interpreted: $w_i = 0.05$ means that 5% of the total portfolio value is held in asset i, and $w_k = -0.01$ means that we hold a short position in asset k, with value 1% of the total portfolio value. The dollar value of asset i held is Vw_i, where V is the total portfolio value, assumed to be positive.

We denote the weight for the cash account, *i.e.*, our cash value divided by the portfolio value, as c. If c is negative, it represents a loan. When $c > 0$ we say the portfolio is *diluted* with cash; when $c < 0$, the portfolio is *margined*. The dollar value of the cash account is Vc.

By definition the weights sum to one, so we have

$$\mathbf{1}^T w + c = 1, \tag{3}$$

where $\mathbf{1}$ is the vector with all entries one. The first term, $\mathbf{1}^T w$, is the total weight on the non-cash assets, and we refer to it as the total asset weight. The cash weight is one minus the total asset weight, *i.e.*, $c = 1 - \mathbf{1}^T w$.

Several portfolio types can be expressed in terms of the holdings. A *long-only* portfolio is one with all asset weights nonnegative, *i.e.*, $w \geq 0$ (elementwise). A portfolio with $c = 0$, *i.e.*, no cash holdings, is called *fully invested*. In such a portfolio we have $\mathbf{1}^T w = 1$, *i.e.*, the

total asset weight is one. As another example, a *cash-neutral* portfolio is one with $c = 1$. For a cash-neutral portfolio we have $\mathbf{1}^T w = 0$, *i.e.*, the total (net) asset weight is zero.

Leverage. The *leverage* of the portfolio, denoted L, is

$$L = \sum_{i=1}^{n} |w_i| = \|w\|_1.$$

(Several other closely related definitions are also used. Our definition is commonly referred to as the *gross leverage* [AGv11].) The leverage does not include the cash account.

In a long-only portfolio, the leverage is equal to the total asset weight. The *130-30 portfolio* [LEB09] refers to a fully invested portfolio with leverage $L = 1.6$. For such a portfolio, the total weight of the short positions (*i.e.*, negative w_i) is -0.3 and the total weight of the long positions (*i.e.*, positive w_i) is 1.3.

Benchmark and active weights. In some cases our focus is on portfolio performance relative to a *benchmark portfolio*. We let $w^{\mathrm{b}} \in \mathbf{R}^n$ denote the weights of the benchmark. Typically the benchmark does not include any cash weight, so $\mathbf{1}^T w^{\mathrm{b}} = 1$. We refer to $w - w^{\mathrm{b}}$ as the *active weights* of our portfolio. A positive active weight on asset i, *i.e.*, $w_i - w_i^{\mathrm{b}} > 0$, means our portfolio is *over-weight* (relative to the benchmark) on asset i; a negative active weight, $w_i - w_i^{\mathrm{b}} < 0$, means our portfolio is *under-weight* on asset i.

2.2 Holding constraints and costs

Several constraints and costs are associated with the portfolio holdings w and c.

Weight limits. Asset and cash *weight limits* have the form

$$w_i^{\min} \leq w_i \leq w_i^{\max}, \quad i = 1, \ldots, n, \quad c^{\min} \leq c \leq c^{\max},$$

where $w^{\min}$ and $w^{\max}$ are given vectors of lower and upper limits on asset weights, and $c^{\min}$ and $c^{\max}$ are given lower and upper limits on the cash weight. We write the asset weight inequalities in vector form as $w^{\min} \leq w \leq w^{\max}$. We have already encountered a simple example: a long-only portfolio has $w^{\min} = 0$.

Portfolio weight limits can reflect hard requirements, for example that a portfolio must (by legal or regulatory requirements) be long-only. Portfolio weight limits can also be used to avoid excessive concentration of a portfolio, or limit short positions. For example, $w^{\max} = 0.15$ means that our portfolio cannot hold more than 15% of the total portfolio value in any one asset. (Here we adopt the convention that in a vector-scalar inequality, the scalar is implicitly multiplied by $\mathbf{1}$.) As another example, $w^{\min} = -0.05$ means that the short position in any asset can never exceed 5% of the total portfolio value. For large portfolios it is reasonable to also limit holdings relative to the asset capitalization, *e.g.*, to require that our portfolio holdings of each asset are no more than 10% of the asset capitalization.

Weight limits can also be used to capture the portfolio manager's views on how the market will evolve. For example, she might insist on a long position for some assets, and a short position for some others.

When a benchmark is used, we can impose limits on active weights. For example $|w - w^{\mathrm{b}}| \leq 0.10$ means that no asset in the portfolio can be more than 10% over-weight or under-weight.

Leverage limit. In addition to weight limits, we can impose a leverage limit,

$$L \leq L^{\mathrm{tar}}, \tag{4}$$

where L^{tar} is a specified maximum or target leverage value. (Other authors have suggested including leverage as a penalty term in the objective, to model leverage aversion and identify the optimal amount of leverage in the presence of leverage aversion [JL13].)

Holding costs. In general a fee is paid to borrow an asset in order to enter a short position. Analogously we pay a borrow cost fee for a negative cash weight. We will assume these holding costs are a linear function of the negative weights, *i.e.*, of the form

$$\phi^{\mathrm{hold}}(w, c) = (\kappa^{\mathrm{short}})^T(-w)_+ + \kappa^{\mathrm{borrow}}(-c)_+, \tag{5}$$

where $(a)_+ = \max\{a, 0\}$ denotes the nonnegative part, applied elementwise and in its first use above. Here $\kappa^{\mathrm{short}} \geq 0$ is the vector of borrow cost (rates) for the assets, and $\kappa^{\mathrm{borrow}} \geq 0$ is the borrow cost for cash.

Other holding constraints. There are many other constraints on weights that might be imposed, some convex, and others not. A *concentration limit* is an example of a useful constraint that is convex. It states that the sum of the K largest absolute weights cannot exceed some limit. As a specific example, we can require that no collection of five assets can have a total absolute weight of more than 30% [SR20, ApS23]. A *minimum nonzero holding* constraint is an example of a commonly imposed nonconvex constraint. It states that any nonzero weight must have an absolute value exceeding some given minimum, such as 0.5%. (This one is easily handled using a heuristic based on convex optimization; see §4.3.)

2.3 Trades

Trade vector. We let w^{pre} and c^{pre} denote the pre-trade portfolio weights, *i.e.*, the portfolio weights before we carry out the trades to construct the portfolio given by w and c. We need the pre-trade weights to account for transaction costs. We refer to

$$z = w - w^{\mathrm{pre}}, \tag{6}$$

the current weights minus the previous ones, as the (vector of) *trades* or the *trade list*. These trades have a simple interpretation: $z_i = 0.01$ means we buy an amount of asset i equal in

value to 1% of our total portfolio value, and $z_i = -0.03$ means we sell an amount of asset i equal to 3% of the portfolio value.

Since $\mathbf{1}^T w^{\text{pre}} + c^{\text{pre}} = 1$, we have

$$c = c^{\text{pre}} - \mathbf{1}^T z, \tag{7}$$

i.e., the post-trade cash weight is the pre-trade cash weight minus the net weight of the trades. This does not include holding and transaction costs, discussed below.

Turnover. The quantity

$$T = \frac{1}{2}\sum_{i=1}^{n} |z_i| = \frac{1}{2}\|z\|_1$$

is the *turnover*. Here too, several other different but closely related definitions are also used, for example the minimum of the total weight bought and the total weight sold [GK00, Chap. 16]. A turnover $T = 0.01$ means that the average of total amount bought and total amount sold is 1% of the total portfolio value. The turnover is often annualized, by multiplying by the number of trading periods per year.

2.4 Trading constraints and costs

We typically have constraints on the trade vector z, as well as a trading cost that depends on z.

Trade limits. Trade limits impose lower and upper bounds on trades, as

$$z^{\min} \leq z \leq z^{\max},$$

where $z^{\min}$ and $z^{\max}$ are given limits. These trade limits can be used to limit market participation, defined as the ratio of the magnitude of each trade to the trading volume, using, *e.g.*,

$$|z| \leq 0.05v, \tag{8}$$

where $v \in \mathbf{R}^n$ is the trading volumes of the assets, expressed as multiples of the portfolio value. This constraint limits our participation for each asset to be less than 5%. (It corresponds to trade limits $z^{\max} = -z^{\min} = 0.05v$.) Since the trading volumes are not known when z is chosen, we use a forecast instead of the realized trading volumes.

Turnover limit. In addition to trade limits, we can limit the turnover as

$$T \leq T^{\text{tar}}, \tag{9}$$

where T^{tar} is a specified turnover limit.

Trading cost. Trading cost refers to the cost of carrying out a trade. For example, if we buy a small quantity of an asset, we pay the ask price, while if we sell an asset, we receive the bid price. Since the nominal price of an asset is the midpoint between the ask and bid prices, we can think of buying or selling the asset as doing so at the nominal price, plus an additional positive cost that is the trade amount times one-half the bid-ask spread. This *bid-ask spread transaction cost* has the form

$$\sum_{i=1}^{n} \kappa_i^{\text{spread}} |z_i| = (\kappa^{\text{spread}})^T |z|,$$

where $\kappa^{\text{spread}} \in \mathbf{R}^n$ is the vector of one-half the asset bid-ask spreads (which are all positive). This is the transaction cost expressed as a fraction of the portfolio value. For small trades this is a reasonable approximation of transaction cost.

For larger trades we 'eat through' the order book. To buy a quantity of an asset, we buy each ask lot, in order from lowest price, until we fill our order. An analogous situation occurs when selling. This means that we end up paying more per share than the ask price when buying, or receiving less than the bid price when selling. This phenomenon is called *market impact*.

A useful approximation of transaction cost that takes market impact into account is

$$\phi^{\text{trade}}(z) = (\kappa^{\text{spread}})^T |z| + (\kappa^{\text{impact}})^T |z|^{3/2}, \tag{10}$$

where the first term is the bid-ask spread component of transaction cost, and the second models the market impact, *i.e.*, the additional cost incurred as the trade eats through the order book. The vector κ^{impact} has positive entries and typically takes the form

$$\kappa_i^{\text{impact}} = a s_i v_i^{-1/2},$$

where s_i is the volatility of asset i over the trading period, v_i is the volume of market trading, expressed as a multiple of the portfolio value, and a is a constant on the order of one; see [GK00, Loe83, TLD+11, MTB14] and [BBD+17, §2.3]. Evidently the transaction cost increases with volatility, and decreases with market volume. Several other approximations of transaction cost are used [AC00, RRJ12].

Liquidation cost. Suppose we liquidate the portfolio, *i.e.*, close out all asset positions, which corresponds to the trade vector $z = -w$. The *liquidation cost* is

$$\phi^{\text{trade}}(-w) = (\kappa^{\text{spread}})^T |w| + (\kappa^{\text{impact}})^T |w|^{3/2}.$$

If the liquidation is carried out over multiple periods, the bid-ask term stays the same, but the market impact term decreases. For this reason a common approximation of the liquidation cost ignores the market impact term. A liquidation cost constraint has the form

$$(\kappa^{\text{spread}})^T |w| \leq \ell^{\max}, \tag{11}$$

where $\ell^{\max}$ is a maximum allowable liquidation cost, such as 1%. This is a weight constraint; it limits our holdings in less liquid assets, which have higher bid-ask spreads. It can be interpreted as a liquidity-weighted leverage (taking the bid-ask spread as a proxy for liquidity). When all assets have the same bid-ask spread, the liquidation constraint reduces to a leverage constraint. For example with all bid-ask spreads equal to 0.001 (*i.e.*, 10 basis points or bps) and a maximum liquidation cost $\ell^{\max} = 0.01$ (*i.e.*, 1% of the total portfolio value), the liquidation cost limit (11) reduces to a leverage limit (4) with $L^{\text{tar}} = 10$.

Transaction cost forecasts. When the trades z are chosen, we do not know the bid-ask spreads, the volatilities, or the volumes. Instead we use forecasts of these quantities in (8), (10), and (11). Simple forecasts, such as a trailing average or median of realized values, are typically used. More sophisticated forecasts take can into account calendar effects such as seasonality, or the typically low trading volume the day after Thanksgiving.

3 Return and risk forecasts

3.1 Return

Gross portfolio return. We let r_i denote the return, adjusted for dividends, splits, and other corporate actions, of asset i over the investment period. We collect these asset returns into a return vector $r = (r_1, \ldots, r_n) \in \mathbf{R}^n$. The portfolio return from asset i is $r_i w_i$. We let r^{rf} denote the risk-free interest rate, so the return in the cash account is $r^{\text{rf}} c$. The (gross) total portfolio return is then

$$R = r^T w + r^{\text{rf}} c.$$

This gross return does not include holding or trading costs. A closely related quantity is the *excess return*, the portfolio return minus the risk-free return, $R - r^{\text{rf}} = r^T w + r^{\text{rf}}(c - 1)$.

Net portfolio return. The net portfolio return is the gross return minus the holding costs and transaction costs,

$$R^{\text{net}} = R - \phi^{\text{hold}}(w) - \phi^{\text{trade}}(z). \tag{12}$$

Active return. The *active portfolio return* is the return relative to a benchmark portfolio,

$$r^T w + r^{\text{rf}} c - r^T w^{\text{b}} = r^T (w - w^{\text{b}}) + r^{\text{rf}} c.$$

If we subtract holding and trading costs we obtain the *net active portfolio return*.

Cash as slack. Since we do not know but only forecast the bid-ask spread, volatility, and volume, which appear in the transaction cost (10) (which is itself only an approximation) we should consider the post-trade cash c in (7) as an approximation that uses a forecast of holding and transaction costs, not the realized holding and transaction costs. We do not expect the realized post-trade cash weight to be exactly c.

3.2 Probabilistic asset return model

When we choose the trades z we do not know the asset returns r. Instead, we model r as a multivariate random variable with mean $\mu \in \mathbf{R}^n$ and covariance matrix $\Sigma \in \mathbf{S}^n_{++}$ (the set of symmetric positive definite $n \times n$ matrices),

$$\mathbf{E}\, r = \mu, \qquad \mathbf{E}(r-\mu)(r-\mu)^T = \Sigma.$$

The entries of the mean μ are often referred to as *trading signals* [Isi21]. The asset return mean and covariance are forecasts, as described below. The asset return volatilities $s \in \mathbf{R}^n$ appearing in the transaction cost model (10) can be expressed as $s = \mathbf{diag}(\Sigma)^{1/2}$, where the squareroot is elementwise.

Expected return and risk. With this statistical model of r, the portfolio return R is a random variable with mean $\bar{R} = \mathbf{E}\, R$ and variance $\sigma^2 = \mathbf{var}\, R$ given by

$$\bar{R} = \mu^T w + r^{\mathrm{rf}} c, \qquad \sigma^2 = w^T \Sigma w.$$

The *risk* of the portfolio is defined as the standard deviation of the portfolio return, *i.e.*, σ.

Similarly, the active return R^{a} is a random variable with mean and variance

$$\bar{R}^{\mathrm{a}} = \mu^T(w - w^{\mathrm{b}}) + r^{\mathrm{rf}} c = \bar{R} - \mu^T w^{\mathrm{b}}, \qquad (\sigma^{\mathrm{a}})^2 = (w - w^{\mathrm{b}})^T \Sigma (w - w^{\mathrm{b}}),$$

and the *active risk* is σ^{a}. The risk and active risk are often given in annualized form, obtained by multiplying them by the squareroot of the number of periods per year.

The parameters μ and Σ are estimates or forecasts of the statistical model of asset returns, which is itself an approximation. For this reason the risk σ is called the *ex-ante* risk, to distinguish it from the standard deviation of the realized portfolio returns when trading, the *ex-post* risk. Similarly we refer to σ^{a} as the ex-ante active risk.

Optimizing expected return and risk. We have two objectives, high expected return and low risk. Perhaps the most common method for combining these objectives is to form a *risk-adjusted return*,

$$\bar{R} - \gamma\sigma^2,$$

where $\gamma > 0$ is the *risk aversion parameter*. Maximizing risk-adjusted return (possibly with other objective terms, and subject to constraints) gives the desired portfolio. Increasing γ gives us a portfolio with lower risk and also lower expected return. The risk aversion parameter allows us to explore the risk-return trade-off. This risk-adjusted return approach became popular in part because the resulting optimization problem is typically a quadratic program (QP), for which reliable solvers were developed even in the 1960s.

Another approach is to maximize expected return (possibly with other objective terms), subject to a *risk budget* or *risk target* constraint

$$\sigma \leq \sigma^{\mathrm{tar}}. \tag{13}$$

(The corresponding optimization problem is not a QP, but is readily handled by convex optimization solvers developed in the 1990s [LVBL98b, NN94, Stu99, TTT99].) This formulation seems more natural, since a portfolio manager will often have a target risk in her mind, *e.g.*, 8% annualized. This is the basic formulation that we recommend.

There are many other ways to combine expected return and risk. For example, we can maximize the return/risk ratio, called the *Sharpe ratio* (with no benchmark) or *information ratio* (with a benchmark). This problem too can be solved via convex optimization, at least when the constraints are simple [BV23].

3.3 Factor model

In practice, and especially for large universes, it is common to use a *factor model* for the returns. The factor return model, with k factors (typically with $k \ll n$), has the form

$$r = Ff + \epsilon, \tag{14}$$

where $F \in \mathbf{R}^{n\times k}$ is the *factor loading matrix*, $f \in \mathbf{R}^k$ is the vector of *factor returns*, and $\epsilon \in \mathbf{R}^n$ is the *idiosyncratic return*. The term Ff is interpreted as the component of asset returns explainable or predicted by the factor returns.

At portfolio construction time the factor loading matrix F is known, and the factor return f and idiosyncratic return ϵ are modeled as uncorrelated random variables with means and covariance matrices

$$\mathbf{E}\, f = \bar{f}, \qquad \mathbf{cov}\, f = \Sigma^{\mathrm{f}}, \qquad \mathbf{E}\, \epsilon = \bar{\epsilon}, \qquad \mathbf{cov}\, \epsilon = D,$$

where D is diagonal (with positive entries). The entries $\bar{\epsilon}$, the means of the idiosyncratic returns, are also referred to as the *alphas*, especially when there is only one factor which is the overall market return. They are the part of the asset returns not explained by the factor returns.

With the factor model (14) the asset return mean and covariance are

$$\mu = F\bar{f} + \bar{\epsilon}, \qquad \Sigma = F\Sigma^f F^T + D.$$

The return covariance matrix in a factor model has a special form, low rank plus diagonal. The portfolio return mean and variance are

$$\bar{R} = (F\bar{f})^T w + \bar{\epsilon}^T w + r^{\mathrm{rf}} c, \qquad \sigma^2 = (F^T w)^T \Sigma^{\mathrm{f}} (F^T w) + w^T D w.$$

The factor returns are constructed to have explanatory power for the returns of assets in our universe. For equities, they are typically the returns of other portfolios, such as the overall market (with weights proportional to capitalization), industries, and style portfolios like the celebrated Fama-French factors [FF92, FF93]. For bonds, the factors are typically constructed from yield curves, interest rates, and spreads. These traditional factors are interpretable.

Factors can also can be constructed directly from previous realized asset returns using methods such as principal component analysis (PCA) [BN08, Bai03, LP20a, LP20b, PX22b, PX22a]. Aside from the first principal component, which typically is close to the market return, these factors are less interpretable.

Factor and idiosyncratic returns. A factor model gives an alternative method to specify the expected return as $\mu = F\bar{f} + \bar{\epsilon}$, where $\bar{f}$ is a forecast of the factor returns and $\bar{\epsilon}$ is a forecast of the idiosyncratic returns, *i.e.*, the asset alphas. One common method uses only a forecast of the factor returns, with $\bar{\epsilon} = 0$, so $\mu = F\bar{f}$. A complementary method assumes zero factor returns, so we have $\mu = \bar{\epsilon}$, *i.e.*, the mean asset returns are the same as the idiosyncratic asset mean returns.

Factor betas and neutrality. Under the factor model (14), the covariance of the portfolio return R with the factor returns f is the k-vector

$$\mathbf{cov}(R, f) = \Sigma^{\mathrm{f}} F^T w.$$

The *betas* of the portfolio with respect to the factors divide these covariances by the variance of the factors,

$$\beta = \mathbf{diag}(s^{\mathrm{f}})^{-2}\Sigma^{\mathrm{f}} F^T w,$$

where $s^{\mathrm{f}} = \mathbf{diag}(\Sigma^{\mathrm{f}})^{-1/2}$ is the vector of factor return volatilities.

The constraint that our portfolio return is uncorrelated (or has zero beta) with the ith factor return f_i, under the factor model (14), is

$$\mathbf{cov}(R, f)_i = (\Sigma^f F^T w)_i = 0. \tag{15}$$

This is referred to as *factor neutrality* (with respect to the ith factor). It is a simple linear equality constraint, which can be expressed as $a^T w = 0$, where a is the ith column of $F\Sigma^{\mathrm{f}}$. Factor neutrality constraints are typically used with active weights. In this case, factor neutrality means that the portfolio beta matches the benchmark beta for that factor. This also is a linear equality constraint that can be expressed as $\beta_i = \beta_i^{\mathrm{b}}$, with β^{b} the benchmark betas.

Advantages of a factor model. Especially with large universes, the factor model (specified by F, Σ^{f}, and D) can give a better estimate of the return covariance, compared to methods that directly estimate the $n \times n$ matrix Σ [JOP+23]. Another substantial advantage is computational. By exploiting the low-rank plus diagonal structure of the return covariance with a factor model, we can reduce the computational complexity of solving the Markowitz optimization problem from $O(n^3)$ flops (without exploiting the factor model) to $O(nk^2)$ flops (exploiting the factor form). These computational savings can be dramatic, *e.g.*, for a whole world portfolio with $n = 10000$ and $k = 100$, where we obtain a 10000 fold decrease in solve time; see §5.6.

3.4 Return and risk forecasts

Here we briefly discuss the forecasting of μ and Σ (or F, Σ^{f}, and D in a factor model). Markowitz himself did not address the question of estimating μ and Σ; when asked by practitioners how one should choose these forecasts, his reply was [SS23]

"*That's your job, not mine.*"

It is well documented that poor or naïve estimates of these, *e.g.*, the sample mean and covariance, can yield poor portfolio performance [Mic89]. But even reasonable forecasts will have errors, which can degrade performance. We show some methods to mitigate this forecast uncertainty in §3.5.

Asset returns estimate. The expected returns vector μ is by far the most important parameter in the portfolio construction process, and methods for estimating it, or the factor and idiosyncratic return means are for obvious reasons in general proprietary. It is also the most challenging data to estimate. There is no consensus on how to estimate the mean returns, and the literature is vast.

Regularization methods can improve mean estimates. As an example, the Black-Litterman model [BL90] allows a portfolio manager to incorporate her views on how the expected returns differ from the market consensus, and in essence acts as a form of regularization of the portfolio toward the market portfolio. Another method that serves implicitly as regularization is winsorization, where the mean estimates are clipped when they go outside a specified range [WZ07], [GK00, Chap. 14]. Yet another method is cross-sectionalization, where the preliminary estimate of returns μ is replaced with $\tilde{\mu}$, the same values monotonically mapped to (approximately) a Gaussian distribution [GK00, Chap. 14].

Return covariance estimate. There are many ways to estimate the covariance matrix, with or without a factor model. Approaches that work well in practice include the exponentially weighted moving average (EWMA) [OS96], dynamic conditional correlation (DCC) [Eng02], and iterated EWMA [BB22]. For a detailed discussion on how to estimate a covariance matrix for financial return data, see [JOP+23] and the references therein.

3.5 Making return and risk forecasts robust

In this section we address methods to mitigate the impact of forecast errors in return and covariance estimation, which can lead to poor performance. This directly addresses one of the main criticisms of the Markowitz method, that it is too sensitive to estimation errors. Here, we briefly review how to address robust return mean and covariance estimation, and refer the reader to [BBD+17, §4.3] and [FKPF07, TK04] for more detailed discussions.

Robust return forecast. We model our uncertainty in the mean return vector by giving an interval of possible values for each return mean. We let $\mu \in \mathbf{R}^n$ denote our nominal estimate of the return means, and we take the nonnegative vector $\rho \in \mathbf{R}^n_+$ to describe the half-width or radius of the uncertainty intervals. Thus we imagine that the return can be any vector of the form $\mu + \delta$, where $|\delta_i| \leq \rho_i$. For example $\mu_i = -0.0010$ and $\rho_i = 0.0005$ means that the mean return for asset i lies in the range $[-15, -5]$ bps.

We define the *worst-case mean portfolio return* as the minimum possible mean portfolio return consistent with the given ranges of asset return means:

$$R^{\mathrm{wc}} = \min\{(\mu+\delta)^T w \mid |\delta| \leq \rho\}.$$

We can think of this as an adversarial game. The portfolio manager (PM) chooses the portfolio w, and an adversary then chooses the worst mean return consistent with the given uncertainty intervals. This second step has an obvious solution: We choose $\mu_i - \rho_i$ when $w_i \geq 0$, and we choose $\mu_i + \rho_i$ when $w_i < 0$. In words: For long positions the worst return is the minimum possible; for short positions the worst return is the maximum possible. With this observation, we obtain a simple formula for the worst-case portfolio mean return,

$$R^{\mathrm{wc}} = \bar{R} - \rho^T |w|. \tag{16}$$

The first term is the nominal mean return; the second term, which is nonpositive, gives the degradation of return induced by the uncertainty. We refer to $\rho^T|w|$ as the portfolio *return forecast error penalty* in our return forecast. The return forecast error penalty has a nice interpretation as an uncertainty-weighted leverage.

When the portfolio is long-only, so $w \geq 0$, the worst-case asset returns are obvious: they simply take their minimum values, $\mu - \rho$. In this case the worst-case portfolio mean return (16) is the usual mean portfolio return, with each nominal asset return reduced by its uncertainty.

The return forecast uncertainties ρ can be chosen by several methods. One simple method is to set all entries the same, and equal to some quantile of the entries of $|\mu|$, such as the 20th percentile. A more sophisticated method relies on multiple forecasts of the returns, and sets μ as the mean or median forecast, and ρ as some measure of spread, such as standard deviation, of the forecasts.

Robust covariance forecast. We can also consider uncertainty in the covariance matrix. We let Σ denote our nominal estimate of the covariance matrix. We imagine that the covariance matrix has the form $\Sigma + \Delta$ where $\Delta \in \mathbf{S}^n$ (the set of symmetric $n \times n$ matrices) where the perturbation Δ satisfies

$$|\Delta_{ij}| \leq \varrho(\Sigma_{ii}\Sigma_{jj})^{1/2},$$

where $\varrho \in [0, 1)$ defines the level of uncertainty. For example, $\varrho = 0.04$ means that the diagonal elements of the covariance matrix can change by up to 4% (so the volatilites can change by around 2%), and the asset return correlations can change by up to around 4%. (You should not trust anyone who claims that his asset return covariance matrix estimate is more accurate than this.)

We define the *worst-case portfolio risk* as the maximum possible risk over covariance matrices consistent with our uncertainty set,

$$(\sigma^{\mathrm{wc}})^2 = \max\{w^T(\Sigma+\Delta)w \mid |\Delta_{ij}| \leq \varrho(\Sigma_{ii}\Sigma_{jj})^{1/2}\}.$$

This can be expressed analytically as [BBD+17, §4.3]

$$(\sigma^{\mathrm{wc}})^2 = \sigma^2 + \varrho \left(\sum_{i=1}^{n} \Sigma_{ii}^{1/2} |w_i| \right)^2 . \tag{17}$$

The second term is the *covariance forecast error penalty*. It has a nice interpretation as an additive regularization term, the square of a volatility-weighted leverage. The worst-case risk can be expressed using Euclidean norms as

$$\sigma^{\mathrm{wc}} = \left\| \left(\sigma, \sqrt{\varrho} (\mathbf{diag}(\Sigma)^{1/2})^T |w_i| \right) \right\|_2 . \tag{18}$$

When the portfolio is long-only, the worst-case risk (17) can be simplified. In this case, the worst-case risk is the risk using the covariance matrix $\Sigma + \varrho s s^T$, where $s = \mathbf{diag}(\Sigma)^{1/2}$ is vector of asset volatilities, under the nominal covariance.

4 Convex optimization formulation

4.1 Markowitz problem

In this section we assemble the objective terms and constraints described in §2 and §3 into one convex optimization problem. We obtain the Markowitz problem

$$\begin{array}{ll} \text{maximize} & R^{\mathrm{wc}} - \gamma^{\mathrm{hold}} \phi^{\mathrm{hold}}(w, c) - \gamma^{\mathrm{trade}} \phi^{\mathrm{trade}}(z) \\ \text{subject to} & \mathbf{1}^T w + c = 1, \quad z = w - w^{\mathrm{pre}}, \\ & w^{\mathrm{min}} \leq w \leq w^{\mathrm{max}}, \quad L \leq L^{\mathrm{tar}}, \quad c^{\mathrm{min}} \leq c \leq c^{\mathrm{max}}, \\ & z^{\mathrm{min}} \leq z \leq z^{\mathrm{max}}, \quad T \leq T^{\mathrm{tar}}, \\ & \sigma^{\mathrm{wc}} \leq \sigma^{\mathrm{tar}}, \end{array} \tag{19}$$

with variables $w \in \mathbf{R}^n$ and $c \in \mathbf{R}$, and positive parameters γ^{hold} and γ^{trade} that allow us to scale the holding and transaction costs, respectively. Despite the nonlinear and nondifferentiable functions appearing in the objective and constraints, this is a convex optimization problem, which can be very reliably and efficiently solved. We can add other convex objective terms to this problem, such as factor neutrality or liquidation cost limit, or work with active risk and return with a benchmark.

The objective is our forecast of the (robustified, worst-case) net portfolio return, with the holding and transaction costs scaled by the parameters γ^{hold} and γ^{trade}, respectively. The first line of constraints relate the pre-trade portfolio, which is given, and the post-trade portfolio, which is to be chosen. The second line of constraints are weight limits, and the third line contains the trading constraints. The last line of constraints is the (robustified, worst-case) risk limit.

Data. We divide the constants that need to be specified in the problem (19) into two groups, *data* and *parameters*, although the distinction is not sharp. Data are quantities we *observe* (such as the previous portfolio weights) or forecast (such as return means, market volumes, or bid-ask spreads):

- *Pre-trade portfolio weights* w^{pre} and c^{pre}.
- *Asset return forecast* μ.
- *Risk model* Σ, or for a factor model, F, Σ^{f}, and D.
- *Holding cost parameters* κ^{short} and κ^{borrow}.
- *Trading cost parameters* κ^{spread} and κ^{impact} (which in turn depend on the forecast bid-ask spreads, asset volatilities, and market volume).

Parameters. Parameters are quantities that we *choose* in order to obtain good investment performance, or to reflect portfolio manager preferences, or to comply with legal requirements or regulations. These are

- *Target risk* σ^{tar}.
- *Holding and trading scale factors* γ^{hold} and γ^{trade}.
- *Weight and leverage limits* w^{min}, w^{max}, c^{min}, c^{max}, and L^{tar}.
- *Trade and turnover limits* z^{min}, z^{max}, and T^{tar}.
- *Mean and covariance forecast uncertainties* ρ and ϱ.

We list the mean and covariance forecast uncertainties as parameters since they are closer to being chosen than measured or estimated. When the mean return uncertainties are chosen as described above from a collection of return forecasts, they would be closer to data.

Initial default choices for parameters. The target risk, and the weight, leverage, trade, and turnover limits are interpretable and can be assigned reasonable values by the PM. The return and risk uncertainty parameters ρ and ϱ can be chosen as described above. The hold and trade scale factors can be chosen to be around one.

To improve performance the PM will want to adjust or tune these parameter values around their natural or default values, as discussed in §5.4.

4.2 Softening constraints

The Markowitz problem (19) includes a number of constraints. This can present two challenges in practice. First, it can lead to substantial trading, for example to satisfy our leverage or ex-ante worst-case risk limits, even when they would have been violated only slightly, which can lead to poor performance due to excessive trading. Second, the problem can be infeasible, meaning there is no choice of the variables that satisfy all the constraints. This can complicate back-tests or simulations, as well as running the trading policy in production, where such infeasiblilities naturally occur most frequently during periods of market stress, putting the PM under additional pressure.

Soft constraints. Here we explain a standard method in optimization, in which some of the constraints can be softened, which means we allow them to be somewhat violated, if needed. In optimization, softness refers to how much we care about different values of an objective. We can think of the objective as infinitely soft: We will accept any objective value, but we prefer larger values (if we are maximizing). We can think of constraints as infinitely hard: We will not accept any violation of them, even if it is only by a small amount. *Soft constraints*, described below, are in between. They should normally act as constraints, but when needed, they can be violated. When a soft constraint is violated, and by how much, depends on our priorities, with high priority meaning that the constraint should be violated only when absolutely necessary.

Consider a (hard) constraint such as $f \leq f^{\max}$. This means that we will not accept any choice of the variables for which $f > f^{\max}$. To make it a *soft constraint*, we remove the constraint from the problem and form a penalty term

$$\gamma(f - f^{\max})_+$$

which we subtract from the objective, when we are maximizing. The number $(f - f^{\max})_+$ is the *violation* of the original constraint $f \leq f^{\max}$. The positive parameter γ is called the *priority parameter* associated with the softened constraint. In this context, we refer to the parameter $f^{\max}$ as a *target* for the value of f, not a limit. With the softened constraint, we can accept variable choices for which $f > f^{\max}$, but the optimizer tries to avoid this given the penalty paid (in the objective) when this occurs. Softening constraints preserves convexity of a problem.

Markowitz problem with soft constraints. A number of constraints in (19) should be left as (hard) constraints. These include the constraints relating the proposed and previous weights, *i.e.*, the first line of constraints in (19). When the portfolio is long-only, the constraint $w \geq 0$ should be left as a hard constraint, and similarly for a constraint such as $c \geq 0$, *i.e.*, that we do not borrow cash. When a leverage limit is strict or imposed by a mandate, it should be left as a hard constraint; when it is imposed by the portfolio manager to improve performance, or more likely, to help her avoid poor outcomes, it can be softened.

The other constraints in (19) are candidates for softening. Weight and trade limits, including leverage and turnover limits, should be softened (except in the cases described above). The worst-case risk limit $\sigma \leq \sigma^{\text{tar}}$ should be softened, with a risk penalty term

$$\gamma^{\text{risk}}(\sigma - \sigma^{\text{tar}})_+$$

subtracted from the objective. When the associated priority parameter γ^{risk} is chosen appropriately, this allows us to occasionally violate our risk limit a bit when the violation is small. We refer to the softened Markowitz problem as the Markowitz++ problem.

One nice attribute of the Markowitz++ problem is that it is always feasible; the choice $z = 0$, *i.e.*, no trading, is always feasible, even when it is a poor choice. This means that the softened Markowitz problem can be used to define a trading policy that runs with little or no human intervention (with, however, any soft constraints that exceed their targets reported to the portfolio manager).

Priority parameters. When we soften the worst-case risk, leverage, and turnover constraints, we gain several more parameters,

$$\gamma^{\text{risk}}, \quad \gamma^{\text{lev}}, \quad \gamma^{\text{turn}}.$$

Evidently the larger each of these priority parameters is, the more reluctant the optimizer is to violate it. (Here we anthropomorphize the optimization problem solver.) When the priority parameters are large, the associated soft constraints are effectively hard. Beyond these observations, however, it is hard to know what values should be used.

Choosing priority parameters. Here we describe a simple method to obtain reasonable useful initial values for the priority parameters associated with softened constraints. Our method is based on Lagrange multipliers or dual variables. Suppose we solve a problem with hard constraints, and obtain optimal Lagrange multipliers for each of the constraints. If we use these Lagrange multipliers as priorities in a softened version of the problem, all the original constraints will be satisfied. Roughly speaking, the Lagrange multipliers give us values of priorities for which the soft constraints are effectively hard. We would want to use priority values a bit smaller, so that the original constraints can occasionally be violated.

Now we describe the method in detail. We start by solving multiple instances of the problem with hard constraints, for example in a back-test, recording the values of the Lagrange multipliers for each problem instance (when the problem is feasible). We then set the priority parameters to some quantile, such as the 80th percentile, of the Lagrange multipliers. With this choice of priority parameters, we expect (very roughly) the original constraints to hold around 80% of the time. For hard constraints that are only occasionally tight, another method for choosing the priority parameters is as a fraction of the maximum Lagrange multiplier observed.

Using this method we can obtain reasonable starting values of the priority parameters. The final choice of priority parameters is done by back-testing and parameter tuning, starting from these reasonable values, as discussed in §4.4.

4.3 Nonconvex constraints and objectives

All objective terms and constraints discussed so far are convex, and the Markowitz problem (19), and its softened version, are convex optimization problems. They can be reliably and efficiently solved.

Some other constraints and objective terms are not convex. The most obvious one is that the trades must ultimately involve an integer number of shares. As a few other practical examples, we might limit the number of nonzero weights, or insist on a minimum nonzero weight absolute value. When these constraints are added to (19), the problem becomes nonconvex. Great advances have been made in solvers that handle so-called mixed-integer convex problems [KBLG18], and these can be used to solve these portfolio construction problems. The disadvantage is longer solve time, compared to a similar convex problem, and sometimes, dramatically longer solve time if we insist on solving the problem to global

optimality. A convex portfolio construction problem that can be solved in a small fraction of a second can take many seconds, or even minutes or more, to solve when nonconvex constraints are added.

For production, where the problem is solved daily, or even hourly, this is fine. The slowdown incurred with nonconvex optimization is however very bad for back-testing and validation, where many thousands, or hundreds of thousands, of portfolio construction problems are to be solved. One sensible approach is to carry out back-testing using a convex formulation, so as to retain the speed and reliability of a convex optimization, and run a nonconvex version in production. As a variant on this, back-tests using convex optimization can be used for parameter search, and one final back-test with a nonconvex formulation can be used to be sure the results are close. Running backtests using only convex constraints works because the nonconvex constraints typically only have a small impact on the portfolio and its performance.

Heuristics based on convex optimization. Essentially all solvers for nonconvex problems that attempt to find a global solution rely on convex optimization under the hood [HT13]. The issue is that a very large number of convex optimization problems might need to be solved to find a global solution.

But many nonconvex constraints can be handled heuristically by solving just a few convex optimization problem. As a simple example we might simply round the numbers of shares in a trade list to an integer. This rounding should have little effect unless the portfolio value is very small.

Other nonconvex constraints are readily handled by heuristics that involve solving just a handful of convex problems. One general method is called relax-round-solve [DTB18]. We illustrate this method to handle the constraint that the minimum nonzero weight absolute value is 0.001 (10 bps). First we solve the problem ignoring this constraint. If the weights satisfy the constraint, we are done (and the choice is optimal). If not, we set a threshold and divide the assets into those with absolute weight smaller than the threshold, those with weights larger than the threshold, and those which are less than minus the threshold. We then add constraints to the original problem, setting the weights to zero, more than 0.001, and less than -0.001, depending on the weights found in the first problem. These are convex constraints, and when we solve the second time we are guaranteed to satisfy the nonconvex constraint. We thus solve two convex problems. In the first one, we essentially decide which weights will be zero, which will be more than the minimum nonzero long weight, and which will be short more than the minimum. In the second one we adjust all the weights, ensuring that the minimum absolute nonzero weight constraint holds.

4.4 Back-testing and parameter tuning

Back-testing. Back-testing refers to simulating a trading strategy using historical data. To do this we provide the forecasts for all quantities needed, including the mean return and covariance, for Markowitz portfolio construction in each period. In each period these

forecasts, together with the parameters, are sent in to the Markowitz portfolio construction method, which determines a set of trades. We then use the *realized* values of return, volatility, bid-ask spread, and market volume to compute the (simulated) realized net return R_t^{net}, where the subscript gives the time period. Note that while the Markowitz trading engine uses forecasts of various quantities, the simulation uses the realized historical values. This gives a reasonably realistic approximation of what the result would have been, had we actually carried out this trading. (It is still only an approximation, since it uses our particular trading cost model. Of course a more complex or realistic trading cost model could be used for simulation.) The back-test simulation can also include practical aspects like trading only an integer number of shares or blocks of shares. The simulation can also include external cash entering or leaving the portfolio, such as liabilities that must be paid each period.

In the simulation we log the trajectory of the portfolio. We can compute various quantities of interest such as the realized return, volatility, Sharpe or information ratio, turnover, and leverage, all potentially over multiple time periods such as quarters or years. We can determine the portfolio value versus periods, given by

$$V_t = V_1 \prod_{\tau=1}^{t-1} (1 + R_\tau^{\text{net}}),$$

where V_1 is the portfolio value at period $t = 1$. From this we can evaluate quantities like the average or maximum value of drawdown over quarters or years.

Variations. The idea of back-testing or simulating portfolio performance can be used for several other tasks. In one variation on a back-test called a *stress test*, we use historical data modified to be more challenging, *e.g.*, lower returns or higher costs than actually occurred.

Another variation called *performance forecasting* uses data that are simulated or generated, starting from the current portfolio out to some horizon like one year in the future, or the end of current fiscal year. We generate some number of possible future values of quantities such as returns, along with the corresponding forecasts of them, and simulate the performance for each of these. This gives us an idea of what we can expect our future performance to be, for example as a range of values or quantiles.

Yet another variation is a *retrospective what-if* simulation. Here we take a live portfolio and go back, say, three months. Starting from the portfolio holdings at that time, we simulate forward to the present, after making some changes to our trading method, *e.g.*, modifying some parameters. The fact that the current portfolio value would be higher (according to our simulation) if the PM had reduced the target risk three months ago is of course not directly actionable. But it still very useful information for the PM.

Parameter tuning. Perhaps the most important use of back-testing is to help the PM choose parameter values in the Markowitz portfolio construction problem. While some parameters, like the target risk, are given, others are less obvious. For example, how should we choose γ^{trade}? The default value of one is our best guess of what the single period transaction cost will be. But perhaps we get better performance with $\gamma^{\text{trade}} = 2$, which means, roughly

speaking, that we are exaggerating trading cost by a factor of two. The result, of course, is a reduction in trading compared to the default value one. This will result in smaller realized transaction costs, but also, possibly, higher return, or smaller drawdown. The back-test will reveal what would happen in this case (to the limits of the back-test accuracy).

To choose among a set of choices for parameters, we carry out a back-test with each set, and evaluate multiple metrics, such as realized returns, volatility, and turnover. Our optimization problem contains target values for these, based on our forecasts and models; in a back-test we obtain the ex-post or realized values of these metrics.

To make a final choice of parameters, we must scalarize our metrics, *i.e.*, create one scalar metric from them, so we can choose among different sets of parameter values. For example we might choose to maximize Sharpe ratio, subject to other metrics being within specified bounds. Or we could form some kind of weighted combination of the individual metrics.

At the very minimum, a PM should always carry out back-tests in which all of her chosen parameters are, one by one, increased or decreased by, say, 20%. Even with 10 parameters, this requires only 20 back-tests. If any of these back-tests results in substantially improved performance, she will need to explain or defend her choices.

This simple method of changing one parameter at a time can be extended to carry out a crude but often effective parameter search. We cycle over the parameters, increasing or decreasing each and carrying out a back-test. When we find a new set of parameter values that has better performance than the current set of values, we take it as our new values. This continues until increasing or decreasing each parameter value does not improve performance.

Another traditional method of parameter tuning is *gridding*. We choose a small number of candidate values for each parameter, and then carry out a back-test for each combination, evaluating multiple performance metrics. Of course this is practical only when we are choosing just a few parameters, and we consider only a few candidate values for each one. Gridding is often carried out with a first crude parameter gridding, with the candidate values spaced by a factor of ten or so; then, when good values of these parameters are found, a more refined grid search is used to focus in on parameters near the good ones found in the first crude search. In any case there is no reason to find or specify parameter values very accurately; specifying them to even 10% is not needed. For one thing, the back-test itself is only an approximation. To put in a negative light, if a back-test reveals that $\gamma^{\text{trade}} = 2.1$ works well, but that $\gamma^{\text{trade}} = 1.9$ and $\gamma^{\text{trade}} = 2.3$ work poorly, it is very unlikely that our trading method will work well in practice. Similar to the way we want our trading policy to be robust to variations in the input data, we also want it to be robust to variation in the parameters.

More sophisticated parameter search methods can also be used. Many such methods build a statistical model of the good parameter values found so far, and obtain new values to try by sampling from the distribution; see, *e.g.*, [MBK+22] for more discussion. Another option is to obtain not just the value of some composite metric, but also its gradient with respect to the parameters. This very daunting computation can be carried out by automatic differentiation systems that can differentiate through the solution of a convex optimization problem, such as CVXPYlayers [AAB+19, BAB20].

5 Numerical experiments

In this section we present numerical experiments that illustrate the ideas and methods discussed above. In the first set of experiments, described in §5.2, we show the effect of several constraints and objective terms that serve as effective regularizers and improve performance. In §5.4 we illustrate how parameter tuning via back-tests can improve performance, and in §5.6 we show how the methods we describe scale with problem size.

5.1 Data and back-tests

Data. Throughout the experiments we use the same data set, which is based on the stocks in the S&P 100 index. We use daily adjusted close price data from 2000-01-04 to 2023-09-22. We exclude stocks without data for the entire period, and acknowledge that this inherent survivorship bias in the data set would make it unsuitable for a real portfolio construction method, but it is sufficient for our experiment, which is only concerned with the relative performance of the different methods. We end up with a universe of $n = 74$ assets. In addition to the price data, we use bid-ask spread data to estimate the trading costs, as well as the effective federal funds rate [Fed23] for short term borrowing and lending. We make the data set available with the code for reproducibility and experimentation at

https://github.com/cvxgrp/markowitz-reference.

Mean prediction. Simple estimates of the means work poorly, so in the spirit of [BBD+17], we use synthetic return predictions to simulate a proprietary mean prediction method. For each asset, the synthetic returns for each day are given by

$$\hat{r}_t = \alpha(r_t + \epsilon_t),$$

where ϵ_t is a zero-mean Gaussian noise term with variance chosen to obtain a specified information coefficient and r_t is the mean return of the asset in the week starting on day t. We take the noise variance to be $\sigma^2(1/\alpha - 1)$, where α is the square of the information coefficient, and σ^2 is the variance of the return. (These mean predictions are done for each asset separately.) We choose an information coefficient of $\sqrt{\alpha} = 0.15$. Using this parameterization, the sign of the return is predicted correctly in 52.1% of all observations, with this number ranging from 50.3% to 54.1% for the individual assets.

Covariance prediction. For the covariance prediction, we use a simple EWMA estimator, *i.e.*, the covariance matrix at time t is estimated as

$$\hat{\Sigma}_t = \alpha_t \sum_{\tau=1}^{t} \beta^{t-\tau} r_\tau r_\tau^T,$$

where

$$\alpha_t = \left(\sum_{\tau=1}^{t} \beta^{t-\tau} \right)^{-1} = \frac{1-\beta}{1-\beta^t}$$

is the normalization constant, and $\beta \in (0, 1)$ is the decay factor. (We use the second moment as the covariance, since the contribution from the mean term is negligible.) We use a half-life of 125 trading days, which corresponds to a decay factor of $\beta \approx 0.994$. We note that the specific choice of the half-life does not change the results of the experiments qualitatively.

Spread. Our simulations include the transaction cost associated with bid-ask spread. In simulation we use the realized bid-ask spread; for the Markowitz problems we use a simple forecast of spread, the average realized bid-ask spread over the previous five trading days.

Shorting and leverage costs. We use the effective federal funds rate as a proxy for interest on cash for both borrowing and lending. When shorting an asset we add a 5% annualized spread over the effective federal funds rate to approximate the shorting cost in our simulation. For forecasting, we set κ^{short} to 7.5% annualized, and κ^{borrow} to the effective federal funds rate.

Back-tests. We use a simple back-test to evaluate the performance of the different methods. We start with a warm-up period of 500 trading days for our estimators leaving us with 5,686 trading days, or approximately 22 years of data. The first 1,250 trading days (five years) are used to initialize the priority parameters. This leaves us with 4,436 out-of-sample trading days, approximately 17 years. Starting with an initial cash allocation of $1,000,000, we call the portfolio construction method each day to obtain the target weights. We then execute the trades at the closing price, rebalancing the portfolio to the new target weights, taking into account the weight changes due to the returns from the previous day. Buy and sell orders are executed at the ask and bid prices, respectively, and interest is paid on borrowed cash and short stocks, and received on cash holdings.

5.2 Taming Markowitz

In this first experiment we show how a basic Markowitz portfolio construction method can lead to the undesirable behavior that would prompt the alleged deficiencies described in §1.2. We then show how adding just one more reasonable constraint or objective term improves the performance, taming the basic Markowitz method.

Basic Markowitz. We start by solving the basic Markowitz problem (1) for each day in the data set, with the risk target set to 10% annualized volatility. Unsurprisingly the basic Markowitz problem results in poor performance, as seen in the second line of table 1. It has low mean return, high volatility (well above the target 10%), a low Sharpe ratio, high leverage and turnover, and a maximum drawdown of almost 80%. This basic Markowitz portfolio performs considerably worse than an equal-weighted portfolio, which we give as a baseline on the top line of table 1.

	Return	Volatility	Sharpe	Turnover	Leverage	Drawdown
Equal weight	14.1%	20.1%	0.66	1.2	1.0	50.5%
Basic Markowitz	3.7%	14.5%	0.19	1145.2	9.3	78.9%
Weight-limited	20.2%	11.5%	1.69	638.4	5.1	30.0%
Leverage-limited	22.9%	11.9%	1.86	383.6	1.6	14.9%
Turnover-limited	19.0%	11.8%	1.54	26.1	6.5	25.0%
Robust	15.7%	9.0%	1.64	458.8	3.2	24.7%
Markowitz++	38.6%	8.7%	4.32	28.0	1.8	7.0%
Tuned Markowitz++	41.8%	8.8%	4.65	38.6	1.6	6.4%

Table 1: Back-test results for different trading policies.

Markowitz with regularization. In a series of four experiments we show how adding just one more reasonable constraint or objective term to the basic Markowitz method can greatly improve the performance.

In the first experiment we add portfolio weight limits of 10% for long positions and -5% for short positions. We limit the cash weight to lie between -5% and 100% (which guarantees feasibility). Adding these asset and cash weight limits leads to a significant improvement in the performance of the portfolio shown in the third row of table 1, with the Sharpe ratio increasing to 1.69 (from 0.19), and the maximum drawdown decreasing to 30%. In addition the realized volatility, 11.5%, is closer to the target value 10% than the basic Markowitz trading policy. The turnover is still very high, however, and the maximum leverage is still large at above 5.

In the second experiment we add a leverage limit to the basic Markowitz problem, with $L^{\mathrm{tar}} = 1.6$. This one additional constraint also greatly improves performance, as seen in the fourth row of table 1, but with a lower turnover and (not surprisingly) a lower maximum leverage, which is at our target value 1.6.

Our third experiment adds a turnover limit of $T^{\mathrm{tar}} = 25$ to the basic Markowitz problem. This additional constraint drops the turnover considerably, to a value near the target, but still achieves high return, Sharpe ratio, and even lower maximum drawdown.

Our fourth experiment adds robustness to the return and risk forecasts. As simple choices we set all entries of ρ to the 20th percentile of the absolute value of the return forecast at each time step, and use $\varrho = 0.02$. This robustification also improves performance. Not surprisingly the realized risk comes in under our target, since we use the robust risk ex-ante; we could achieve realized risk closer to our desired target 10% by increasing the target to something like 11.5% (which we didn't do).

5.3 Markowitz++

In the four experiments described above, we see that adding just one reasonable additional constraint or objective term to the basic Markowitz problem greatly improves the perfor-

mance. In our last experiment, we include all of these constraints and terms, with parameters

$$\gamma^{\text{hold}} = 1, \quad \gamma^{\text{trade}} = 1, \quad \sigma^{\text{tar}} = 0.10$$
$$c^{\min} = -0.05, \quad c^{\max} = 1.00, \quad w^{\min} = -0.05, \quad w^{\max} = 0.10, \quad L^{\text{tar}} = 1.6$$
$$z^{\min} = -0.10, \quad z^{\max} = 0.10, \quad T^{\text{tar}} = 25.$$

The mean uncertainty parameter ρ is chosen as the 20th percentile of the absolute value of the return forecast, and $\varrho = 0.02$. We soften the risk target, leverage limit, and turnover limit, using the priority parameters

$$\gamma^{\text{risk}} = 5 \times 10^{-2}, \quad \gamma^{\text{lev}} = 5 \times 10^{-4}, \quad \gamma^{\text{turn}} = 2.5 \times 10^{-3}.$$

These were chosen as the 70th percentiles for the corresponding Lagrange multipliers of the hard constraints in the basic Markowitz problem for the risk and turnover limits, and as 25% of the maximum Lagrange multiplier for the leverage limit, over the five years leading up to the out-of-sample study. (We selected γ^{lev} this way since the corresponding constraint was active very rarely in the basic Markowitz problem.)

With this Markowitz++ method, we obtain the performance listed in the second from bottom row of table 1. It is considerably better than the performance achieved by adding just one additional constraint, as in the four previous experiments, and very much better than the basic Markowitz method. Not surprisingly it achieves good performance on all metrics, with a high Sharpe ratio, reasonable tracking of our volatility target, modest turnover and leverage, and very small maximum drawdown. When the parameters are tuned annually, as detailed in the next section, we see even more improvement, as shown in the bottom row of table 1.

The Sharpe ratios on the bottom two rows are high. We remind the reader that our data has survivorship bias and uses synthetic (but realistic) mean return forecasts, so the performance should not be thought of as implementable. But the differences in performance of the different trading methods is significant.

5.4 Parameter tuning

In this section we show how parameter tuning can be used to improve the performance of the portfolio construction method. We will tune the parameters γ^{hold}, γ^{trade}, γ^{lev}, γ^{risk}, and γ^{turn}, keeping the other parameters fixed. We start from the values used in Markowitz++.

Experimental setup. We tune the parameters at the start of every year, on the previous two years of data, and then fix the tuned parameters for the following year. To tune the parameters we use the simple cyclic tuning method described in §4.4. We cycle through the parameters one by one. Each time a parameter is encountered in the loop, we increase it by 25%; if this yields an improvement in the performance (defined below), we keep the new value and continue with the next parameter; if not, we decrease the parameter by 20% and check if this yields an improvement. We continue this process until a full loop through all

parameters does not yield any improvement. By improvement in performance we mean that all the following are satisfied:

- The in-sample Sharpe ratio increases.
- The in-sample annualized turnover is no more than 50.
- The in-sample maximum leverage is no more than 2.
- The in-sample annualized volatility is no more than 15%.

Results. Tuning the parameters every year yields the performance given in the last row of table 1. We see a modest but significant boost in performance over untuned Markowitz++.

The tuned parameters over time are shown in figure 1. We can note several intuitive patterns in the parameter values. For example, γ^{risk} increases during 2008 to account for the high uncertainty in the market during this period. Similarly, γ^{turn} decreases during the same period, likely to allow us to trade more freely to satisfy the other constraints; interestingly γ^{trade} increases during the same period, likely to push us toward more liquid stocks when trading increases. During the same period γ^{lev} increases to reduce leverage. Similar patterns can be observed in 2020.

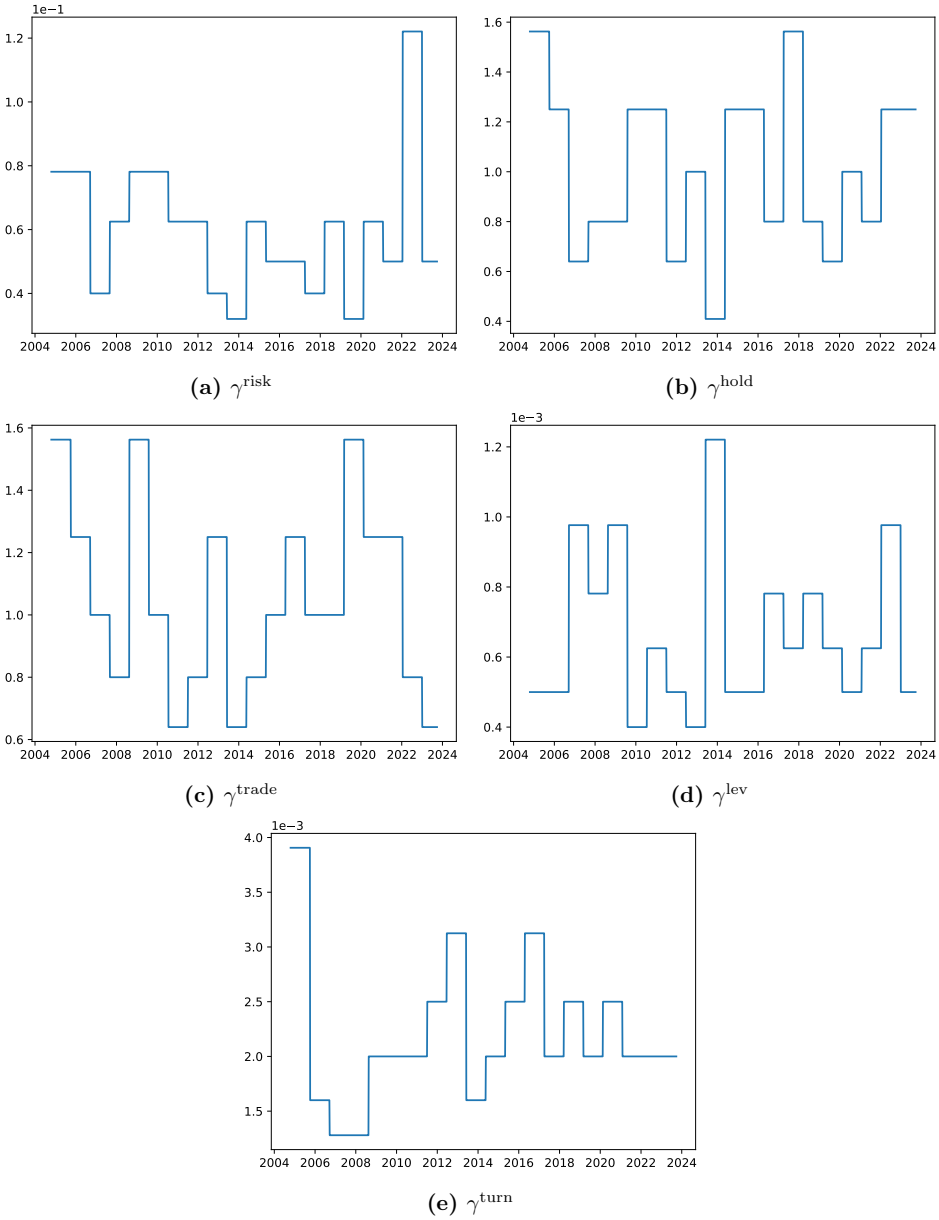
Tuning evolution. Here we show an example of the evolution of tuning, showing both in- and out-of-sample values of Sharpe ratio, turnover, leverage, and volatility. The in-sample period is April 19, 2016 to March 19, 2018, and the out-of-sample period March 20, 2018 to March 4, 2019. These are shown in figure 2. This tuning process converged after 45 back-tests to the parameter values

$$\gamma^{\text{risk}} = 4 \times 10^{-2}, \quad \gamma^{\text{hold}} = 0.64, \quad \gamma^{\text{trade}} = 0.64, \quad \gamma^{\text{lev}} = 5 \times 10^{-4}, \quad \gamma^{\text{turn}} = 1.6 \times 10^{-3}.$$

We can see that tuning increases the Sharpe ratio both in- and out-of-sample, while keeping the leverage, turnover, and volatility reasonable. In this example we end up changing 4 of our 5 adjustable parameters, although not by much, which shows that our initial default parameter values were already quite good. Still, we obtain a significant boost in performance.

5.5 Annual performance

The performance analyses described above and summarized in table 1 give aggregate metrics over a 17 year out-of-sample period, long enough to include multiple distinct market regimes as a well as a few market crashes. For such a long back-test, it is interesting to see how the performance in individual years varies with different market regimes. The realized annual return, volatility, and Sharpe ratio are shown in figure 3, for basic Markowitz, equal weights, and tuned Markowitz++. Here we see that Markowitz++ not only gives the performance improvements seen in table 1, but in addition has less variability in performance across different market regimes.

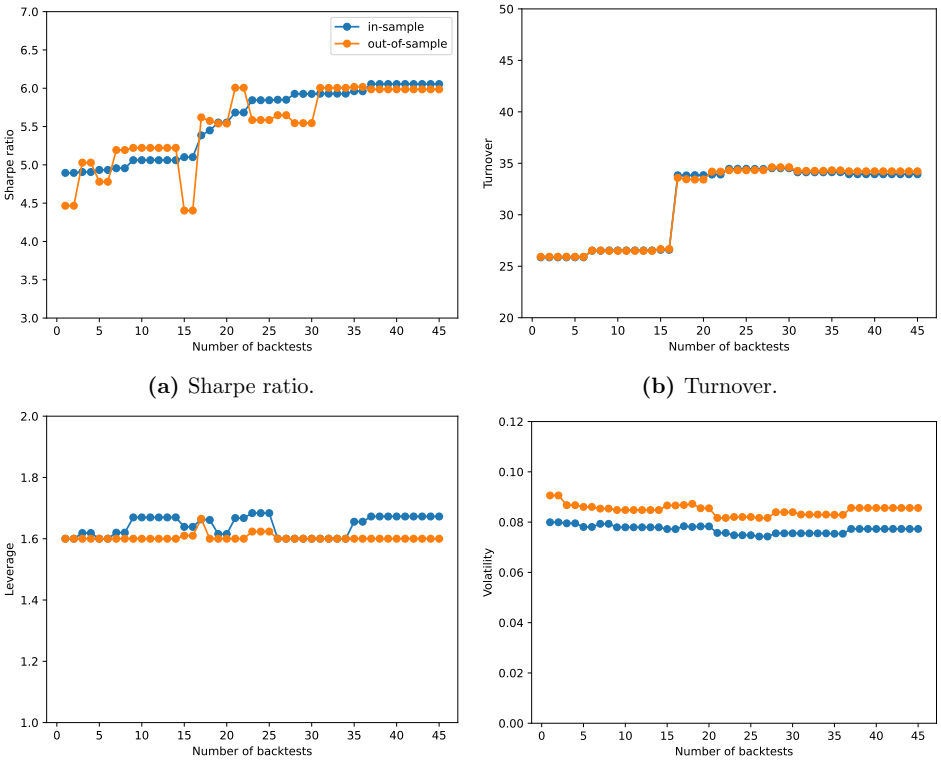


(a) γ^{risk} **(b)** γ^{hold}

(c) γ^{trade} **(d)** γ^{lev}

(e) γ^{turn}

Figure 1: Tuned parameters over time.



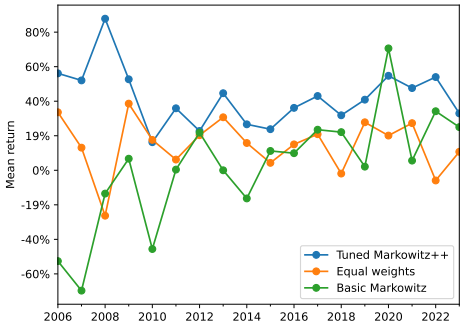
(a) Sharpe ratio.

(b) Turnover.

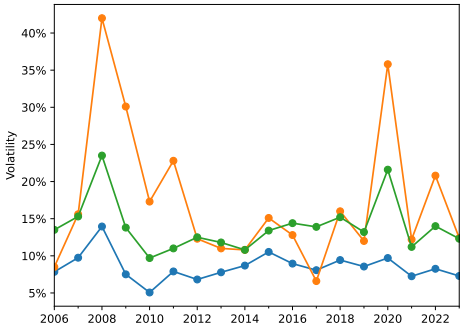
(c) Leverage.

(d) Volatility.

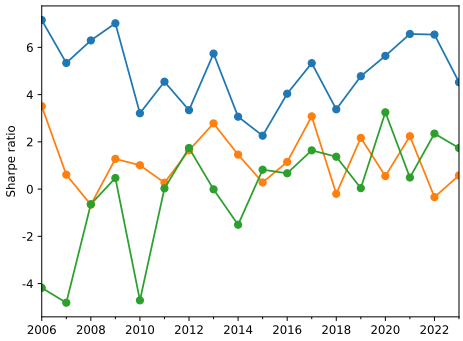
Figure 2: Parameter tuning results.



(a) Yearly annualized returns.



(b) Yearly annualized volatilities.



(c) Yearly annualized mean Sharpe ratios.

Figure 3: Yearly annualized metrics for the equal weight portfolio, basic Markowitz, and tuned Markowitz++.

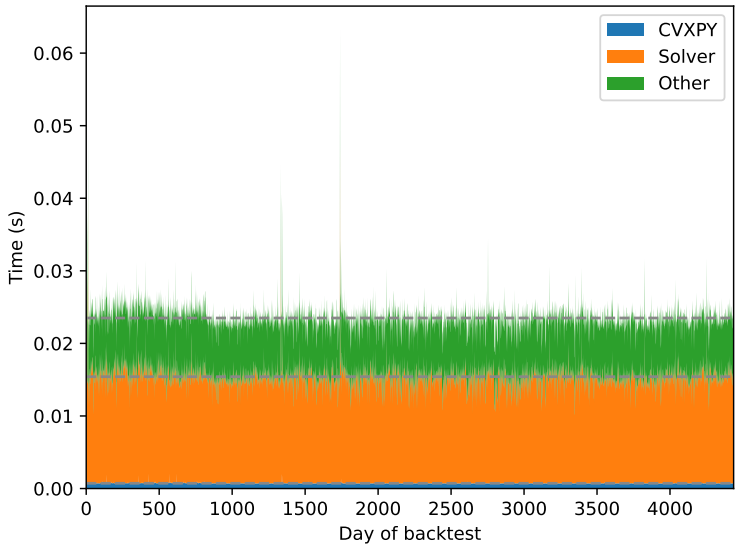


Figure 4: Timing results for the Markowitz problem with a single backtest setting.

5.6 Scaling

We now turn from the performance of the portfolios to the algorithmic performance of the portfolio construction method itself.

Small problems. We start with the small problem used in the previous section, with $n = 74$ assets, and without a factor risk model. Figure 4 shows the time required for each of the 4,436 days in the back-test, broken down into updating and logging (shown in green), CVXPY overhead (shown in blue, negligible), and solver time, the time required to solve the resulting cone program. (We do not count factorizing the covariance matrix, or computing the mean forecasts, since these are done ahead of time, and the time is amortized across all back-tests.)

The 17 year back-test, which involve solving 4,436 problems, takes around 104 seconds on a MacBook Pro with an M1 Pro processor, or about 23ms per day on average. About 63% of the time is spent in the solver, which in this case is MOSEK [ApS20], with other solvers giving similar results, including open-source solvers such as ECOS [DCB13], Clarabel [GC24], and SCS [OCPB16]. Only 3% of the time is spent in the compilation step using CVXPY. The averages for each component of the timings are indicated by the horizontal lines in the figure.

Assets n	Factors k	Solve time (s)
100	10	0.01
500	20	0.07
500	50	0.10
2,000	50	0.23
2,000	100	0.22
10,000	50	0.65
10,000	100	0.89
50,000	200	9.00
50,000	500	17.77

Table 2: Average solve times for Markowitz++ problem for MOSEK, for different problem sizes.

For a small problem like this one, we can carry out a one-year back-test (around 250 trading days) in around six seconds, on a single thread. A single processor with 32 threads can carry out around 20,000 one-year back-tests in an hour. There is little excuse for a PM who does not carry out many back-tests, even if only to vary the parameters around their chosen values.

Large problems. We now investigate the scaling of the method with problem size. As outlined in §3.3, a factor model improves the scaling from $O(n^3)$ to $O(nk^2)$. To illustrate this, we solve the Markowitz problem for different values of n and k using randomly generated but realistic data. Table 2 shows the average solve time for each problem size across 30 instances using MOSEK. (Solve times with open-source solvers such as Clarabel were a bit longer.) We can see that even very large problems can be solved with stunning speed.

We solved many more problems than those shown in table 2, and used the solve times to fit a log-log model, approximating the solve time as an^bk^c, with parameters a, b, c. We obtained coefficients $b = 0.79$ and $c = 1.72$, consistent with the theoretical scaling of $O(nk^2)$.

When the problems are even larger, generic software reaches its limits. In such cases, users may consider switching to first-order methods like the Alternating Direction Method of Multipliers (ADMM) [MGBK22, Fou23, FB18, PB13, BPC+11], which can offer better scalability and efficiency for very large problems.

6 Conclusions

It was Markowitz's great insight to formulate the choice of an investment portfolio as an optimization problem that trades off multiple objectives, originally just expected return and risk, taken to be the standard deviation of the portfolio return. His original proposal yielded an optimization problem with an analytical solution for the long-short case, and a QP for the long-only case, both of which were tractable to solve (for very small problems) even in the 1950s. Since then, stunning advances in computer power, together with advances in optimization, now allow us to formulate and solve much more complex optimization problems, that directly handle various practical constraints and mitigate the effects of forecasting errors. We can solve these problems fast enough that very large numbers of back-tests can be carried out, to give us a good idea of the performance we can expect, and to help choose good values of the parameters. It is hardly surprising that these methods are widely used in quantitative trading today.

While we have vastly more powerful computers, far better software, and easier access to data, than Markowitz did in 1952, we feel that the more complex Markowitz++ optimization problem simply realizes his original idea of an optimization-based portfolio construction method that takes multiple objectives into account.

Acknowledgments

The authors thank Trevor Hastie, Mykel Kochenderfer, Mark Mueller, and Rishi Narang for helpful feedback on an earlier version of this paper. We gratefully acknowledge support from the Office of Naval Research. This work was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems. Kasper Johansson was partially funded by the Sweden America Foundation.

References

[AAB+19] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[AC00] R. Almgren and N. Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, pages 5–39, 2000.

[AGv11] A. Ang, S. Gorovyy, and G. van Inwegen. Hedge fund leverage. *Journal of Financial Economics*, 102(1):102–126, 2011.

[ApS20] MOSEK ApS. MOSEK modeling cookbook, 2020.

[ApS23] MOSEK ApS. MOSEK portfolio optimization cookbook, 2023.

[AVDB18] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[BAB20] S. Boyd, A. Agrawal, and S. Barratt. Embedded convex optimization for control. In *Proceedings of the 59th IEEE Conference on Decision and Control*, 2020.

[Bai03] J. Bai. Inferential theory for factor models of large dimensions. *Econometrica*, 71(1):135–171, 2003.

[BB21] S. Barratt and S. Boyd. Stochastic control with affine dynamics and extended quadratic costs. *IEEE Transactions on Automatic Control*, 67(1):320–335, 2021.

[BB22] S. Barratt and S. Boyd. Covariance prediction via convex optimization. *Optimization and Engineering*, 2022.

[BBC11] D. Bertsimas, D. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[BBD+17] S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth. Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76, 2017.

[Bel66] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[Ber12] D. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific, 2012.

[BHH+21] M. Bynum, G. Hackebeil, W. Hart, C. Laird, B. Nicholson, J. Siirola, J. Watson, and D. Woodruff. *Pyomo–Optimization modeling in Pthon*, volume 67. Springer Science & Business Media, third edition, 2021.

[BL90] F. Black and R. Litterman. Asset allocation: Combining investor views with market equilibrium. *Goldman Sachs Fixed Income Research*, 115(1):7–18, 1990.

[Bla16] L. Blackmore. Autonomous precision landing of space rockets. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium*, volume 46, pages 15–20. The Bridge, 2016.

[BMOW13] S. Boyd, M. Mueller, B. O'Donoghue, and Y. Wang. Performance bounds and suboptimal policies for multi-period investment. *Foundations and Trends in Optimization*, 1(1):1–72, 2013.

[BN08] J. Bai and S. Ng. Large dimensional factor analysis. *Foundations and Trends in Econometrics*, 3(2):89–163, 2008.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[Bra10] M. Brandt. Portfolio choice problems. In *Handbook of financial econometrics: Tools and techniques*, pages 269–336. Elsevier, 2010.

[Bra13] P. Brandimarte. *Numerical methods in finance and economics: a MATLAB-based introduction*. John Wiley & Sons, 2013.

[BS07] H. Beyer and B. Sendhoff. Robust optimization—a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007.

[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*, volume 28. Princeton University Press, 2009.

[BTN02] A. Ben-Tal and A. Nemirovski. Robust optimization—methodology and applications. *Mathematical Programming*, 92:453–480, 2002.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[BV18] S. Boyd and L. Vandenberghe. *Introduction to applied linear algebra: Vectors, matrices, and least squares*. Cambridge University Press, 2018.

[BV23] S. Boyd and L. Vandenberghe. Convex optimization additional exercises. https://github.com/cvxgrp/cvxbook_additional_exercises, 2023.

[Caj22] D. Cajas. Convex optimization of portfolio kurtosis. *SSRN Electronic Journal*, 2022.

[Cpl09] IBM ILOG Cplex. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[CT06] G. Cornuejols and R. Tütüncü. *Optimization methods in finance*, volume 5. Cambridge University Press, 2006.

[CY16] J. Chen and M. Yuan. Efficient portfolio selection in a large market. *Journal of Financial Econometrics*, 14(3):496–524, 2016.

[Dan51] G. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 13:339–347, 1951.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DCB13] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *2013 European control conference*, pages 3071–3076. IEEE, 2013.

[Del23] H. Delatte. skfolio. https://github.com/skfolio/skfolio, 2023.

[dF40] B. de Finetti. Il problema dei "pieni". *Giornale dell'Istituto Italiano degli Attuari*, 11:1–88, 1940.

[DP16] M. De Prado. Building diversified portfolios that outperform out of sample. *Journal of Portfolio Management*, 42(4):59–69, 2016.

[DTB18] S. Diamond, R. Takapoui, and S. Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

[Eng02] R. Engle. Dynamic conditional correlation. *Journal of Business & Economic Statistics*, 20(3):339–350, 2002.

[Fam65] E. Fama. The behavior of stock-market prices. *Journal of Business*, 38(1):34–105, 1965.

[FB18] C. Fougner and S. Boyd. *Parameter Selection and Preconditioning for a Graph Form Solver*, pages 41–61. Springer International Publishing, 2018.

[Fed23] Federal Reserve Bank of St. Louis. Federal funds effective rate (DFF). https://fred.stlouisfed.org/series/DFF, 2023. Accessed on 2023-09-22.

[FF92] E. Fama and K. French. The cross-section of expected stock returns. *Journal of Finance*, 47(2):427–465, 1992.

[FF93] E. Fama and K. French. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56, 1993.

[FKPF07] F. Fabozzi, P. Kolm, D. Pachamanova, and S. Focardi. *Robust portfolio optimization and management*. John Wiley & Sons, 2007.

[FNB17] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *arXiv preprint arXiv:1711.07582*, 2017.

[Fou23] C. Fougner. POGS - proximal operator graph solver. https://foges.github.io/pogs/, 2023. Accessed on 2023-09-22.

[GB14] M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming, version 2.1, 2014.

[GC24] P. Goulart and Y. Chen. Clarabel documentation, 2024. Accessed on 2023-09-07.

[GD09] B. Graham and D. Dodd. *Security analysis*. McGraw-Hill, 6th edition, 2009.

[GK00] R. Grinold and R. Kahn. *Active portfolio management*. McGraw Hill, 2000.

[GK20] R. Grinold and R. Kahn. *Advances in Active Portfolio Management: New Developments in Quantitative Investing*. McGraw-Hill, 2020.

[Glo23] SRL Global. Portfolio visualizer. https://www.portfoliovisualizer.com/, 2023.

[GMT14] V. Gabrel, C. Murat, and A. Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014.

[GPS22] L. Goldberg, A. Papanicolaou, and A. Shkolnik. The dispersion bias. *SIAM Journal on Financial Mathematics*, 13(2):521–550, 2022.

[Gra73] B. Graham. *The intelligent investor*. New York: Harper, 4th edition, 1973.

[Gro23] Stanford University Convex Optimization Group. CVXMARKOWITZ. https://github.com/cvxgrp/cvxmarkowitz, 2023.

[Gue10] J. Guerard, editor. *Handbook of Portfolio Construction*. Springer, 2010.

[Gur23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[HT13] R. Horst and H. Tuy. *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: Data mining, inference, and prediction*, volume 2. Springer, 2009.

[HWW11] W. Hart, J. Watson, and D. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[Isi21] M. Isichenko. *Quantitative portfolio management: The art and science of statistical arbitrage*. John Wiley & Sons, 2021.

[JL13] B. Jacobs and K. Levy. Leverage aversion, efficient frontiers, and the efficient region. *Journal of Portfolio Management*, 39(3):54–64, 2013.

[JM03] R. Jagannathan and T. Ma. Risk reduction in large portfolios: Why imposing the wrong constraints helps. *Journal of Finance*, 58(4):1651–1683, 2003.

[JOP+23] K. Johansson, G. Ogut, M. Pelger, T. Schmelzer, and S. Boyd. A simple method for predicting covariance matrices of financial returns. *Foundations and Trends in Econometrics*, 12(4):324–407, 2023.

[KBLG18] J. Kronqvist, D. Bernal, A. Lundell, and I. Grossmann. A review and comparison of solvers for convex MINLP. *Optimization and Engineering*, 20(2):397–455, 2018.

[KDG96] I. Khalil, J. Doyle, and K. Glover. *Robust and optimal control*. Prentice Hall, 1996.

[Koc15] M. Kochenderfer. *Decision making under uncertainty: theory and application*. MIT Press, 2015.

[KTF14] P. Kolm, R. Tütüncü, and F. Fabozzi. 60 years of portfolio optimization: Practical challenges and current trends. *European Journal of Operational Research*, 234(2):356–371, 2014.

[KWW22] M. Kochenderfer, T. Wheeler, and K. Wray. *Algorithms for decision making*. MIT Press, 2022.

[LB23] E. Luxenberg and S. Boyd. Portfolio construction with Gaussian mixture returns and exponential utility via convex optimization. *Optimization and Engineering*, pages 1–20, 2023.

[LDD+23] M. Lubin, O. Dowson, J. Dias Garcia, J. Huchette, B. Legat, and J. Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 2023.

[LEB09] M. Leibowitz, S. Emrich, and A. Bova. *Modern Portfolio Management: Active Long/Short 130/30 Equity Strategies*. Wiley, 2009.

[LM79] H. Levy and H. Markowitz. Approximating expected utility by a function of mean and variance. *American Economic Review*, 69(3):308–317, 1979.

[Lob00] M. Lobo. *Robust and convex optimization with applications in finance*. PhD thesis, Stanford university, 2000.

[Loe83] T. Loeb. Trading cost: The critical link between investment information and results. *Financial Analysts Journal*, 39(3):39–44, 1983.

[Lof04] J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*, pages 284–289. IEEE, 2004.

[LP20a] M. Lettau and M. Pelger. Estimating latent asset-pricing factors. *Journal of Econometrics*, 218(1):1–31, 2020.

[LP20b] M. Lettau and M. Pelger. Factors that fit the time series and cross-section of stock returns. *Review of Financial Studies*, 33(5):2274–2325, 2020.

[LUM22] X. Li, A. Uysal, and J. Mulvey. Multi-period portfolio optimization using model predictive control with mean-variance and risk parity frameworks. *European Journal of Operational Research*, 299(3):1158–1176, 2022.

[LVBL98a] M. Lobo, L. Vandenberghe, S. Boyd, and H Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1):193–228, 1998.

[LVBL98b] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and Its Applications*, 284(1-3):193–228, 1998.

[LW04] O. Ledoit and M. Wolf. Honey, I shrunk the sample covariance matrix. *Journal of Portfolio Management*, 30(4):110–119, 2004.

[Mar52] H. Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77–91, 1952.

[Mar59] H. Markowitz. *Portfolio selection: Efficient diversification of investments*. John Wiley, 1959.

[Mar99] H. Markowitz. The early history of portfolio theory: 1600–1960. *Financial Analysts Journal*, 55:5–16, 7 1999.

[Mar09] H. Markowitz. *Harry Markowitz: Selected works*, volume 1. World Scientific, 2009.

[Mar19] H. Markowitz. *From Portfolio Theory to Practice: An AQR interview with Harry Markowitz*. AQR Capital Management, 2019.

[Mar21] R. Martin. Pyportfolioopt: portfolio optimization in Python. *Journal of Open Source Software*, 6(61):3066, 2021.

[Mar23] H. Markowitz. Nobel Prize lecture, 2023. Accessed on 2023-12-13.

[Mat23] MathWorks. MATLAB financial toolbox documentation, 2023.

[MB12] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13:1–27, 2012.

[MB14] H. Markowitz and K. Blay. *Risk-return analysis: The theory and practice of rational investing*, volume 1. McGraw Hill, 2014.

[MBK+22] G. Maher, S. Boyd, M. Kochenderfer, C. Matache, D. Reuter, A. Ulitsky, S. Yukhymuk, and L. Kopman. A light-weight multi-objective asynchronous hyper-parameter optimizer. *arXiv preprint arXiv:2202.07735*, 2022.

[MGBK22] N. Moehle, J. Gindi, S. Boyd, and M. Kochenderfer. Portfolio construction as linearly constrained separable optimization. *Optimization and Engineering*, 24(3):1667–1687, 2022.

[Mic89] R. Michaud. The Markowitz optimization enigma: Is 'optimized' optimal? *Financial analysts journal*, 45(1):31–42, 1989.

[MM08] R. Michaud and R. Michaud. *Efficient asset management: a practical guide to stock portfolio optimization and asset allocation*. Oxford University Press, 2008.

[MRT10] S. Maillard, T. Roncalli, and J. Teïletche. The properties of equally weighted risk contribution portfolios. *Journal of Portfolio Management*, 36(4):60–70, 2010.

[MTB14] I. Mastromatteo, B. Tóth, and J. Bouchaud. Agent-based models for latent liquidity and concave price impact. *Physical Review E*, 89:042805, Apr 2014.

[Mul93] P. Muller. *Empirical tests of biases in equity portfolio optimization: What do ex-ante mean-variance optimal portfolios look like ex-post?*, pages 80–98. Cambridge University Press, 1993.

[Nar09] R. Narang. *Inside the Black Box: The Simple Truth about Quantitative Trading*. J. Wiley & Sons, 2009.

[NN92] Y. Nesterov and A. Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, Philadelphia, PA, 1994.

[OCPB16] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169:1042–1068, 2016.

[OS96] J. Ongerstaey and M. Spencer. *RiskMetrics: Technical Document*. JP Morgan and Reuters, 1996.

[PB13] N. Parikh and S. Boyd. Block splitting for distributed optimization. *Mathematical Programming Computation*, 6(1):77–102, 2013.

[Por23a] Portfolio123. Portfolio123 website. https://www.portfolio123.com/, 2023.

[Por23b] PortfoliosLab. PortfoliosLab website. https://portfolioslab.com/, 2023.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PX22a] M. Pelger and R. Xiong. Interpretable sparse proximate factors for large dimensions. *Journal of Business & Economic Statistics*, 40(4):1642–1664, 2022.

[PX22b] M. Pelger and R. Xiong. State-varying factor models of large dimensions. *Journal of Business & Economic Statistics*, 40(3):1315–1333, 2022.

[Qon23] Qontigo. Axioma by Qontigo. https://qontigo.com/products/axioma-risk, 2023.

[Ros84] B. Rosenberg. Prediction of common stock investment risk. *Journal of Portfolio Management*, pages 44–53, 1984.

[RRJ12] Engle R., Ferstenberg R., and Russell J. Measuring and modeling execution cost and risk. *Journal of Portfolio Management*, 38(2):14–28, 2012.

[Rub06] M. Rubinstein. Bruno de Finetti and mean-variance portfolio selection. *Journal of Investment Management*, 4(3), 2006.

[SBD+22] M. Schaller, G. Banjac, S. Diamond, A. Agrawal, B. Stellato, and S. Boyd. Embedded code generation with CVXPY. *IEEE Control Systems Letters*, 6:2653–2658, 2022.

[SBG+20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[SDR21] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on stochastic programming: Modeling and theory*. SIAM, 2021.

[SH13] T. Schmelzer and R. Hauser. Seven sins in portfolio optimization. *arXiv preprint arXiv:1310.3396*, 2013.

[Sha63] W. Sharpe. A simplified model for portfolio analysis. *Management Science*, 9(2):277–293, 1963.

[She96] A. Sheikh. BARRA's risk models. *Barra Research Insights*, pages 1–24, 1996.

[Shk23] A. Shkolnik. On the markowitz enigma for minimum variance. In *Seminar of the Advanced Financial Technologies Laboratory*, December 7 2023. Talk presented at Stanford University.

[SR20] Perrin S. and T. Roncalli. *Machine Learning Optimization Algorithms & Portfolio Allocation*. John Wiley & Sons, 2020.

[SS23] S. Sexauer and L. Siegel. Harry Markowitz and the philosopher's stone. Email conversation, 2023. Personal communication.

[Stu99] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[SXD20] E. Sarmas, P. Xidonas, and H. Doukas. *Multicriteria portfolio construction with python*. Springer, 2020.

[TA77] A. Tikhonov and V. Arsenin. *Solutions of ill-posed problems*. V. H. Winston & Sons, 1977. Translated from Russian.

[Tha23] Hudson & Thames. Hudson & Thames website. https://hudsonthames.org/, 2023.

[The23] The QuantLib contributors. QuantLib: a free/open-source library for quantitative finance. https://github.com/lballabio/QuantLib, 2023.

[TK04] R. Tütüncü and M. Koenig. Robust asset allocation. *Annals of Operations Research*, 132(1):157–187, 2004.

[TLD+11] B. Tóth, Y. Lemperiere, C. Deremble, J. De Lataillade, J. Kockelkoren, and J. Bouchaud. Anomalous price impact and the critical nature of liquidity in financial markets. *Physical Review X*, 1(2):021006, 2011.

[TTT99] K. Toh, M. Todd, and R. Tütüncü. SDPT3—a MATLAB software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1-4):545–581, 1999.

[UMZ+14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28. IEEE, 2014.

[VB96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[VNM47] J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior, 2nd rev.* Princeton University Press, 1947.

[Wil38] J. Williams. *The theory of investment value*. Cambridge, MA: Harvard University Press, 1938.

[Wol59] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, 1959.

[WZ07] R. Welsch and X. Zhou. Application of robust statistics to asset allocation models. *REVSTAT–Statistical Journal*, pages 97–114, 2007.

[ZD98] K. Zhou and J. Doyle. *Essentials of robust control*, volume 104. Prentice Hall, 1998.

[ZP21] R. Zhou and D. Palomar. Solving high-order portfolios via successive convex approximation algorithms. *IEEE Transactions on Signal Processing*, 69:892–904, 2021.

A Coding tricks

The problem described in §4.1 can essentially be typed directly into a DSL such as CVXPY, with very few changes. In this section we mention a few simple tricks in formulating the problem (for a DSL) that lead to better performance.

Quadratic forms versus Euclidean norms. Traditional portfolio construction optimization formulations use quadratic forms such as $w^T\Sigma w$. Modern convex optimization solvers can directly handle the Euclidean norm without squaring to obtain a quadratic form. Using norm expressions instead of quadratic forms is often more natural, and has better numerical properties. For example a risk limit, traditionally expressed using a quadratic form as

$$w^T\Sigma w \leq (\sigma^{\mathrm{tar}})^2,$$

is better expressed using a Euclidean norm as

$$\|L^T w\|_2 \leq \sigma^{\mathrm{tar}},$$

where L is the Cholesky factor of Σ, *i.e.*, $LL^T = \Sigma$, with L lower triangular with positive diagonal entries.

Exploiting the factor model. To exploit the factor model, it is critical to *never* form the covariance matrix $\Sigma = F\Sigma^{\mathrm{f}}F^T + D$. The first disadvantage of doing this is that we have to (needlessly) store an $n \times n$ matrix, which can be a challenge when n is on the order of tens of thousands. In addition, the solver will be slowed by a dramatic factor as mentioned in §3.3.

To exploit the factor model, we introduce the data matrix $\tilde{F} = FL$, where L is the Cholesky factor of Σ^{f}, so $\tilde{F}\tilde{F}^T = F\Sigma^{\mathrm{f}}F^T$. The portfolio variance is

$$\sigma^2 = w^T\tilde{F}\tilde{F}^T w + w^T D w = \|\tilde{F}^T w\|_2^2 + \|D^{1/2}w\|_2^2,$$

so the risk can be expressed using Euclidean norms as

$$\sigma = \left\| \left(\|\tilde{F}^T w\|_2, \|D^{1/2}w\|_2 \right) \right\|_2.$$

In this expression, the outer norm is of a 2-vector; the inner lefthand norm is of a k-vector, and the inner righthand norm is of an n-vector. Here we should be careful to express D as a diagonal matrix, or to express $D^{1/2}w$ as the elementwise (Hadamard) product of two vectors.

B CVXPY code listing

We provide a reference implementation for the problem described in §4. This implementation is not optimized for performance, contains no error checking, and is provided for illustrative purposes only. For a more performant and robust implementation, we refer the reader to the cvxmarkowitz package [Gro23]. Below, we assume that the data and parameters are already defined in corresponding data structures. The complete code for the reference implementation is available at

https://github.com/cvxgrp/markowitz-reference.

```
import cvxpy as cp

w, c = cp.Variable(data.n_assets), cp.Variable()

z = w - data.w_prev
T = cp.norm1(z) / 2
L = cp.norm1(w)

# worst-case (robust) return
factor_return = (data.F @ data.factor_mean).T @ w
idio_return = data.idio_mean @ w
mean_return = factor_return + idio_return + data.risk_free * c
return_uncertainty = param.rho_mean @ cp.abs(w)
return_wc = mean_return - return_uncertainty

# worst-case (robust) risk
factor_risk = cp.norm2((data.F @ data.factor_covariance_chol).T @ w)
idio_risk = cp.norm2(cp.multiply(data.idio_volas, w))
risk = cp.norm2(cp.hstack([factor_risk, idio_risk]))
risk_uncertainty = param.rho_covariance**0.5 * data.volas @ cp.abs(w)
risk_wc = cp.norm2(cp.hstack([risk, risk_uncertainty]))

asset_holding_cost = data.kappa_short @ cp.pos(-w)
cash_holding_cost = data.kappa_borrow * cp.pos(-c)
holding_cost = asset_holding_cost + cash_holding_cost

spread_cost = data.kappa_spread @ cp.abs(z)
impact_cost = data.kappa_impact @ cp.power(cp.abs(z), 3 / 2)
trading_cost = spread_cost + impact_cost

objective = (
    return_wc
```

```
33      - param.gamma_hold * holding_cost
34      - param.gamma_trade * trading_cost
35  )
36
37  constraints = [
38      cp.sum(w) + c == 1,
39      param.w_min <= w, w <= param.w_max,
40      L <= param.L_tar,
41      param.c_min <= c, c <= param.c_max,
42      param.z_min <= z, z <= param.z_max,
43      T <= param.T_tar,
44      risk_wc <= param.risk_target,
45  ]
46
47  problem = cp.Problem(cp.Maximize(objective), constraints)
48  problem.solve()
```

We start by importing the CVXPY package in line 1 and define the variables of the problem in line 3. The variable `w` is the vector of asset weights, and `c` is the cash weight. We then define the trade vector `z`, turnover `T`, and leverage `L` in lines 5–7 to simplify the notation in the remainder of the code.

In the next block we first define the mean return in lines 10–12, taking into account the factor and idiosyncratic returns, as well as the risk-free rate. We then define the uncertainty in the mean return in line 13, which then reduces the mean return to the worst-case return in line 14.

Similarly, the robust risk is obtained in lines 17–21 by first defining the factor and idiosyncratic risk components, which are combined to the portfolio risk. The uncertainty in the risk, which depends on the asset volatilities, is combined with the portfolio risk to obtain the worst-case risk in line 21. The holding cost is defined in lines 23–25, followed by the trading cost in lines 27–29.

We form the objective function in lines 31–35 by combining the worst-case return with the holding and trading costs, weighted by the corresponding parameters. The constraints are collected in lines 37–45, starting with the budget constraint, followed by the holding and trading constraints, and ending with the risk constraint.

Finally, the problem is defined in line 47, combining the objective and constraints. It is solved in line 48 by simply calling the `.solve()` method on the problem instance, with a suitable solver being chosen automatically.

In only 48 lines of code we have defined and solved the Markowitz problem with all the constraints and objectives described in §4. This underlines the power of using a DSL such as CVXPY to specify convex optimization problems in a way that closely follows the mathematical formulation.

Parameters. Using parameters can provide both a convenient way to specify the problem, as well as a way to reduce the overhead of CVXPY when solving multiple instances. To obtain this speedup requires some restrictions on the problem formulation. For a precise definition we refer the reader to [AAB+19]. Here we only mention that we require expressions to additionally be linear, or affine, in the parameters. For example, we can use CVXPY parameters to easily and quickly change the mean return by writing to the the `.value` attribute of the `mean` and `risk_free` parameters.

```
mean = cp.Parameter(n_assets)
risk_free = cp.Parameter()

mean_return = w @ mean + risk_free * c
```

In some cases, it is necessary to reformulate the problem to satisfy the additional restrictions required to obtain the speedup, *e.g.*, by introducing auxiliary variables. For convenience, we provide a parametrized implementation of the Markowitz problem in the code repository, where these reformulations have already been carried out.

Part III.

Disciplined Saddle Programming

This chapter is a reprint of:

Schiele, P., Luxenberg, E., Boyd, S. (2023). Disciplined Saddle Programming. *Available at https://arxiv.org/abs/2301.13427*. Published in *Transactions on Machine Learning Research*, *available at https://openreview.net/forum?id=KhMLfEIoUm*.

The included version is the second revision of the essay.

Copyright information:

Author contributions:

The idea for this essay emerged during discussions about robust bond portfolio construction methods between Stephen Boyd, Philipp Schiele, and Eric Luxenberg, recognizing the potential for a more general framework. Eric Luxenberg first suggested applying recent work on conic representable saddle functions. Based on this idea, the initial draft was prepared by Philipp Schiele and Eric Luxenberg, who also jointly developed the corresponding software package. Philipp Schiele contributed the insight that solving the negated problem can be used to also obtain the concave coordinates of the saddle point. Stephen Boyd continuously edited and provided feedback on the essay and supervised the project.

Supplementary material available at: https://github.com/cvxgrp/dsp

Disciplined Saddle Programming

Philipp Schiele*[1], Eric Luxenberg*[2], and Stephen Boyd[2]

[1]Department of Statistics, Ludwig-Maximilians-Universität München
[2]Department of Electrical Engineering, Stanford University

January 11, 2024

Abstract

We consider convex-concave saddle point problems, and more generally convex optimization problems we refer to as *saddle problems*, which include the partial supremum or infimum of convex-concave saddle functions. Saddle problems arise in a wide range of applications, including game theory, machine learning, and finance. It is well known that a saddle problem can be reduced to a single convex optimization problem by dualizing either the convex (min) or concave (max) objectives, reducing a min-max problem into a min-min (or max-max) problem. Carrying out this conversion by hand can be tedious and error prone. In this paper we introduce *disciplined saddle programming* (DSP), a domain specific language (DSL) for specifying saddle problems, for which the dualizing trick can be automated. The language and methods are based on recent work by Juditsky and Nemirovski [JN22], who developed the idea of conic-representable saddle point programs, and showed how to carry out the required dualization automatically using conic duality. Juditsky and Nemirovski's conic representation of saddle problems extends Nesterov and Nemirovski's earlier development of conic representable convex problems; DSP can be thought of as extending disciplined convex programming (DCP) to saddle problems. Just as DCP makes it easy for users to formulate and solve complex convex problems, DSP allows users to easily formulate and solve saddle problems. Our method is implemented in an open-source package, also called DSP.

*Equal contribution.

Contents

1 Introduction

We consider saddle problems, by which we mean convex-concave saddle point problems or, more generally, convex optimization problems that include the partial supremum or infimum of convex-concave saddle functions. Saddle problems arise in various fields such as game theory, robust and minimax optimization, machine learning, and finance.

While there are algorithms specifically designed to solve some types of saddle point or minimax problems, another approach is to convert them into standard convex optimization problems using a trick based on duality that can be traced back to at least the 1920s. The idea is to express the infima or suprema that appear in the saddle problem via their duals, which converts them to suprema or infima, respectively. Roughly speaking, this turns a min-max problem into a min-min (or max-max) problem, which can then be solved by standard methods. Specific cases of this trick are well known; the classical example is converting a matrix game, a specific saddle point problem, into a linear program (LP) [MVN53]. While the dualizing trick has been known and used for almost 100 years, it has always been done by hand, for specific problems. It can only be carried out by those who have a working knowledge of duality in convex optimization, and are aware of the trick.

In this paper we propose an automated method for carrying out the dualizing trick. Our method is based on the theory of conic representation of saddle point problems, developed recently by Juditsky and Nemirovski [JN22]. Based on this development, we have designed a domain specific language (DSL) for describing saddle problems, which we refer to as disciplined saddle programming (DSP). When a problem description complies with the syntax rules, *i.e.*, is DSP-compliant, it is easy to verify that it is a valid saddle problem, and more importantly, automatically carry out the dualizing trick. We have implemented the DSL in an open source software package, also called DSP, which works with CVXPY [DB16], a DSL for specifying and solving convex optimization problems. DSP makes it easy to specify and solve saddle problems, without any expertise in (or even knowledge of) convex duality. Even for those with the required expertise to carry out the dualizing trick by hand, DSP is less tedious and error prone.

DSP is *disciplined*, meaning it is based on a small number of syntax rules that, if followed, guarantee that the specified problem is a valid saddle problem. It is analogous to disciplined convex programming (DCP) [GBY06], which is a DSL for specifying convex optimization problems. When a problem specification follows these syntax rules, *i.e.*, is DCP-compliant, it is a valid convex optimization problem, and more importantly can be automatically converted to an equivalent cone program, and then solved. As a practical matter, DCP allows a large number of users to specify and solve even complex convex optimization problems, with no knowledge of the reduction to cone form. Indeed, most DCP users are blissfully unaware of how their problems are solved, *i.e.*, a reduction to cone form. DCP was based on the theory of conic representations of convex functions and problems, pioneered by Nesterov and Nemirovski [NN92]. Widely used implementations of DCP include CVXPY [DB16], Convex.jl [Ude+14], CVXR [FNB20], YALMIP [Lof04], and CVX [GB14]. Like DCP did for convex problems, DSP makes it easy to specify and solve saddle problems, with most users unaware of the dualization trick and reduction used to solve their problems.

1.1 Previous and related work

Saddle problems. Studying saddle problems is a long-standing area of research, resulting in many theoretical insights, numerous algorithms for specific classes of problems, and a large number of applications.

Saddle problems are often studied in the context of minimax or maximin optimization [DM90; DP95], which, while dating back to the 1920s and the work of von Neumann and Morgenstern on game theory [MVN53], continue to be active areas of research, with many recent advancements for example in machine learning [Goo+14]. A variety of methods have been developed for solving saddle point problems, including interior point methods [HT03; Nem99], first-order methods [Kor76; Nem04; Nes07; NO09; CLO13], and second-order methods [NP06; Nes08], where many of these methods are specialized to specific classes of saddle problems. Depending on the class of saddle problem, the methods differ in convergence rate. For example, for the subset of smooth minimax problems, an overview of rates for different curvature assumptions is given in [The+19]. Due to their close relation to Lagrange duality, saddle problems are commonly studied in the context of convex analysis (see, for example, [BV04, §5.4], [Roc70, §33–37], [RW09, §11.J], [BL06, §4.3]), with an analysis via monotone operators given in [RY22].

The practical usefulness of saddle programming in many applications is also increasingly well known. Many applications of saddle programming are robust optimization problems [BBC11; BTEGN09]. For example, in statistics, distributionally robust models can be used when the true distribution of the data generating process is not known [DA19]. Another common area of application is in finance, with [CPT18, §19.3–4] describing a range of financial applications that can be characterized as saddle problems. Similarly, [Boy+17; GI03; LB00] describe variations of the classical portfolio optimization problem as saddle problems.

Disciplined convex programming. DCP is a grammar for constructing optimization problems that are provably convex, meaning that they can be solved globally, efficiently and accurately. It is based on the rule that the convexity of a function f is preserved under composition if all inner expressions in arguments where f is nondecreasing are convex, and all expressions where f is nonincreasing are concave, and all other expressions are affine. A detailed description of the composition rule is given in [BV04, §3.2.4]. Using this rule, functions can be composed from a small set of primitives, called atoms, where each atom has known curvature, sign, and monotonicity. Every function that can be constructed from these atoms according to the composition rule is convex, but the converse is not true. The DCP framework has been implemented in many programming languages, including MATLAB [GB14; Lof04], Python [DB16], R [FNB20], and Julia [Ude+14], and is used by researchers and practitioners in a wide range of fields.

Well-structured convex-concave saddle point problems. As mentioned earlier, disciplined saddle programming is based on Juditsky and Nemirovski's recent work on well-structured convex-concave saddle point problems [JN22].

1.2 Our contributions

We summarize our contributions as follows:

- We introduce disciplined saddle programming, a domain specific language for specifying and solving convex-concave saddle problems. To solve the saddle problems, automated dualization is applied to the conic representation of the problem. We extend the existing literature by deriving a procedure that returns both the convex and concave coordinates of the saddle point. This also guarantees that a valid saddle point was found without the need to check for technical conditions (such as compactness). These developments make the theory of conic representable saddle problems practically applicable for the first time.
- We specify and implement the first DSL that encodes sufficient conditions for conic representability of saddle problems. We develop an open-source Python package, also called DSP, providing a user-friendly interface for specifying and solving saddle problems. Using this implementation, we demonstrate the effectiveness of the framework by solving a variety of saddle problems from different application domains.

1.3 Outline

In §2 we describe saddle programming, which includes the classical saddle point problem, as well as convex problems that include functions described via partial minimization or maximization of a saddle function. We describe some typical applications of saddle programming in §3. In §4 we describe disciplined saddle programming, which is a way to specify saddle programs in such a way that validity is easy to verify, and the reduction to an equivalent cone program can be automated. We describe our implementation in §5, showing how saddle functions, saddle extremum functions, saddle point problems, and saddle problems are specified. We present numerical examples in §6.

2 Saddle programming

2.1 Saddle functions

A *saddle function* (also referred to as a convex-concave saddle function) $f : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$ is one for which $f(\cdot, y)$ is convex for any fixed $y \in \mathcal{Y}$, and $f(x, \cdot)$ is concave for any fixed $x \in \mathcal{X}$. The argument domains $\mathcal{X} \subseteq \mathbf{R}^n$ and $\mathcal{Y} \subseteq \mathbf{R}^m$ must be nonempty closed convex. We refer to x as the convex variable, and y as the concave variable, of the saddle function f.

Examples.

- *Functions of x or y alone.* A convex function of x, or a concave function of y, are trivial examples of saddle functions.

- *Lagrangian of a convex optimization problem.* The convex optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & Ax = b, \quad f_i(x) \leq 0, \quad i = 1, \ldots, m, \end{array}$$

  with variable $x \in \mathbf{R}^n$, where $f_0, \ldots, f_m$ are convex and $A \in \mathbf{R}^{p \times n}$, has Lagrangian

$$L(x, \nu, \lambda) = f(x) + \nu^T(Ax - b) + \lambda_1 f_1(x) + \cdots + \lambda_m f_m(x),$$

  for $\lambda \geq 0$ (elementwise). It is convex in x and affine (and therefore also concave) in $y = (\nu, \lambda)$, so it is a saddle function with

$$\mathcal{X} = \bigcap_{i=0,\ldots,m} \mathbf{dom}\, f_i, \qquad \mathcal{Y} = \mathbf{R}^p \times \mathbf{R}^m_+,$$

- *Bi-affine function.* The function $f(x, y) = (Ax + b)^T(Cy + d)$, with $\mathcal{X} = \mathbf{R}^p$ and $\mathcal{Y} = \mathbf{R}^q$, is evidently a saddle function. The inner product $x^T y$ is a special case of a bi-affine function. For a bi-affine function, either variable can serve as the convex variable, with the other serving as the concave variable.

- *Convex-concave inner product.* The function $f(x, y) = F(x)^T G(y)$, where $F : \mathbf{R}^p \to \mathbf{R}^n$ is a nonnegative elementwise convex function and $G : \mathbf{R}^q \to \mathbf{R}^n$ is a nonnegative elementwise concave function.

- *Weighted ℓ_2 norm.* The function

$$f(x, y) = \left(\sum_{i=1}^n y_i x_i^2 \right)^{1/2},$$

  with $\mathcal{X} = \mathbf{R}^n$ and $\mathcal{Y} = \mathbf{R}^n_+$, is a saddle function.

- *Weighted log-sum-exp.* The function

$$f(x, y) = \log \left(\sum_{i=1}^n y_i \exp x_i \right),$$

  with $\mathcal{X} = \mathbf{R}^n$ and $\mathcal{Y} = \mathbf{R}^n_+$, is a saddle function.

- *Weighted geometric mean.* The function $f(x, y) = \prod_{i=1}^n y_i^{x_i}$, with $\mathcal{X} = \mathbf{R}^n_+$ and $\mathcal{Y} = \mathbf{R}^n_+$, is a saddle function.

- *Quadratic form with quasi-semidefinite matrix.* The function

$$f(x, y) = \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} P & S \\ S^T & Q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

  where the matrix is quasi-semidefinite, *i.e.*, $P \in \mathbf{S}^n_+$ (the set of symmetric positive semidefinite matrices) and $-Q \in \mathbf{S}^n_+$.

- *Quadratic form.* The function $f(x, Y) = x^T Y x$, with $\mathcal{X} = \mathbf{R}^n$ and $\mathcal{Y} = \mathbf{S}^n_+$ (the set of symmetric positive semidefinite $n \times n$ matrices), is a saddle function.
- As a more esoteric example, the function $f(x, Y) = x^T Y^{1/2} x$, with $\mathcal{X} = \mathbf{R}^n$ and $\mathcal{Y} = \mathbf{S}^n_+$, is a saddle function.

Combination rules. Saddle functions can be combined in several ways to yield saddle functions. For example the sum of two saddle functions is a saddle function, provided the domains have nonempty intersection. A saddle function scaled by a nonnegative scalar is a saddle function. Scaling a saddle function with a nonpositive scalar, and swapping its arguments, yields a saddle function: $g(x, y) = -f(y, x)$ is a saddle function provided f is. Saddle functions are preserved by pre-composition of the convex and concave variables with an affine function, *i.e.*, if f is a saddle function, so is $f(Ax + b, Cx + d)$. Indeed, the bi-affine function is just the inner product with an affine pre-composition for each of the convex and concave variables.

2.2 Saddle point problems

A *saddle point* $(x^\star, y^\star) \in \mathcal{X} \times \mathcal{Y}$ is any point that satisfies

$$f(x^\star, y) \leq f(x^\star, y^\star) \leq f(x, y^\star) \text{ for all } x \in \mathcal{X},\ y \in \mathcal{Y}. \tag{1}$$

In other words, $x^\star$ minimizes $f(x, y^\star)$ over $x \in \mathcal{X}$, and $y^\star$ maximizes $f(x^\star, y)$ over $y \in \mathcal{Y}$. The basic *saddle point problem* is to find such a saddle point,

$$\text{find } x^\star,\ y^\star \text{ which satisfy (1).} \tag{2}$$

The value of the saddle point problem is $f(x^\star, y^\star)$.

Existence of a saddle point for a saddle function is guaranteed, provided some technical conditions hold. For example, Sion's theorem [Sio58] guarantees the existence of a saddle point when $\mathcal{Y}$ is compact. There are many other cases.

Examples.

- *Matrix game.* In a matrix game, player one chooses $i \in \{1, \ldots, m\}$, and player two chooses $j \in \{1, \ldots, n\}$, resulting in player one paying player two the amount C_{ij}. Player one wants to minimize this payment, while player two wishes to maximize it. In a mixed strategy, player one makes choices at random, from probabilities given by x and player two makes independent choices with probabilities given by y. The expected payment from player one to player two is then $f(x, y) = x^T C y$. With $\mathcal{X} = \{x \mid x \geq 0,\ \mathbf{1}^T x = 1\}$, and similarly for $\mathcal{Y}$, a saddle point corresponds to an equilibrium, where no player can improve her position by changing (mixed) strategy. The saddle point problem consists of finding a stable equilibrium, *i.e.*, an optimal mixed strategy for each player.
- *Lagrangian.* A saddle point of a Lagrangian of a convex optimization problem is a primal-dual optimal pair for the convex optimization problem.

2.3 Saddle extremum functions

Suppose f is a saddle function. The function $G : \mathcal{X} \to \mathbf{R} \cup \{\infty\}$ defined by

$$G(x) = \sup_{y \in \mathcal{Y}} f(x, y), \quad x \in \mathcal{X}, \tag{3}$$

is called a *saddle max function*. Similarly, the function $H : \mathcal{Y} \to \mathbf{R} \cup \{-\infty\}$ defined by

$$H(x) = \inf_{x \in \mathcal{X}} f(x, y), \quad y \in \mathcal{Y}, \tag{4}$$

is called a *saddle min function*. Saddle max functions are convex, and saddle min functions are concave. We will use the term *saddle extremum* (SE) functions to refer to saddle max or saddle min functions. Which is meant is clear from context, *i.e.*, whether it is defined by minimization (infimum) or maximization (supremum), or its curvature (convex or concave). Note that in SE functions, we always maximize (or take supremum) over the concave variable, and minimize (or take infimum) over the convex variable. This means that evaluating $G(x)$ or $H(y)$ involves solving a convex optimization problem.

Examples.

- *Dual function.* Minimizing a Lagrangian $L(x, \nu, \lambda)$ over x gives the dual function of the original convex optimization problem.
- Maximizing a Lagrangian $L(x, \nu, \lambda)$ over $y = (\nu, \lambda)$ gives the objective function restricted to the feasible set.
- *Conjugate of a convex function.* Suppose f is convex. Then $g(x, y) = f(x) - x^T y$ is a saddle function, the Lagrangian of the problem of minimizing f subject to $x = 0$. Its saddle min is the negative conjugate function: $\inf_x g(x, y) = -f^*(y)$.
- *Sum of k largest entries.* Consider $f(x, y) = x^T y$, with $\mathcal{Y} = \{y \mid 0 \leq y \leq 1,\ \mathbf{1}^T y = k\}$. The associated saddle max function G is the sum of the k largest entries of x.

Saddle points via SE functions. A pair $(x^\star, y^\star)$ is a saddle point of a saddle function f if and only if $x^\star$ minimizes the convex SE function G in (3) over $x \in \mathcal{X}$, and $y^\star$ maximizes the concave SE function H defined in (4) over $y \in \mathcal{Y}$. This means that we can find saddle points, *i.e.*, solve the saddle point problem (2), by solving the convex optimization problem

$$\begin{array}{ll} \text{minimize} & G(x) \\ \text{subject to} & x \in \mathcal{X}, \end{array} \tag{5}$$

with variable x, and the convex optimization problem

$$\begin{array}{ll} \text{maximize} & H(y) \\ \text{subject to} & y \in \mathcal{Y}, \end{array} \tag{6}$$

with variable y. The problem (5) is called a minimax problem, since we are minimizing a function defined as the maximum over another variable. The problem (6) is called a maximin problem.

While the minimax problem (5) and maximin problem (6) are convex, they cannot be directly solved by conventional methods, since the objectives themselves are defined by maximization and minimization, respectively. There are solution methods specifically designed for minimax and maximin problems [LJJ20; MB09], but as we will see minimax problems involving SE functions can be transformed to equivalent forms that can be directly solved using conventional methods.

2.4 Saddle problems

In this paper we consider convex optimization problems that include SE functions in the objective or constraints, which we refer to as *saddle problems*. The convex problems that solve the basic saddle point problem (5) and (6) are special cases, where the objective is an SE function. As another example consider the problem of minimizing a convex function ϕ subject to the convex SE constraint $H(y) \leq 0$, which can be expressed as

$$\begin{array}{ll} \text{minimize} & \phi(x) \\ \text{subject to} & f(x,y) \leq 0 \text{ for all } y \in \mathcal{Y}, \end{array} \tag{7}$$

with variable x. The constraint here is called a *semi-infinite constraint*, since (when $\mathcal{Y}$ is not a singleton) it can be thought of as an infinite collection of convex constraints, one for each $y \in \mathcal{Y}$ [HK93].

Saddle problems include the minimax and maximin problems (that can be used to solve the saddle point problem), and semi-infinite problems that involve SE functions. There are many other examples of saddle problems, where SE functions can appear in expressions that define the objective and constraints.

Robust cost LP. As a more specific example of a saddle problem consider the linear program with robust cost,

$$\begin{array}{ll} \text{minimize} & \sup_{c \in \mathcal{C}} c^T x \\ \text{subject to} & Ax = b, \quad x \geq 0, \end{array} \tag{8}$$

with variable $x \in \mathbf{R}^n$, with $\mathcal{C} = \{c \mid Fc \leq g\}$. This is an LP with worst case cost over the polyhedron $\mathcal{C}$ [BBC11; BTEGN09]. This is a saddle problem with convex variable x, concave variable y, and an objective which is a saddle max function.

2.5 Solving saddle problems

Special cases with tractable analytical expressions. There are cases where an SE function can be worked out analytically. An example is the max of a linear function over a box,

$$\sup_{l \leq y \leq u} y^T x = (1/2)(u+l)^T x + (1/2)(u-l)^T |x|,$$

where the absolute value is elementwise. We will see other cases in our examples.

Subgradient methods. We can readily compute a subgradient of a saddle max function (or a supergradient of a saddle min function) at a given input, by simply maximizing over the concave variable (minimizing over the convex variable), which is itself a convex optimization problem, and then obtaining a subgradient (supergradient) at that maximizer (minimizer). We can then use any method to solve the saddle problem using these subgradients, *e.g.*, subgradient-type methods, ellipsoid method, or localization methods such as the analytic center cutting plane method. In [MB09] such an approach is used for general minimax problems.

Methods for specific forms. Many methods have been developed for finding saddle points of saddle functions with the special form

$$f(x, y) = x^T K y + \phi(x) + \psi(y),$$

where ϕ is convex, ψ is concave, and K is a matrix [BS15; Con13; CP11; Nes05a; Nes05b; CP16]. Beyond this example, there are many other special forms of saddle functions, with different methods adapted to properties such as smoothness, separability, and strong-convex-strong-concavity.

2.6 Dual reduction

A well-known trick can be used to transform a saddle point problem into an equivalent problem that does not contain SE functions. This method of transforming an inner minimization is not new; it has been used since the 1950s when Von Neumann proved the minimax theorem using strong duality in his work with Morgenstern on game theory [MVN53]. Using this observation, he showed that the minimax problem of a two player game is equivalent to an LP. Duality allows us to express the convex (concave) SE function as an infimum (supremum), which facilitates the use of standard convex optimization. We think of this as a reduction to an equivalent problem that removes the SE functions from the objective and constraints.

Robust cost LP. We illustrate the dualization method for the robust cost LP (8). The key is to express the robust cost or saddle max function $\sup_{Fc \leq g} c^T x$ as an infimum. We first observe that this saddle max function is the optimal value of the LP

$$\begin{array}{ll} \text{maximize} & x^T c \\ \text{subject to} & Fc \leq g, \end{array}$$

with variable c. Its dual is

$$\begin{array}{ll} \text{minimize} & g^T \lambda \\ \text{subject to} & F^T \lambda = x, \quad \lambda \geq 0, \end{array}$$

with variable λ. With $\mathcal{C} = \{c \mid Fc \leq g\}$, and assuming $\mathcal{C}$ is nonempty, this dual problem has the same optimal value as the primal, *i.e.*,

$$\sup_{c \in \mathcal{C}} c^T x = \inf_{\lambda \geq 0,\ F^T\lambda = x} g^T \lambda$$

Substituting this into (8) we obtain the problem

$$\begin{array}{ll} \text{minimize} & g^T\lambda \\ \text{subject to} & Ax = b, \quad x \geq 0, \quad F^T\lambda = x, \quad \lambda \geq 0, \end{array} \tag{9}$$

with variables x and λ. This simple LP is equivalent to the original robust LP (8), in the sense that if $(x^\star, \lambda^\star)$ is a solution of (9), then $x^\star$ is a solution of the robust LP (8).

We will see this dualization trick in a far more general setting in §4.

3 Applications

In this section we describe a few applications of saddle programming.

3.1 Robust bond portfolio construction

We describe here a simplified version of the problem described in much more detail in [LSB22]. Our goal is to construct a portfolio of n bonds, giving by its holdings vector $h \in \mathbf{R}^n_+$, where h_i is the number of bond i held in the portfolio. Each bond produces a cash flow, *i.e.*, a sequence of payments to the portfolio holder, up to some period T. Let $c_{i,t}$ be the payment from bond i in time period t. Let $y \in \mathbf{R}^T$ be the yield curve, which gives the time value of cash: A payment of one dollar at time t is worth $\exp(-ty_t)$ current dollars, assuming continuously compounded returns. The bond portfolio value, which is the present value of the total cash flow, can be expressed as

$$V(h, y) = \sum_{i=1}^{n} \sum_{t=1}^{T} h_i c_{i,t} \exp(-ty_t).$$

This function is convex in the yields y and concave (in fact, linear) in the holdings vector h.

Now suppose we do not know the yield curve, but instead have a convex set $\mathcal{Y}$ of possible values, with $y \in \mathcal{Y}$. The worst case value of the bond portfolio, over this set of possible yield curves, is

$$V^{\text{wc}}(h) = \inf_{y \in \mathcal{Y}} V(h, y).$$

We recognize this as a saddle min function. (In this application, y is the convex variable of the saddle function V, whereas elsewhere in this paper we use y to denote the concave variable.)

We consider a robust bond portfolio construction problem of the form

$$\begin{array}{ll} \text{minimize} & \phi(h) \\ \text{subject to} & h \in \mathcal{H}, \quad V^{\text{wc}}(h) \geq V^{\text{lim}}, \end{array} \tag{10}$$

where ϕ is a convex objective, typically a measure of return and risk, $\mathcal{H}$ is a convex set of portfolio constraints (for example, imposing $h \geq 0$ and a total budget), and V^{lim} is a specified limit on worst case value of the portfolio over the yield curve set $\mathcal{Y}$, which has a saddle min as a constraint.

For some simple choices of $\mathcal{Y}$ the worst case value can be found analytically. One example is when $\mathcal{Y}$ has a maximum element. In this special case, the maximum element is the minimizer of the value over $\mathcal{Y}$ (since V is a monotone decreasing function of y). For other cases, however, we need to solve the saddle problem (10).

3.2 Model fitting robust to data weights

We wish to fit a model parametrized by $\theta \in \Theta \subseteq \mathbf{R}^n$ to m observed data points. We do this by minimizing a weighted loss over the observed data, plus a regularizer,

$$\sum_{i=1}^{m} w_i \ell_i(\theta) + r(\theta),$$

where ℓ_i is the convex loss function for observed data point i, r is a convex regularizer function, and the weights w_i are nonnegative. The weights can be used to adjust a data sample that was not representative, as in [BAB21], or to ignore some of the data points (by taking $w_i = 0$), as in [BGM20]. Evidently the weighted loss is a saddle function, with convex variable θ and concave variable w.

We consider the case when the weights are unknown, but lie in a convex set, $w \in \mathcal{W}$. The robust fitting problem is to choose θ to minimize the worst case loss over the set of possible weights, plus the regularizer,

$$\max_{w \in \mathcal{W}} \sum_{i=1}^{m} w_i \ell_i(\theta) + r(\theta).$$

We recognize the first term, *i.e.*, the worst case loss over the set of possible weights, as a saddle max function.

For some simple choices of $\mathcal{W}$ the worst case loss can be expressed analytically. For example with

$$\mathcal{W} = \{w \mid 0 \leq w \leq 1,\ \mathbf{1}^T w = k\},$$

(with $k \in [0, n]$), the worst case loss is given by

$$\max_{w \in \mathcal{W}} \sum_{i=1}^{m} w_i \ell_i(\theta) = \phi(\ell_1, \ldots, \ell_m),$$

where ϕ is the sum-of-k-largest entries [BV04, §3.2.3]. (Our choice of symbol k suggests that k is an integer, but it need not be.) In this case we judge the model parameter θ by its worst loss on any subset of k of data points. Put another way, we judge θ by dropping the $m - k$ data points on which it does best (*i.e.*, has the smallest loss) [BGM20].

CVXPY directly supports the sum-of-k-largest function, so the robust fitting problem can be formulated and solved without using DSP. To support this function, CVXPY carries out a transformation very similar to the one that DSP does. The difference is that the transformation in CVXPY is specific to this one function, whereas the one carried out in DSP is general, and would work for other convex weight sets. One such case would be to constrain the Wasserstein distance of the weights to a nominal distribution.

3.3 Robust production problem with worst case prices

We consider the choice of a vector of quantities $q \in \mathcal{Q} \subseteq \mathbf{R}^n$. Positive entries indicate goods we buy, and negative quantities are goods we sell. The set of possible quantities $\mathcal{Q}$ is our production set, which is convex. In addition, we have a manufacturing cost associated with the choice q, given by $\phi(q)$, where ϕ is a convex function. The total cost is the manufacturing cost plus the cost of goods (which includes revenue), $\phi(q) + p^T q$, where $p \in \mathbf{R}^n$ is vector of prices.

We consider the situation when we do not know the prices, but we have a convex set they lie in, $p \in \mathcal{P}$. The worst case cost of the goods is $\max_{p\in\mathcal{P}} p^T q$. The robust production problem is

$$\begin{array}{ll} \text{minimize} & \phi(q) + \max_{p\in\mathcal{P}} p^T q \\ \text{subject to} & q \in \mathcal{Q}, \end{array} \tag{11}$$

with variable q. Here too we can work out analytical expressions for simple choices of $\mathcal{P}$, such as a range for each component, in which case the worst case price is the upper limit for goods we buy, and the lower limit for goods we sell. In other cases, we solve the saddle problem (11).

3.4 Robust Markowitz portfolio construction

Markowitz portfolio construction [Mar52] chooses a set of weights (the fraction of the total portfolio value held in each asset) by solving the convex problem

$$\begin{array}{ll} \text{maximize} & \mu^T w - \gamma w^T \Sigma w \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array}$$

where the variable is the vector of portfolio weights $w \in \mathbf{R}^n$, $\mu \in \mathbf{R}^n$ is a forecast of the asset returns, $\gamma > 0$ is the risk aversion parameter, $\Sigma \in \mathbf{S}^n_{++}$ is a forecast of the asset return covariance matrix, and $\mathcal{W}$ is a convex set of feasible portfolios. The objective is called the risk adjusted (mean) return.

Markowitz portfolio construction is known to be fairly sensitive to the (forecasts) μ and Σ, which have to be chosen with some care; see, *e.g.*, [BL91]. One approach is to specify a convex uncertainty set $\mathcal{U}$ that (μ, Σ) must lie in, and replace the objective with its worst case (smallest) value over this uncertainty set. This gives the robust Markowitz portfolio construction problem

$$\begin{array}{ll} \text{maximize} & \inf_{(\mu,\Sigma)\in\mathcal{U}} \left(\mu^T w - \gamma w^T \Sigma w\right) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array}$$

with variable w. This is described in, *e.g.*, [Boy+17; GI03; LB00]. We observe that this is directly a saddle problem, with a saddle min objective, *i.e.*, a maximin problem.

For some simple versions of the problem we can work out the saddle min function explicitly. One example, given in [Boy+17], uses $\mathcal{U} = \mathcal{M} \times \mathcal{S}$, where

$$\begin{aligned} \mathcal{M} &= \{\mu + \delta \mid |\delta_i| \leq \rho_i,\ i = 1, \ldots, n\}, \\ \mathcal{S} &= \{\Sigma + \Delta \mid \Sigma + \Delta \succeq 0,\ |\Delta_{ij}| \leq \eta(\Sigma_{ii}\Sigma_{jj})^{1/2},\ i, j = 1, \ldots, n\}, \end{aligned}$$

where $\rho > 0$ is a vector of uncertainties in the forecast returns, and $\eta \in (0, 1)$ is a parameter that scales the perturbation to the forecast covariance matrix. (We interpret δ and Δ as perturbations of the nominal mean and covariance μ and Σ, respectively.) We can express the worst case risk adjusted return analytically as

$$\inf_{(\mu,\Sigma)\in\mathcal{U}} \left(\mu^T w - \gamma w^T \Sigma w\right) = \mu^T w - \gamma w^T \Sigma w - \rho^T |w| - \gamma\eta \left(\sum_{i=1}^{n} \Sigma_{ii}^{1/2} |w_i|\right)^2.$$

The first two terms are the nominal risk adjusted return; the last two terms (which are nonpositive) represent the cost of uncertainty.

4 Disciplined saddle programming

4.1 Saddle function calculus

We use the notation $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \subseteq \mathbf{R}^{n\times m} \to \mathbf{R}$ to denote a saddle function with concave variables x and convex variables y. The set of operations that, when performed on saddle functions, preserves the saddle property are called the *saddle function calculus*. The calculus is quite simple, and consists of the following operations:

1. *Conic combination of saddle functions.* Let $\phi_i(x_i, y_i)$, $i = 1, \ldots, k$ be saddle functions. Let $\theta_i \geq 0$ for each i. Then the conic combination, $\phi(x, y) = \sum_{i=1}^{k} \theta_i \phi_i(x_i, y_i)$, is a saddle function.

2. *Affine precomposition of saddle functions.* Let $\phi(x, y)$ be a saddle function, with $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$. Let $A \in \mathbf{R}^{n\times q}$, $b \in \mathbf{R}^n$, $C \in \mathbf{R}^{m\times p}$, and $d \in \mathbf{R}^m$. Then, with $u \in \mathbf{R}^q$ and $v \in \mathbf{R}^p$, the affine precomposition, $\phi(Au + b, Cv + d)$, is a saddle function.

3. *Precomposition of saddle functions.* Let $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \subseteq \mathbf{R}^{n\times m} \to \mathbf{R}$ be a saddle function, with $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$. The precomposition with a function $f : \mathbf{R}^p \to \mathbf{R}^n$, $\phi(f(u), y)$, is a saddle function if for each $i = 1, \ldots, n$ one of the following holds:

   - $f_i(u)$ is convex and ϕ is nondecreasing in x_i for all $y \in \mathcal{Y}$ and all $x \in \mathcal{X}$.
   - $f_i(u)$ is concave and ϕ is nonincreasing in x_i for all $y \in \mathcal{Y}$ and all $x \in \mathcal{X}$.

   Similarly, the precomposition with a function $g : \mathbf{R}^q \to \mathbf{R}^m$, $\phi(x, g(v))$, is a saddle function if for each $j = 1, \ldots, m$ one of the following holds:

- $g_j(v)$ is convex and ϕ is nonincreasing in y_j for all $x \in \mathcal{X}$ and all $y \in \mathcal{Y}$.
- $g_j(v)$ is concave and ϕ is nondecreasing in y_j for all $x \in \mathcal{X}$ and all $y \in \mathcal{Y}$.

4.2 Conic representable saddle functions

Nemirovski and Juditsky propose a class of *conic representable* saddle functions which facilitate the automated dualization of saddle problems [JN22]. We will first introduce some terminology and notation, and then describe the class of conic representable saddle functions.

Notation. We use the notation $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \subseteq \mathbf{R}^{n \times m} \to \mathbf{R}$ to denote a saddle function which is convex in x and concave in y. Let K_x, K_y and K be members of a collection $\mathcal{K}$ of closed, convex, and pointed cones with nonempty interiors in Euclidean spaces such that $\mathcal{K}$ contains a nonnegative ray, is closed with respect to taking finite direct products of its members, and is closed with respect to passing from a cone to its dual. We denote conic membership $z \in K$ by $z \succeq_K 0$. We call a set $\mathcal{X} \subseteq \mathbf{R}^n$ $\mathcal{K}$-representable if there exist constant matrices A and B, a constant vector c, and a cone $K \in \mathcal{K}$ such that

$$\mathcal{X} = \{x \mid \exists u : Ax + Bu \preceq_K c\}.$$

CVXPY [DB16] can implement a function f exactly when its epigraph $\{(x, u) \mid f(x) \leq u\}$ is $\mathcal{K}$-representable.

Conic representable saddle functions. Let $\mathcal{X}$ and $\mathcal{Y}$ be nonempty and possessing $\mathcal{K}$-representations

$$\mathcal{X} = \{x \mid \exists u : Ax + Bu \preceq_K c\}, \quad \mathcal{Y} = \{y \mid \exists v : Cy + Dv \preceq_K e\}.$$

A saddle function $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$ is $\mathcal{K}$-representable if there exist constant matrices P, Q, R, constant vectors p and s and a cone $K \in \mathcal{K}$ such that for each $x \in \mathcal{X}$ and $y \in \mathcal{Y}$,

$$\phi(x, y) = \inf_{f,t,u} \{f^T y + t \mid Pf + tp + Qu + Rx \preceq_K s\}.$$

Here f is a vector of the same dimension as y, t is a scalar, and u is a vector. This definition generalizes simple class of bilinear saddle functions. See [JN22] for much more detail.

Automated dualization. Suppose we have a $\mathcal{K}$-representable saddle function ϕ as above. The conic form allows us to derive a dualized representation of the saddle extremum function

$$\Phi(x) = \sup_{y \in \mathcal{Y}} \phi(x, y)$$

which again admits a tractable conic form, meaning that it can be represented in a DSL like CVXPY. Specifically,

$$\begin{aligned}
\Phi(x) &= \sup_{y\in\mathcal{Y}} \phi(x,y) \\
&= \sup_{y\in\mathcal{Y}} \inf_{f,t,u} \left\{ f^T y + t \;\middle|\; Pf + tp + Qu + Rx \preceq_K s \right\} \\
&= \inf_{f,t,u} \left\{ \sup_{y\in\mathcal{Y}} \left(f^T y + t \right) \;\middle|\; Pf + tp + Qu + Rx \preceq_K s \right\} \qquad (12) \\
&= \inf_{f,t,u} \left\{ \sup_{y\in\mathcal{Y}} \left(f^T y \right) + t \;\middle|\; Pf + tp + Qu + Rx \preceq_K s \right\} \\
&= \inf_{f,t,u} \left\{ \inf_{\lambda} \left\{ \lambda^T e \;\middle|\; \begin{array}{l} C^T\lambda = f,\ D^T\lambda = 0 \\ \lambda \succeq_{K^*} 0 \end{array} \right\} + t \;\middle|\; Pf + tp + Qu + Rx \preceq_K s \right\} \qquad (13)
\end{aligned}$$

where in (12) we use Sion's minimax theorem [Sio58] to reverse the inf and sup, and in (13) we invoke strong duality to replace the supremum over y with an infimum over λ. Concretely, strong duality and the conic structure allow us to equate

$$\sup_y \left\{ f^T y \;\middle|\; Cy + Dv \preceq_K e \right\} = \inf_{\lambda} \left\{ \lambda^T e \;\middle|\; C^T\lambda = f,\ D^T\lambda = 0,\ \lambda \succeq_{K^*} 0 \right\},$$

where K^* is the dual cone of K. This is exactly the automated dualization made possible by the conic representable form of ϕ (which DSP provides). Given the conic representation of ϕ, the dualized form is obtained via the explicit formula given in (13).

The final line implies a conic representation of the epigraph of $\Phi(x)$,

$$\{(x,u) \mid \Phi(x) \le u\} = \left\{ (x,u) \;\middle|\; \exists \lambda, f, t, u : \begin{array}{l} \lambda^T e + t \le u \\ C^T\lambda = f,\ D^T\lambda = 0,\ \lambda \succeq_{K^*} 0 \\ Pf + tp + Qu + Rx \preceq_K s \end{array} \right\},$$

which is tractable and can be implemented in a DSL like CVXPY. This transformation is exact, and so there is no notion of approximation error or optimality gap arising from the dualization procedure.

A mathematical nuance. Switching the inf and sup in (12) requires Sion's theorem to hold. A sufficient condition for Sion's theorem to hold is that the set $\mathcal{Y}$ is compact. However, the min and max can be exchanged even if $\mathcal{Y}$ is not compact. Then, due to the max-min inequality

$$\max_{y\in\mathcal{Y}} \min_{x\in\mathcal{X}} f(x,y) \le \min_{x\in\mathcal{X}} \max_{y\in\mathcal{Y}} f(x,y),$$

the equality in (13) is replaced with a less than or equal to, and we obtain a convex restriction. Thus, if a user creates a problem involving an SE function (as opposed to a saddle point problem only containing saddle functions in the objective), then DSP guarantees that the problem generated is a restriction. This means that the variables returned are feasible and the returned optimal value is an upper bound on the optimal value for the user's problem.

Obtaining convex and concave saddle point coordinates. One challenge that arises in transforming the mathematical concept of conic representable saddle functions into a practical implementation is that the automated dualization removes the concave variable from the problem. Additionally, the procedure relies on the technical conditions such as compactness, which we believe should not be exposed in a user interface. We now address these points.

In our implementation, a saddle problem with an SE function in the objective is solved by applying the above automatic dualization to both the objective ϕ and $-\phi$ and then solving each resulting convex problem. Note that $\phi(x, y)$ is convex in x and concave in y, while $-\phi(x, y)$ is concave in x and convex in y. We do so in order to obtain both the convex and concave components of the saddle point, since the dualization removes the concave variable. To see this, note that (13) contains x but not y (and the opposite holds for the negated problem). The saddle problem is only reported as solved if the optimal value of the problem with objective ϕ, u, is within a numerical tolerance of the negated optimal value of the problem with objective $-\phi$, $-l$. If this holds, this actually implies that

$$\max_{y\in\mathcal{Y}} \min_{x\in\mathcal{X}} \phi(x, y) = \min_{x\in\mathcal{X}} \max_{y\in\mathcal{Y}} \phi(x, y),$$

i.e., (12) was valid, even if for example $\mathcal{Y}$ is not compact. To see this, note that solving for ϕ as well as $-\phi$ results in an upper and a lower bound on the optimal value of the saddle point problem,

$$\max_{y\in\mathcal{Y}} \min_{x\in\mathcal{X}} \phi(x, y) \leq \min_{x\in\mathcal{X}} \max_{y\in\mathcal{Y}} \phi(x, y) \leq u, \quad \text{and} \quad \max_{x\in\mathcal{X}} \min_{y\in\mathcal{Y}} -\phi(x, y) \leq \min_{y\in\mathcal{Y}} \max_{x\in\mathcal{X}} -\phi(x, y) \leq -l.$$

Using symmetry and combining the above inequalities, we obtain

$$l \leq \max_{y\in\mathcal{Y}} \min_{x\in\mathcal{X}} \phi(x, y) \leq \min_{x\in\mathcal{X}} \max_{y\in\mathcal{Y}} \phi(x, y) \leq u.$$

Suppose now that $l = u$. Note that since

$$\phi(x^\star, y^\star) \leq \max_{y\in\mathcal{Y}} \phi(x^\star, y) = u, \quad \text{and} \quad \phi(x^\star, y^\star) \geq \min_{x\in\mathcal{X}} \phi(x, y^\star) = l,$$

we have that $\phi(x^\star, y^\star) = l = u$. That is, the pair $(x^\star, y^\star)$ obtains the optimal value of the saddle point problem. All that remains is to verify that this pair satisfies the saddle point property. We have that $\phi(x^\star, y) \leq \phi(x^\star, y^\star)$ for all $y \in \mathcal{Y}$, since otherwise $u = \phi(x^\star, y^\star) < \max_{y\in\mathcal{Y}} \phi(x^\star, y) = u$, a contradiction. Similarly, $\phi(x, y^\star) \geq \phi(x^\star, y^\star)$ for all $x \in \mathcal{X}$. Taken together, these inequalities state that $(x^\star, y^\star)$ is a saddle point, since

$$\phi(x^\star, y) \leq \phi(x^\star, y^\star) \leq \phi(x, y^\star), \quad \forall x \in \mathcal{X}, y \in \mathcal{Y}.$$

Thus, a user need not concern themselves with the compactness of $\mathcal{Y}$ (or any other sufficient condition for Sion's theorem) when using DSP to find a saddle point; if a saddle point problem is solved, then the saddle point property is guaranteed to hold. This mathematical insight extends the work of [JN22], which assumes compactness of $\mathcal{Y}$, allowing users who might be unfamiliar with this technical restriction to use DSP.

5 Implementation

In this section we describe our Python implementation of the concepts and methods described in §4, which we also call DSP. It can be accessed online under an open source license at `https://github.com/cvxgrp/dsp`. DSP works with CVXPY [DB16], an implementation of a DSL for convex optimization based on DCP. We use the term DSP in two different ways. We use it to refer to the mathematical concept of disciplined saddle programming, and also our specific implementation; which is meant should be clear from the context. The term DSP-compliant refers to a function or expression that is constructed according to the DSP composition rules given in §5.2. It can also refer to a problem that is constructed according to these rules. In the code snippets below, we use the prefix `cp` to indicate functions and classes from CVXPY. (We give functions and classes from DSP without prefix, whereas they would likely have a prefix such as `dsp` in real code.)

5.1 Atoms

Saddle functions in DSP are created from fundamental building blocks or atoms. These building blocks extend the atoms from CVXPY [DB16]. In CVXPY, atoms are either jointly convex or concave in all their variables, but in DSP, atoms are (jointly) convex in a subset of the variables and (jointly) concave in the remaining variables. We describe some DSP atoms below. The listing is not exhaustive, and additional atoms can be added as necessary.

Inner product. The atom `inner(x,y)` represents the inner product $x^T y$. Since either x or y could represent the convex variable, we adopt the convention in DSP that the first argument of `inner` is the convex variable. According to the DSP rules, both arguments to `inner` must be affine, and the variables they depend on must be disjoint.

Saddle inner product. The atom `saddle_inner(F, G)` corresponds to the function $F(x)^T G(y)$, where F and G are vectors of nonnegative and respectively elementwise convex and concave functions. It is DSP-compliant if F is DCP convex and nonnegative and G is DCP concave. If the function G is not DCP nonnegative, then the DCP constraint `G >= 0` is attached to the expression. This is analogous to how the DCP constraint `x >= 0` is added to the expression `cp.log(x)`. As an example consider

```
f = saddle_inner(cp.square(x), cp.log(y)).
```

This represents the saddle function

$$f(x, y) = x^2 \log y - I(y \geq 1),$$

where I is the $\{0, \infty\}$ indicator function of its argument.

Weighted ℓ_2 norm. The `weighted_norm2(x, y)` atom represents the saddle function $(\sum_{i=1}^n y_i x_i^2)^{1/2}$, with $y \geq 0$. It is DSP-compliant if `x` is either DCP affine or both convex and nonnegative, and y is DCP concave. Here too, the constraint `y >= 0` is added if y is not DCP nonnegative.

Weighted log-sum-exp. The `weighted_log_sum_exp(x, y)` atom represents the saddle function $\log(\sum_{i=1}^n y_i \exp x_i)$, with $y \geq 0$. It is DSP-compliant if `x` is DCP convex, and y is DCP concave. The constraint `y >= 0` is added if y is not DCP nonnegative.

Quasi-semidefinite quadratic form. The `quasidef_quad_form(x, y, P, Q, S)` atom represents the function

$$f(x,y) = \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} P & S \\ S^T & Q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

where the matrix is quasi-semidefinite, *i.e.*, $P \in \mathbf{S}_+^n$ and $-Q \in \mathbf{S}_+^n$. It is DSP-compliant if x is DCP affine and y is DCP affine.

Quadratic form. The `saddle_quad_form(x, Y)` atom represents the function $x^T Y x$, where Y is a PSD matrix. It is DSP-compliant if x is DCP affine, and Y is DCP PSD.

5.2 Calculus rules

The atoms can be combined according to the calculus described below to form expressions that are DSP-compliant. For example, saddle functions can be added or scaled. DCP-compliant convex and concave expressions are promoted to saddle functions with no concave or convex variables, respectively. For example, with variables `x`, `y`, and `z`, the expression

```
f = 2.5 * saddle_inner(cp.square(x), cp.log(y)) + cp.minimum(y,1) - z
```

is DSP-compliant, with convex variable `x`, concave variable `y`, and affine variable `z`.

Calling the `is_dsp` method of an expression returns `True` if the expression is DSP-compliant. The methods `convex_variables`, `concave_variables`, and `affine_variables`, list the convex, concave, and affine variables, respectively. The convex variables are those that could only be convex, and similarly for concave variables. We refer to the convex variables as the unambiguously convex variables, and similarly for the concave variables. The three lists of variables gives a partition of all the variables the expression depends on. For the expression above, `f.is_dsp()` evaluates as `True`, `f.convex_variables()` returns the list `[x]`, `f.concave_variables()` returns the list `[y]`, and `f.affine_variables()` returns the list `[z]`. Note that the role of `z` is ambiguous in the expression, since it could be either a convex or concave variable.

No mixing variables rule. The DSP rules prohibit mixing of convex and concave variables. For example if we add two saddle expressions, no variable can appear in both its convex and concave variable lists.

DSP-compliance is sufficient but not necessary to be a saddle function. Recall that if an expression is DCP convex (concave), then it is convex (concave), but the converse is false. For example, the expression `cp.sqrt(1 + cp.square(x))` represents the convex function $\sqrt{1+x^2}$, but is not DCP. But we can express the same function as `cp.norm2(cp.hstack([1, x]))`, which is DCP. The same holds for DSP and saddle function: If an expression is DSP-compliant, then it represents a saddle function; but it can represent a saddle function and not be DSP-compliant. As with DCP, such an expression would need to be rewritten in DSP-compliant form, to use any of the other features of DSP (such as a solution method). As an example, the expression `x.T @ C @ y` represents the saddle function x^TCy, but is not DSP-compliant. The same function can be expressed as `inner(x, C @ y)`, which is DSP-compliant. While this restrictive syntax is an inherent limitation of disciplined convex programming in general, it is required for any parser based on the DSP composition rules.

When there are affine variables in a DSP-compliant expression, it means that those variables could be considered either convex or concave; either way, the function is a saddle function.

Example. The code below defines the bi-linear saddle function $f(x, y) = x^TCy$, the objective of a matrix game, with x the convex variable and y the concave variable.

Creating a saddle function.

```
1  from dsp import *  # notational convenience
2  import cvxpy as cp
3  import numpy as np
4
5  x = cp.Variable(2)
6  y = cp.Variable(2)
7  C = np.array([[1, 2], [3, 1]])
8
9  f = inner(x, C @ y)
10
11 f.is_dsp()  # True
12
13 f.convex_variables()  # [x]
14 f.concave_variables()  # [y]
15 f.affine_variables()  # []
```

Lines 1–3 import the necessary packages (which we will use but not show in the sequel). In lines 5–7, we create two CVXPY variables and a constant matrix. In line 9 we construct the saddle function `f` using the DSP atom `inner`. Both its arguments are affine, so this matches the DSP rules. In line 11 we check if `saddle_function` is DSP-compliant, which it is. In lines 13–15 we call functions that return lists of the convex, concave, and affine

variables, respectively. The results of lines 13–15 might seem odd, but recall that `inner` marks its first argument as convex and its second as concave.

5.3 Saddle point problems

Saddle point problem objective. To construct a saddle point problem, we first create an objective using

```
obj = MinimizeMaximize(f),
```

where `f` is a CVXPY expression. The objective `obj` is DSP-compliant if the expression `f` is DSP-compliant. This is analogous to the CVXPY contructors `cp.Minimize(f)` and `cp.Maximize(f)`, which create objectives from expressions.

Saddle point problem. A saddle point problem is constructed using

```
prob = SaddlePointProblem(obj, constraints, cvx_vars, ccv_vars)
```

Here, `obj` is a `MinimizeMaximize` objective, `constraints` is a list of constraints, `cvx_vars` is a list of convex variables and `ccv_vars` is a list of concave variables. The objective must be DSP-compliant for the problem to be DSP-compliant. We now describe the remaining conditions under which the constructed problem is DSP-compliant.

Each constraint in the list must be DCP, and can only involve convex variables or concave variables; convex and concave variables cannot both appear in any one constraint. The list of convex and concave variables partitions all the variables that appear in the objective or the constraints. In cases where the role of a variable is unambiguous, it is inferred, and does not need to be in either list. For example with the objective

```
MinimizeMaximize(weighted_log_sum_exp(x, y) + cp.exp(u) + cp.log(v) + z),
```

`x` and `u` must be convex variables, and `y` and `v` must be concave variables, and so do not need to appear in the lists used to construct a saddle point problem. The variable `z`, however, could be either a convex or concave variable, and so must appear in one of the lists.

The role of a variable can also be inferred from the constraints: Any variable that appears in a constraint with convex (concave) variables must also be convex (concave). With the objective above, the constraint `z + v <= 1` would serve to classify `z` as a concave variable. With this constraint, we could pass empty variable lists to the saddle point constructor, since the roles of all variables can be inferred. When the roles of all variables are unambiguous, the lists are optional.

The roles of the variables in a saddle point problem `prob` can be found by calling `prob.convex_variables()` and `prob.concave_variables()`, which return lists of variables, and is a partition of all the variables appearing in the objective or constraints. This is useful for debugging, to be sure that DSP agrees with you about the roles of all variables. A DSP-compliant saddle point problem must have an empty list of affine variables. (If it did not, the problem would be ambiguous.)

Solving a saddle point problem. The `solve()` method of a `SaddlePointProblem` object canonicalizes and solves the problem. This involves checking the objective and constraints for DSP-compliance. The conic representation of the problem is obtained, which involves setting up an auxiliary problem and compiling it using CVXPY. Then, the dualization is carried out, which results in another CVXPY problem which is then solved to yield the objective value. This has the side effect of setting all convex variables' `.value` attribute. To also obtain the values of the concave variables, the saddle point problem is solved again with a negated objective and the roles of the minimization and maximization variables reversed. We emphasize that as DSP acts as a compiler, it does not implement any optimization algorithms itself, but rather relies on the solvers accessible through CVXPY.

Example. Here we create and solve a matrix game, continuing the example above where `f` was defined. We do not need to pass in lists of variables since their roles can be inferred.

Creating and solving a matrix game.

```
obj = MinimizeMaximize(f)
constraints = [x >= 0, cp.sum(x) == 1, y >= 0, cp.sum(y) == 1]
prob = SaddlePointProblem(obj, constraints)

prob.is_dsp()  # True
prob.convex_variables()  # [x]
prob.concave_variables()  # [y]
prob.affine_variables()  # []

prob.solve()  # solves the problem
prob.value  # 1.6666666666666667
x.value  # array([0.66666667, 0.33333333])
y.value  # array([0.33333333, 0.66666667])
```

5.4 Saddle extremum functions

Local variables. An SE function has one of the forms

$$G(x) = \sup_{y \in \mathcal{Y}} f(x, y) \quad \text{or} \quad H(y) = \inf_{x \in \mathcal{X}} f(x, y),$$

where f is a saddle function. Note that y in the definition of G, and x in the definition of H, are local or dummy variables, understood to have no connection to any other variable. Their scope extends only to the definition, and not beyond.

To express this subtlety in DSP, we use the class `LocalVariable` to represent these dummy variables. The variables that are maximized over (in a saddle max function) or minimized over (in a saddle min function) must be declared using the `LocalVariable()` constructor. Any `LocalVariable` in an SE function cannot appear in any other SE function.

Constructing SE functions. We construct SE functions in DSP using

`saddle_max(f, constraints)` or `saddle_min(f, constraints)`.

Here, `f` is a CVXPY scalar expression, and `constraints` is a list of constraints. We now describe the rules for constructing a DSP-compliant SE function.

If a `saddle_max` is being constructed, `f` must be DSP-compliant, and the function's concave variables, and all variables appearing in the list of constraints, must be `LocalVariable`s, while the function's convex variables must all be regular `Variable`s. A similar rule applies for `saddle_min`.

The list of constraints is used to specify the set over which the sup or inf is taken. Each constraint must be DCP-compliant, and can only contain `LocalVariable`s.

With `x` a `Variable`, `y_loc` a `LocalVariable`, `z_loc` a `LocalVariable`, and `z` a `Variable`, consider the following two SE functions:

```
f_1 = saddle_max(inner(x, y_loc) + z, [y_loc <= 1])
f_2 = saddle_max(inner(x, y_loc) + z_loc, [y_loc <= 1, z_loc <= 1])
```

Both are DSP-compliant. For the first, calling `f_1.convex_variables()` would return `[x, z]`, and calling `f_1.concave_variables()` would return `[y_loc]`. For the second, calling `f_2.convex_variables()` would return `[x]`, and `f_2.concave_variables()` return `[y_loc, z_loc]`.

Let `y` be a `Variable`. Both of the following are not DSP-compliant:

```
f_3 = saddle_max(inner(x, y_loc) + z, [y_loc <= 1, z <= 1])
f_4 = saddle_max(inner(x, y) + z_loc, [y_loc <= 1, z_loc <= 1])
```

The first is not DSP-compliant because `z` is not a `LocalVariable`, but appears in the constraints. The second is not DSP-compliant because `y` is not a `LocalVariable`, but appears as a concave variable in the saddle function.

SE functions are DCP. When they are DSP-compliant, a `saddle_max` is a convex function, and a `saddle_min` is a concave function. They can be used anywhere in CVXPY that a convex or concave function is appropriate. You can add them, compose them (in appropriate ways), use them in the objective or either side of constraints (in appropriate ways).

Examples. Now we provide full examples demonstrating construction of a `saddle_max`, which we can use to solve the matrix game described in §5.3 as a saddle problem involving an SE function.

Creating a saddle max.

```
# Creating variables
x = cp.Variable(2)

```

```
# Creating local variables
y_loc = LocalVariable(2)

# Convex in x, concave in y_loc
f = saddle_inner(C @ x, y_loc)

# maximizes over y_loc
G = saddle_max(f, [y_loc >= 0, cp.sum(y_loc) == 1])
```

Note that `G` is a CVXPY expression. Constructing a `saddle_min` works exactly the same way.

5.5 Saddle problems

A saddle problem is a convex problem that uses SE functions. To be DSP-compliant, the problem must be DCP (which implies all SE functions are DSP-compliant). When you call the solve method on a saddle problem involving SE functions, and the solve is successful, then all variables' `.value` fields are overwritten with optimal values. This includes `LocalVariable`s that the SE functions maximized or minimized over; they are assigned to the value of *a particular* maximizer or minimizer of the SE function at the value of the non-local variables, with no further guarantees.

Example. We continue our example from §5.4 and solve the matrix game using either a saddle max.

Creating and solving a saddle problem using a saddle max to solve the matrix game.

```
prob = cp.Problem(cp.Minimize(G), [x >= 0, cp.sum(x) == 1])

prob.is_dsp()  # True

prob.solve()  # solving the problem
prob.value  # 1.6666666666666667
x.value  # array([0.66666667, 0.33333333])
```

6 Examples

In this section we give numerical examples, taken from §3, showing how to create DSP-compliant problems. The specific problem instances we take are small, since our main point is to show how easily the problems can be specified in DSP. But DSP will scale to far larger problem instances. Again, code and data for these examples are available at https://github.com/cvxgrp/dsp.

6.1 Robust bond portfolio construction

Our first example is the robust bond portfolio construction problem described in §3.1. We consider portfolios of $n = 20$ bonds, over a period $T = 60$ half-years, *i.e.*, 30 years. The bonds are taken as representative ones in a global investment grade bond portfolio; for more detail, see [LSB22]. The payments from the bonds are given by $C \in \mathbf{R}^{20\times 60}$, with cash flow of bond i in period t denoted $c_{i,t}$. The goal is to choose holdings $h \in \mathbf{R}^{20}_+$, with the portfolio constraint set $\mathcal{H}$ given by

$$\mathcal{H} = \{h \mid h \geq 0,\ p^T h = B\},$$

i.e., the investments must be nonnegative and have a total value (budget) B, which we take to be \$100. Here $p \in \mathbf{R}^{20}_+$ denotes the price of the bonds on September 12, 2022. The portfolio objective is

$$\phi(h) = \frac{1}{2}\|(h - h^{\mathrm{mkt}}) \circ p\|_1,$$

where $h^{\mathrm{mkt}} \in \mathbf{R}^{20}_+$ is the market portfolio scaled to a value of \$100, and $\circ$ denotes Hadamard or elementwise multiplication. This is called the turn-over distance, since it tells us how much we would need to buy and sell to convert our portfolio to the market portfolio.

The yield curve set $\mathcal{Y}$ is described in terms of perturbations to the nominal or current yield curve $y^{\mathrm{nom}} \in \mathbf{R}^{60}$, which is the yield curve on September 12, 2022. We take

$$\mathcal{Y} = \left\{ y^{\mathrm{nom}} + \delta \;\middle|\; \|\delta\|_\infty \leq \delta^{\max},\ \|\delta\|_1 \leq \kappa,\ \sum_{t=1}^{T-1}(\delta_{t+1} - \delta_t)^2 \leq \omega \right\}.$$

We interpret $\delta \in \mathbf{R}^{60}$ as a shock to the yield curve, which we limit elementwise, in absolute sum, and in smoothness. The specific parameter values are given by

$$\delta^{\max} = 0.02, \quad \kappa = 0.9, \quad \omega = 10^{-6}.$$

In the robust bond portfolio problem (10) we take $V^{\mathrm{lim}} = 90$, that is, the worst case value of the portfolio cannot drop below \$90 for any $y \in \mathcal{Y}$.

We solve the problem using the following code, where we assume the cash flow matrix `C`, the price vector `p`, the nominal yield curve `y_nom`, and the market portfolio `h_mkt` are defined.

Robust bond portfolio construction.

```
# Constants and parameters
n, T = C.shape
delta_max, kappa, omega = 0.02, 0.9, 1e-6
B = 100
V_lim = 90

# Creating variables
h = cp.Variable(n, nonneg=True)
```

```
9
10 delta = LocalVariable(T)
11 y = y_nom + delta
12
13 # Objective
14 phi = 0.5 * cp.norm1(cp.multiply(h, p) - cp.multiply(h_mkt, p))
15
16 # Creating saddle min function
17 V = 0
18 for i in range(n):
19     t_plus_1 = np.arange(T) + 1  # Account for zero-indexing
20     V += saddle_inner(cp.exp(cp.multiply(-t_plus_1, y)), h[i] * C[i])
21
22 Y = [
23     cp.norm_inf(delta) <= delta_max,
24     cp.norm1(delta) <= kappa,
25     cp.sum_squares(delta[1:] - delta[:-1]) <= omega,
26 ]
27
28 V_wc = saddle_min(V, Y)
29
30 # Creating and solving the problem
31 problem = cp.Problem(cp.Minimize(phi), [h @ p == B, V_wc >= V_lim])
32 problem.solve()  # 15.32
```

We first define the constants and parameters in lines 2–5, before creating the variable for the holdings `h` in line 8, and the `LocalVariable delta`, which gives the yield curve perturbation, in line 10. In line 11 we define `y` as the sum of the current yield curve `y_nom` and the perturbation `delta`. The objective function is defined in line 14. Lines 17–20 define the saddle function `V` via the `saddle_inner` atom. The yield uncertainty set `Y` is defined in lines 22–26, and the worst case portfolio value is defined in line 25 using `saddle_min`. We use the concave expression `saddle_min` to create and solve a CVXPY problem in lines 31–32.

Table 1 summarizes the results. The nominal portfolio is the market portfolio, which has zero turn-over distance to the market portfolio, *i.e.*, zero objective value. This nominal portfolio, however, does not satisfy the worst-case portfolio value constraint, since there are yield curves in $\mathcal{Y}$ that cause the portfolio value to drop to around \$87, less than our limit of \$90. The solution of the robust problem has turn-over distance \$15.32, and satisfies the constraint that the worst-case value be at least \$90.

6.2 Model fitting robust to data weights

We consider an instance of the model fitting problem described in §3.2. We use the well known Titanic data set [HC17], which gives several attributes for each passenger on the ill-

	Nominal portfolio	Robust portfolio
Turn-over distance	\$0.00	\$15.32
Worst-case value	\$86.99	\$90.00

Table 1: Turn-over distance and worst-case value for the nominal (market) portfolio and the robust portfolio. The nominal portfolio does not meet our requirement that the worst-case value be at least \$90.

fated Titanic voyage, including whether they survived. A classifier is fit to predict survival based on the features sex, age (binned into three groups, 0–26, 26–53, and 53–80), and class (1, 2, or 3). These features are encoded as a Boolean vector $a_i \in \mathbf{R}^7$. The label $y_i = 1$ means passenger i survived, and $y_i = -1$ otherwise. There are 1046 examples, but we fit our model using only the $m = 50$ passengers who embarked from Queenstown, one of three ports of embarkation. This is a somewhat non-representative sample; for example, the survival rate among Queenstown departures is 26%, whereas the overall survival rate is 40.8%. This is a common situation in machine learning, where the distribution of labels in the training data does not match that of the test dataset (known as label shift), for which we seek a robust solution.

We seek a linear classifier $\hat{y}_i = \mathbf{sign}(a_i^T\theta + \beta_0)$, where $\theta \in \mathbf{R}^7$ is the classifier parameter vector and $\beta_0 \in \mathbf{R}$ is the bias. The hinge loss and ℓ_2 regularization are used, given by

$$\ell_i(\theta) = \max(0, 1 - y_i a_i^T\theta), \qquad r(\theta) = \eta\|\theta\|_2^2,$$

with $\eta = 0.05$.

The data is weighted to partially correct for the different survival rates for our training set (26%) and the whole data set (40.8%). To do this we set $w_i = z_1$ when $y_i = 1$ and $w_i = z_2$ when $y_i = -1$. We require $w \geq 0$ and $\mathbf{1}^T w = 1$, and

$$0.408 - 0.05 \leq \sum_{y_i=1} w_i \leq 0.408 + 0.05.$$

Thus $\mathcal{W}$ consists of weights on the Queenstown departure samples that correct the survival rate to within 5% of the overall survival rate.

The code shown below solves this problem, where we assume the data matrix is already defined as `A_train` (with rows a_i^T), the survival label vector is defined as `y_train`, and the indicator of survival in the training set is defined as `surv`.

Model fitting robust to data weights.

```
# Constants and parameters
m, n = A_train.shape
inds_0 = surv == 0
inds_1 = surv == 1
eta = 0.05

```

	Nominal classifier	Robust classifier
Train accuracy	82.0%	80.0%
Test accuracy	76.0%	78.6%

Table 2: Nominal and worst-case objective values for classification and robust classification models.

```
# Creating variables
theta = cp.Variable(n)
beta_0 = cp.Variable()
weights = cp.Variable(m, nonneg=True)
surv_weight_0 = cp.Variable()
surv_weight_1 = cp.Variable()

# Defining the loss function and the weight constraints
y_hat = A_train @ theta + beta_0
loss = cp.pos(1 - cp.multiply(y_train, y_hat))
objective = MinimizeMaximize(saddle_inner(loss, weights)
    + eta * cp.sum_squares(theta))

constraints = [
    cp.sum(weights) == 1,
    0.408 - 0.05 <= weights @ surv,
    weights @ surv <= 0.408 + 0.05,
    weights[inds_0] == surv_weight_0,
    weights[inds_1] == surv_weight_1,
]

# Creating and solving the problem
problem = SaddlePointProblem(objective, constraints)
problem.solve()
```

After defining the constants and parameters in lines 2–5, we specify the variables for the model coefficient and the weights in lines 8–9 and 10–12, respectively. The loss function and regularizer which make up the objective are defined next in lines 15–18. The weight constraints are defined in lines 20–26. The saddle point problem is created and solved in lines 29 and 30.

The results are shown in table 2. We report the test accuracy on all samples in the dataset with a different port of embarkation than Queenstown (996 samples). We see that while the robust classification model has slightly lower training accuracy than the nominal model, it achieves a higher test accuracy, generalizing from the non-representative training data better than the nominal classifier, which uses uniform weights.

6.3 Robust Markowitz portfolio construction

We consider the robust Markowitz portfolio construction problem described in §3.4. We take $n = 6$ assets, which are the (five) Fama-French factors [FF15] plus a risk-free asset. The data is obtained from the Kenneth R. French data library [Fre22], with monthly return data available from July 1963 to October 2022. The nominal return and risk are the empirical mean and covariance of the returns. (These obviously involve look-ahead, but the point of the example is how to specify and solve the problem with DSP, not the construction of a real portfolio.) We take parameters $\rho = 0.02$, $\eta = 0.2$, and risk aversion parameter $\gamma = 1$.

In the code, we use `mu` and `Sigma` for the mean and covariance estimates, respectively, and the parameters are denoted `rho`, `eta`, and `gamma`.

Robust Markowitz portfolio construction.

```
# Constants and parameters
n = len(mu)
rho, eta, gamma = 0.2, 0.2, 1

# Creating variables
w = cp.Variable(n, nonneg=True)

delta_loc = LocalVariable(n)
Sigma_perturbed = LocalVariable((n, n), PSD=True)
Delta_loc = LocalVariable((n, n))

# Creating saddle min function
f = w @ mu + saddle_inner(delta_loc, w) \
    - gamma * saddle_quad_form(w, Sigma_perturbed)

Sigma_diag = Sigma.diagonal()
local_constraints = [
    cp.abs(delta_loc) <= rho, Sigma_perturbed == Sigma + Delta_loc,
    cp.abs(Delta_loc) <= eta * np.sqrt(np.outer(Sigma_diag, Sigma_diag))
]

G = saddle_min(f, local_constraints)

# Creating and solving the problem
problem = cp.Problem(cp.Maximize(G), [cp.sum(w) == 1])
problem.solve()  # 0.076
```

We first define the constants and parameters, before creating the weights variable in line 6, and the local variables for the perturbations in lines 8–10. The saddle function for the objective is defined in line 13, followed by the constraints on the perturbations. Both

	Nominal portfolio	Robust portfolio
Nominal objective	.295	.291
Robust objective	.065	.076

Table 3: Nominal and worst-case objective for the nominal and robust portfolios.

are combined into the concave saddle min function, which is maximized over the portfolio constraints in lines 25–26.

The results are shown in table 3. The robust portfolio yields a slightly lower risk adjusted return of 0.291 compared to the nominal optimal portfolio with 0.295. But the robust portfolio attains a higher worst-case risk adjusted return of 0.076, compared to the nominal optimal portfolio which attains 0.065.

Acknowledgements

P. Schiele is supported by a fellowship within the IFI program of the German Academic Exchange Service (DAAD). This research was partially supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR, and by ONR N000142212121.

References

[Mar52] H. Markowitz. "Portfolio Selection". In: *Journal of Finance* 7 (1 1952), pp. 77–91.

[MVN53] O. Morgenstern and J. Von Neumann. *Theory of games and economic behavior*. Princeton University Press, 1953.

[Sio58] M. Sion. "On general minimax theorems". In: *Pacific Journal of Mathematics* 8.1 (1958), pp. 171–176.

[Roc70] R. Rockafellar. *Convex analysis*. Vol. 18. Princeton university press, 1970.

[Kor76] G. Korpelevich. "The extragradient method for finding saddle points and other problems". In: *Matecon* 12 (1976), pp. 747–756.

[DM90] V. Dem'yanov and V. Malozemov. *Introduction to minimax*. Courier Corporation, 1990.

[BL91] F. Black and R. Litterman. "Asset Allocation". In: *The Journal of Fixed Income* 1.2 (1991), pp. 7–18. ISSN: 1059-8596.

[NN92] Y. Nesterov and A. Nemirovski. "Conic formulation of a convex programming problem and duality". In: *Optimization Methods & Software* 1 (1992), pp. 95–115.

[HK93] R. Hettich and K. Kortanek. "Semi-infinite programming: Theory, methods, and applications". In: *SIAM review* 35.3 (1993), pp. 380–429.

[DP95] D. Du and P. Pardalos. *Minimax and applications*. Vol. 4. Springer Science & Business Media, 1995.

[Nem99] A. Nemirovski. "On self-concordant convex–concave functions". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 303–384.

[LB00] M. Lobo and S. Boyd. *The worst-case risk of a portfolio*. Tech. rep. 2000.

[GI03] D. Goldfarb and G. Iyengar. "Robust Portfolio Selection Problems". In: *Mathematics of Operations Research* 28.1 (2003), pp. 1–38. ISSN: 0364765X, 15265471.

[HT03] B. Halldórsson and R. Tütüncü. "An interior-point method for a class of saddle-point problems". In: *Journal of Optimization Theory and Applications* 116.3 (2003), pp. 559–590.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Lof04] J. Lofberg. "YALMIP : a toolbox for modeling and optimization in MATLAB". In: *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*. 2004, pp. 284–289.

[Nem04] A. Nemirovski. “Prox-method with rate of convergence O(1/t) for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems”. In: *SIAM Journal on Optimization* 15.1 (2004), pp. 229–251.

[Nes05a] Y. Nesterov. “Excessive gap technique in nonsmooth convex minimization”. In: *SIAM Journal on Optimization* 16.1 (2005), pp. 235–249.

[Nes05b] Y. Nesterov. “Smooth minimization of non-smooth functions”. In: *Mathematical programming* 103.1 (2005), pp. 127–152.

[BL06] J. Borwein and A. Lewis. *Convex Analysis*. Springer, 2006.

[GBY06] M. Grant, S. Boyd, and Y. Ye. “Disciplined convex programming”. In: *Global optimization*. Springer, 2006, pp. 155–210.

[NP06] Y. Nesterov and B. Polyak. “Cubic regularization of Newton method and its global performance”. In: *Mathematical Programming* 108.1 (2006), pp. 177–205.

[Nes07] Y. Nesterov. “Dual extrapolation and its applications to solving variational inequalities and related problems”. In: *Mathematical Programming* 109.2 (2007), pp. 319–344.

[Nes08] Y. Nesterov. “Accelerating the cubic regularization of Newton’s method on convex problems”. In: *Mathematical Programming* 112.1 (2008), pp. 159–181.

[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Vol. 28. Princeton university press, 2009.

[MB09] A. Mutapcic and S. Boyd. “Cutting-set methods for robust convex optimization with pessimizing oracles”. In: *Optimization Methods & Software* 24.3 (2009), pp. 381–406.

[NO09] A. Nedić and A. Ozdaglar. “Subgradient methods for saddle-point problems”. In: *Journal of optimization theory and applications* 142.1 (2009), pp. 205–228.

[RW09] R. Rockafellar and R. Wets. *Variational analysis*. Vol. 317. Springer Science & Business Media, 2009.

[BBC11] D. Bertsimas, D. Brown, and C. Caramanis. “Theory and applications of robust optimization”. In: *SIAM review* 53.3 (2011), pp. 464–501.

[CP11] A. Chambolle and T. Pock. “A first-order primal-dual algorithm for convex problems with applications to imaging”. In: *Journal of mathematical imaging and vision* 40.1 (2011), pp. 120–145.

[CLO13] Y. Chen, G. Lan, and Y. Ouyang. *Optimal Primal-Dual Methods for a Class of Saddle Point Problems*. 2013.

[Con13] L. Condat. “A primal–dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms”. In: *Journal of optimization theory and applications* 158.2 (2013), pp. 460–479.

[Goo+14] I. Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014.

[GB14] M. Grant and S. Boyd. “CVX: Matlab software for disciplined convex programming, version 2.1”. In: (2014).

[Ude+14] M. Udell et al. “Convex Optimization in Julia”. In: *SC14 Workshop on High Performance Technical Computing in Dynamic Languages* (2014).

[BS15] K. Bredies and H. Sun. “Preconditioned Douglas–Rachford splitting methods for convex-concave saddle-point problems”. In: *SIAM Journal on Numerical Analysis* 53.1 (2015), pp. 421–444.

[FF15] E. Fama and K. French. “A five-factor asset pricing model”. In: *Journal of Financial Economics* 116.1 (2015), pp. 1–22. ISSN: 0304-405X.

[CP16] A. Chambolle and T. Pock. “On the ergodic convergence rates of a first-order primal–dual algorithm”. In: *Mathematical Programming* 159.1 (2016), pp. 253–287.

[DB16] S. Diamond and S. Boyd. “CVXPY: A Python-embedded modeling language for convex optimization”. In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[Boy+17] S. Boyd et al. “Multi-Period Trading via Convex Optimization”. In: *Foundations and Trends in Optimization* 3.1 (2017), pp. 1–76.

[HC17] F. Harrell Jr. and T. Cason. *Titanic dataset*. 2017. URL: https://www.openml.org/d/40945.

[CPT18] G. Cornuéjols, J. Peña, and R. Tütüncü. *Optimization Methods in Finance*. 2nd ed. Cambridge University Press, 2018.

[DA19] X. Dou and M. Anitescu. “Distributionally robust optimization with correlated data from vector autoregressive processes”. In: *Operations Research Letters* 47.4 (2019), pp. 294–299. ISSN: 0167-6377.

[The+19] K. Thekumparampil et al. “Efficient Algorithms for Smooth Minimax Optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[BGM20] T. Broderick, R. Giordano, and R. Meager. “An Automatic Finite-Sample Robustness Metric: When Can Dropping a Little Data Make a Big Difference?” In: *arXiv preprint arXiv:2011.14999* (2020).

[FNB20] A. Fu, B. Narasimhan, and S. Boyd. “CVXR: An R Package for Disciplined Convex Optimization”. In: *Journal of Statistical Software* 94.14 (2020), pp. 1–34.

[LJJ20] T. Lin, C. Jin, and M. Jordan. “Near-optimal algorithms for minimax optimization”. In: *Conference on Learning Theory*. PMLR. 2020, pp. 2738–2779.

[BAB21] S. Barratt, G. Angeris, and S. Boyd. "Optimal representative sample weighting". In: *Statistics and Computing* 31.2 (2021), pp. 1–14.

[Fre22] K. French. *Kenneth R. French Data Library*. 2022.

[JN22] A. Juditsky and A. Nemirovski. "On well-structured convex–concave saddle point problems and variational inequalities with monotone operators". In: *Optimization Methods and Software* 37.5 (2022), pp. 1567–1602.

[LSB22] E. Luxenberg, P. Schiele, and S. Boyd. *Robust Bond Portfolio Construction via Convex-Concave Saddle Point Optimization*. 2022.

[RY22] E. Ryu and W. Yin. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 2022.

Part IV.

Robust Bond Portfolio Construction via Convex-Concave Saddle Point Optimization

This chapter is a reprint of:

Luxenberg, E., Schiele, P., Boyd, S. (2022). Robust Bond Portfolio Construction via Convex-Concave Saddle Point Optimization. *Available at https://arxiv.org/abs/2212.02570*. Published in the *Journal of Optimization Theory and Applications*, *available at https://link.springer.com/article/10.1007/s10957-024-02436-z*.

The included version is the second revision of the essay.

Author contributions:

During discussions about bond portfolio management, Eric Luxenberg recognized that the concavity of the bond price in the yields and spreads could be used to model the worst case value of the bond. Philipp Schiele contributed relevant practical insights from his experience in the financial industry and literature research on bond portfolio management. Philipp Schiele and Eric Luxenberg then developed the initial draft, with Philipp Schiele leading the implementation of the model and conducting the numerical experiments, including the data collection. The draft was refined by Stephen Boyd, who also supervised the project throughout.

Supplementary material available at: https://github.com/cvxgrp/robust_bond_portfolio

Robust Bond Portfolio Construction via Convex-Concave Saddle Point Optimization

Eric Luxenberg*[1], Philipp Schiele*[2], and Stephen Boyd[1]

[1]Department of Electrical Engineering, Stanford University
[2]Department of Statistics, Ludwig-Maximilians-Universität München

January 11, 2024

Abstract

The minimum (worst case) value of a long-only portfolio of bonds, over a convex set of yield curves and spreads, can be estimated by its sensitivities to the points on the yield curve. We show that sensitivity based estimates are conservative, *i.e.*, underestimate the worst case value, and that the exact worst case value can be found by solving a tractable convex optimization problem. We then show how to construct a long-only bond portfolio that includes the worst case value in its objective or as a constraint, using convex-concave saddle point optimization.

*Equal contribution.

Contents

1 Introduction

We consider a long-only portfolio of bonds, and address the problem of robust analysis and portfolio construction, under a worst case framework. In this framework we have a set of possible yield curves and bond spreads, and consider the worst change in value of the portfolio over this uncertainty set.

In the analysis problem, considered in §3, we fix the portfolio, and ask what is the worst case change in portfolio value. We observe that this is a convex optimization problem, readily solved using standard frameworks or domain specific languages (DSLs) for convex optimization. We also consider the linearized version of the same problem, where the true portfolio value is replaced with its first order Taylor approximation. This approximation can be interpreted as using standard methods to analyze bond portfolio value using durations. We show that this is also a convex optimization problem, and is always conservative, *i.e.*, predicts more of a decrease in portfolio value than the exact method.

In the robust portfolio construction problem, considered in §4, we seek a portfolio of bonds that minimizes an objective that includes a robustness term, *i.e.*, the worst case change in value of the portfolio over the set of possible yield curves and spreads. We show that this problem, and its linearized version, can be formulated as convex-concave saddle point problems, where we identify the worst case yield and spread and at the same time, the optimal portfolio. One interesting ramification of the convex-concave saddle point formulation is that, unlike in general worst case (minimax) optimization problems, where there are generally multiple worst case parameters, we need to consider only one worst case yield curve and set of bond spreads.

In §5 we show how the convex-concave saddle point problem can be solved by solving one convex optimization problem. This is done using the well known technique of expressing the worst case portfolio value as the optimal value of the dual problem, which converts a min-max problem into a min-min problem which we directly solve. We illustrate the method for a specific case, and in a companion paper [SLB23] explain how the reformulation technique can be automated, using methods due to Juditsky and Nemirovski [JN21]. Using our disciplined saddle point programming framework, we can pose the robust bond portfolio construction problem in just a few lines of simple and natural code, and solve it efficiently.

We present several variations and extensions in §6, including cases where the bond portfolio contains bonds with different base yield curves, per-period compounding is used to value bonds, and a formulation with a robustness constraint as opposed to an objective term.

Identifying the convex-concave structure of the robust bond portfolio construction problem is a novel theoretical contribution, and allows us to use the powerful theory of convex duality to extract insights such as the existence of a single worst case yield curve, or construct a robust bond portfolio. In addition, while this paper was under review the importance of properly managing risk in a bond portfolio became quite apparent with the collapse of Silicon Valley Bank, where a major factor was the bank's exposure to interest rate risk.

1.1 Previous and related work

Bond portfolio construction and analysis. Portfolio construction and analysis are well studied problems in finance, however, most of the literature focuses on equity portfolios. These approaches often can not be directly applied to bond portfolios, as there are important differences between the asset classes, such as the finite maturity of bonds. Yet, by making assumptions about the re-investment rate, bond portfolios can be constructed and analyzed via modern portfolio theory (MPT) [Mar52]. For example, assuming that the re-investment rate is given by the current spot rate, standard MPT can be applied to bond portfolios, where the mean and covariance of the bonds can be derived from sample moments or factor models [Puh08]. Similarly, [KK06] use a factor model for the term structure in an MPT setting.

Other approaches for bond portfolio construction that are not based on MPT include exact matching and immunization [EGBG09]. Exact matching is a method for constructing a bond portfolio that minimizes the required investment amount while ensuring that cash flows arising from liabilities are being met. Immunization refers to matching the duration of assets and liabilities, so that the portfolio value is insensitive to (small) changes in interest rates. Both of these problems can be formulated as linear programs and so tractably solved.

Likewise, the factors influencing bond (portfolio) values are well understood, given the practical implications of the problem [FM12, EGBG09]. However, most of the existing literature focuses on parallel shifts in yield curves and spreads, leading to a trivial worst case scenario. Thus the literature is sparse when it comes to robust bond portfolio construction as it relates to possible changes in yield curves and spreads. Instead, most existing work focuses on the problem of robust portfolio construction under parameter uncertainty in an MPT framework (see, *e.g.*, [TK04, KKF14]).

Convex-concave saddle point optimization. Convex-concave saddle point problems are a class of optimization problems with objective functions which are convex in a subset of the optimization variables, and concave in the remaining variables. The goal in such problems is to find a saddle point, *i.e.*, values of the convex variables that minimize the objective, and values of the concave variables that maximize it. Convex-concave saddle point problems have been studied for decades. Indeed, much of the theory of game theory is based on solving convex-concave saddle point problems, with early descriptions dating back to the 1920s [VN28], and solutions based on solving them as a single convex problem via duality dating back to the 1950s [VNM53]. In their 1983 book, Nemirovski and Yudin [NY83] describe the oracle complexity of first order optimization methods for convex-concave saddle point problems, based on their previous work on the convergence of the gradient method for convex-concave saddle point problems [NY78]. Since then, existing work either requires a specific structure of the problem such as convex and concave variables only being coupled via a bilinear term [BS16], or only under strong assumptions on the functions' properties [Nem04].

More recently, Juditsky and Nemirovski [JN21] proposed a general framework for solving convex-concave saddle point problems with a particular conic structure as a single convex minimization via dualization. Building on Juditsky and Nemirovski's work, the authors

of this paper developed disciplined saddle programming (DSP) [SLB23]. The associated DSL makes it easy to express a wide class of convex-concave saddle point problems in a natural way; the problem is then transformed to a single convex optimization problem using Juditsky and Nemirovski's methods. This is analogous to disciplined convex programming (DCP), which makes it easy to specify and solve a wide variety of convex optimization problems [DB16]. The authors' DSP software package allows our formulation of the robust portfolio construction problem to be specified in just a few lines of clear code.

2 Bond portfolio value

2.1 Yield curve and spreads

A bond is a financial contract that obligates the issuer to make a series of specified payments over time to the bond holder. We let $t = 1, \ldots, T$ denote time periods, with $t = 0$ representing now. (The periods are usually six months, a typical time between bond coupon payments.) We represent the bond payments as a vector $c \in \mathbf{R}_+^T$, where T is the number of periods, and $\mathbf{R}_+$ denotes the set of nonnegative reals. For each $t = 1, \ldots, T$, c_t is the payment in period t to the bond holder. A bond has a maturity, which is the period of its last payment; for t larger than the maturity, we have $c_t = 0$. The cash flows c_t include coupon payments as well as the payment of the face value at maturity.

We consider a portfolio of n bonds, with quantities (also called holdings) $h = (h_1, \ldots, h_n) \in \mathbf{R}_+^n$ of each bond, assumed nonnegative (*i.e.*, only long positions). We assume that all bonds in the portfolio mature at or before time period T. We first review some basic facts about bonds, for completeness and also to fix our notation.

Each bond has a known cash flow or sequence of payments, given by $c_i \in \mathbf{R}_+^T$, $i = 1, \ldots, n$. We write $c_{i,t}$ to denote the cash flow from bond i in period t. We have $c_{i,t} = 0$ for t larger than the maturity of bond i. We let $p \in \mathbf{R}_+^n$ denote the price of the bonds. The portfolio value is $V = p^T h$. The bond prices are modeled using a base yield curve and spreads for each bond, explained below.

Yield curve. The yield curve is denoted by $y \in \mathbf{R}^T$. The yield curve gives the discount of a future payment, *i.e.*, the current value of a payment of \$1 received in period t, denoted P_t. These are given by

$$P_t = \exp(-ty_t), \quad t = 1, \ldots, T.$$

We will work with per-period yields, to simplify the formulas, but following convention, we present all final numerical results as annualized. (For example, if the periods represent six months, the associated annualized yields are given by $2y_t$.) We use continuous compounding for simplicity of notation, but all our results readily extend to period-wise compounding, where $P_t = (1 + y_t)^{-t}$ (see §6).

The yield curve gives the discount of future payments, and captures market expectations with respect to macroeconomic factors, fiscal and monetary policy interactions, and the vulnerability of private consumption to future (unexpected) shocks.

Bond spreads. Bond i has spread $s_i \geq 0$, which means that the bond is priced at its net present value using the yield curve $y + s_i\mathbf{1}$, where $\mathbf{1}$ is the vector with all entries one. This is referred to as a 'parallel shift' applied to the base yield curve. We will work with per-period spreads, to simplify the formulas, but will give final results as annualized.

The spread captures the uncertainty in the cash flow associated with the bond, such as default or other optionality, which means that we value a payment of \$1 from the bond at period t as $\exp(-t(y_t + s_i))$, which is less than or equal to $P_t = \exp(-ty_t)$. The riskier the bond, the larger the spread, which means future payments are discounted more heavily.

Bond price. The price of a bond is modeled as the net present value of its cash flow using these discounts,

$$p_i = \sum_{t=1}^{T} c_{i,t} \exp(-t(y_t + s_i)), \quad i = 1, \ldots, n. \tag{1}$$

Portfolio value. The portfolio value can be expressed as

$$V = p^T h = \sum_{i=1}^{n} \sum_{t=1}^{T} h_i c_{i,t} \exp(-t(y_t + s_i)). \tag{2}$$

For reasons mentioned below, it will be convenient to work with the log of the portfolio value,

$$\log V = \log \left(\sum_{i=1}^{n} \sum_{t=1}^{T} h_i c_{i,t} \exp(-t(y_t + s_i)) \right). \tag{3}$$

The portfolio value and log portfolio value are functions of the holdings h, the yield curve y, and the spreads s, but we suppress this dependence to keep the notation light. (The cash flows $c_{i,t}$ are fixed and given.)

Convexity properties. The portfolio value V is a linear function of h, the vector of holdings, for fixed yield curve and spreads. If we fix the holdings, V is a convex function of (y, s), the yield curve and spreads [BV04, Chap. 3]. The log value $\log V$ is a concave function of h, for fixed y and s, since it is a concave function of a linear function. The log value is a convex function of (y, s), for fixed h, since it can be expressed as

$$\log V = \log \left(\sum_{i=1}^{n} \sum_{t=1}^{T} \exp(-t(y_t + s_i) + \log h_i + \log c_{i,t}) \right),$$

which is the log-sum-exp function of an affine function of (y, s) [BV04, §3.1.5]. Thus $\log V$ is a convex-concave function, concave in h and convex in (y, s).

2.2 Change in bond portfolio value

We are interested in the change in portfolio value when the yield curve and spreads change from their current or nominal values $(y^{\rm nom}, s^{\rm nom})$ to the values (y, s), with the holdings fixed at $h^{\rm nom}$. We let V denote the portfolio value with yield curve and spreads (y, s), and $V^{\rm nom}$ the portfolio value with yield curve and spreads $(y^{\rm nom}, s^{\rm nom})$, both with holdings $h^{\rm nom}$. The relative or fractional change in value is given by $V/V^{\rm nom} - 1$. It is convenient to work with the change in the log value,

$$\Delta = \log(V/V^{\rm nom}) = \log V - \log V^{\rm nom}. \tag{4}$$

The relative change in value can be expressed in terms of the change in log value as $\exp\Delta - 1$. Both Δ and the relative change in value are readily interpreted. For example, $\Delta = -0.15$ means the portfolio value decreases by the factor $\exp(-0.15) = 0.861$, *i.e.*, a relative decrease of 13.9%.

Since $\log V$ is a convex function of (y, s), Δ is also a convex function of (y, s).

First order Taylor approximation. The first order Taylor approximation of change in log value, denoted $\hat{\Delta}$, is

$$\hat{\Delta} = D_{\rm yld}^T(y - y^{\rm nom}) + D_{\rm spr}^T(s - s^{\rm nom}) \approx \Delta, \tag{5}$$

where

$$D_{\rm yld} = (\nabla_y \log V)|_{y=y^{\rm nom},\, s=s^{\rm nom}} \in \mathbf{R}^T, \qquad D_{\rm spr} = (\nabla_s \log V)|_{y=y^{\rm nom},\, s=s^{\rm nom}} \in \mathbf{R}^n$$

are the gradients of the log value with respect to the yield curve and spreads, respectively, evaluated at the current value $(y^{\rm nom}, s^{\rm nom})$. These are given by

$$\begin{aligned}
(D_{\rm yld})_t &= -(1/V^{\rm nom}) \sum_{i=1}^{n} t h_i^{\rm nom} c_{i,t} \exp(-t(y_t^{\rm nom} + s_i^{\rm nom})), \\
(D_{\rm spr})_i &= -(1/V^{\rm nom}) \sum_{t=1}^{T} t h_i^{\rm nom} c_{i,t} \exp(-t(y_t^{\rm nom} + s_i^{\rm nom})).
\end{aligned}$$

(In the first expression we sum over the bonds, while in the second we sum over the periods.) The affine approximation (5) is very accurate when (y, s) is near $(y^{\rm nom}, s^{\rm nom})$.

The gradients $D_{\rm yld}$ and $D_{\rm spr}$ can be given traditional interpretations. When $n = 1$, *i.e.*, the portfolio consists of a single bond, $D_{\rm spr}$ is the *duration* of the bond. When $n = 1$ and t is one of the 12 Treasury spot maturities, $(D_{\rm spr})_t$ is a *key rate duration* of the bond. (We use the symbol D since the entries of the gradients can be interpreted as durations.) We refer to the Taylor approximation (5) as the duration based approximation.

A global lower bound. Since Δ is a convex function of (y, s), its Taylor approximation $\hat{\Delta}$ is a global lower bound on Δ (see, *e.g.*, [BV04, §3.1.3]): For any (y, s) we have

$$\hat{\Delta} = D_{\mathrm{yld}}^T(y - y^{\mathrm{nom}}) + D_{\mathrm{spr}}^T(s - s^{\mathrm{nom}}) \leq \Delta. \tag{6}$$

Note that this inequality holds for any (y, s), whereas the approximation (5) is accurate only for (y, s) near $(y^{\mathrm{nom}}, s^{\mathrm{nom}})$. Thus the duration based approximation of the change in log portfolio value is conservative; the true change in log value will be larger than the approximated change in log value.

We can easily obtain a bound on the relative change in portfolio value. Exponentiating the inequality (6) and using the inequality $\exp u \geq 1 + u$, we have

$$V/V^{\mathrm{nom}} - 1 = \exp\Delta - 1 \geq \exp\hat{\Delta} - 1 \geq \hat{\Delta}.$$

Therefore $\hat{\Delta}$ is also a lower bound on the relative change in portfolio value using the duration based approximation; the actual fractional change in value will always be more (positive) than the prediction.

3 Worst case analysis

In this section we assume the portfolio holdings are known and fixed as h^{nom}, and consider a nonempty compact convex set $\mathcal{U} \subset \mathbf{R}^T \times \mathbf{R}^n$ of possible yield curves and spreads. (We will say more about choices of $\mathcal{U}$ in §3.3.) We define the *worst case portfolio value* as

$$V^{\mathrm{wc}} = \min_{(y,s)\in\mathcal{U}} V,$$

i.e., the smallest possible portfolio value over the set of possible yield curves and spreads. It will be convenient to work with the worst case (*i.e.*, most negative) change in log portfolio value, defined as

$$\Delta^{\mathrm{wc}} = \min_{(y,s)\in\mathcal{U}} \Delta = \log V^{\mathrm{wc}} - \log V^{\mathrm{nom}}.$$

When $(y^{\mathrm{nom}}, s^{\mathrm{nom}}) \in \mathcal{U}$, the worst case log value change is nonpositive.

3.1 Worst case analysis problem

We can evaluate Δ^{wc} by solving the convex optimization problem

$$\begin{array}{ll} \text{minimize} & \Delta \\ \text{subject to} & (y, s) \in \mathcal{U}, \end{array} \tag{7}$$

with variables y and s. The optimal value of this problem is Δ^{wc} (from which we can obtain V^{wc}); by solving it, we also find an associated worst case yield curve and spread, which are themselves interesting. We refer to (7) as the *worst case analysis problem*.

Implications. One consequence is that we can evaluate Δ^{wc} very efficiently using standard methods of convex optimization [BV04]. Depending on the uncertainty set $\mathcal{U}$, the problem (7) can be expressed very compactly and naturally using domain specific languages for convex optimization, such as CVXPY [DB16], CVX [GB14], Convex.jl [UMZ+14], or CVXR [FNB20]. Appendix A gives an example illustrating how simple and natural the full CVXPY code to solve the worst case analysis problem is.

Maximum element. We mention here a special case with a simple analytical solution. The objective Δ is monotone nonincreasing in its arguments, *i.e.*, increasing any y_t or s_i reduces the portfolio value. It follows that if $\mathcal{U}$ has a maximum element $(y^{\max}, s^{\max})$, *i.e.*,

$$(y^{\max}, s^{\max}) \geq (y, s) \text{ for all } (y, s) \in \mathcal{U}$$

(with the inequality elementwise), then it is the solution of the worst case analysis problem. As a simple example, consider

$$\mathcal{U} = \{(y, u) \mid y^{\min} \leq y \leq y^{\max},\ s^{\min} \leq s \leq s^{\max}\},$$

i.e., we are given a range of possible values for each point in the yield curve, and for each spread. This uncertainty set, which is a hyper-rectangle or box, has maximum element $(y^{\max}, s^{\max})$, which is (obviously) the choice that minimizes portfolio value.

More interesting choices of uncertainty sets do not have a maximum element; for these cases we must numerically solve the worst case analysis problem (7).

3.2 Linearized worst case analysis problem

We can replace the objective in (7) with the lower bound (6) to obtain the linearized worst case portfolio value problem

$$\begin{array}{ll} \text{minimize} & \hat{\Delta} = D_{\text{yld}}^T(y - y^{\text{nom}}) + D_{\text{spr}}^T(s - s^{\text{nom}}) \\ \text{subject to} & (y, s) \in \mathcal{U}, \end{array} \tag{8}$$

with variables y and s. Here the objective is affine, whereas in (7) the objective is nonlinear (but convex). From the inequality (6), solving this linearized worst case analysis problem gives us a lower bound on Δ^{wc}, as well as a very good approximation when the changes in yield curve and spreads are not large. We refer to (8) as the *linearized worst case analysis problem*, and we denote its optimal value, the worst case change in log value predicted by the linearized approximation, by $\hat{\Delta}^{\text{wc}}$. This estimate of Δ^{wc} is conservative, *i.e.*, we have $\hat{\Delta}^{\text{wc}} \leq \Delta^{\text{wc}}$. The linearized problem is commonly used in practice, and therefore provides a baseline for comparison.

3.3 Yield/spread uncertainty sets

In this section we describe some possible choices of the yield/spread uncertainty set $\mathcal{U}$, described as a list of constraints. Before getting to specifics, we make some comments about high level methods one might use to construct uncertainty sets.

From all historical data. Here we construct $\mathcal{U}$ from all historical data. This conservative approach measures the sensitivity of the portfolio to the yield and spread changing to any previous value, or to a value consistent with some model of the past that we build.

From recent historical data. Here we construct $\mathcal{U}$ from recent historical data, or create a model that places higher weight on recent data. The idea here is to model plausible changes to the yield curve and spreads using a model based on recent historical values.

From forecasts of future values. Here we construct $\mathcal{U}$ as a forecasted set of possible values over the future, for example a confidence set associated with some predictions.

From current yield and spread estimation error. Here $\mathcal{U}$ represents the set of possible values of the *current* yield and spreads, which acknowledges that the current values are only estimates of some true but unknown value. See, for example, [FPY22] for a discussion of yield curve estimation and a method that can provide uncertainty quantification.

3.3.1 Scenarios

Here $\mathcal{U}$ is the convex hull of a set of yield curves and spreads,

$$\mathcal{U} = \mathbf{conv}\{(y^1, s^1), \ldots, (y^K, s^K)\},$$

which is a polyhedron defined by its vertices. In this case we can think of (y^k, s^k) as K economic regimes or scenarios. In the linearized worst case analysis problem, we minimize a linear function over this polyhedron, so there is always a solution at a vertex, *i.e.*, the worst case yield curve and spread is one of our scenarios. In this case we can solve the worst case analysis problem by simply evaluating the portfolio value for each of our scenarios, and taking the smallest value. When using the true portfolio value, however, we must solve the problem numerically, since the worst case scenario need not be on the vertex; it can be a convex combination of multiple vertices.

3.3.2 Confidence ellipsoid

Another natural uncertainty set is based on a vector Gaussian model of (y, s), with mean $\mu \in \mathbf{R}^{T+n}$ and covariance $\Sigma \in \mathbf{S}^{T+n}_{++}$, where $\mathbf{S}^k_{++}$ denotes the set of symmetric positive definite $k \times k$ matrices. We take $\mathcal{U}$ as the associated $(1-\alpha)$-confidence ellipsoid,

$$\mathcal{U} = \{(y, s) \mid ((y, s) - \mu)^T \Sigma^{-1} ((y, s) - \mu) \leq F^{-1}(1-\alpha)\},$$

where F is the cumulative distribution function of a χ^2 distribution with $T + n$ degrees of freedom.

3.3.3 Factor model

A standard method for describing yield curves and spreads is via a factor model, with

$$(y, s) = Zf + v,$$

where $f \in \mathbf{R}^k$ is a vector of factors that drive yield curves and spreads, and v represents idiosyncratic variation, *i.e.*, not due to the factors. (In a statistical model, the entries of v are assumed to be uncorrelated to each other and the factor f.) The matrix $Z \in \mathbf{R}^{(T+n)\times k}$ gives the factor loadings of the yield curve values and spreads.

Typical factors include treasury yields with various maturities, as well as other economic quantities. A simple factor model for yields can contain only two or three factors, which are the first few principal components of historical yield curves, called level, slope, and curvature [LS91, CP05].

Using a factor model, we can specify $\mathcal{U}$ by giving an uncertainty set $\mathcal{F} \subset \mathbf{R}^k$ for the factors, for example as

$$\mathcal{U} = \{Zf + v \mid f \in \mathcal{F},\ \|D^{-1}v\|_2^2 \leq 1\},$$

where D is a positive diagonal matrix with its entries giving the idiosyncratic variation of individual yield curve and spread values. We note that while a factor model is typically used to develop a statistical model of the yield curve and spreads, we use it here to define a (deterministic) set of possible values.

3.3.4 Perturbation description

The uncertainty set $\mathcal{U}$ can be described in terms of possible perturbations to the current values $(y^{\text{nom}}, s^{\text{nom}})$. We describe this for the yield curve only, but similar ideas can be used to describe the spreads as well. We take $y = y^{\text{nom}} + \delta$, where $\delta \in \mathbf{R}^T$ is the perturbation to the yield curve. We might impose constraints on the perturbations such as

$$\delta^{\min} \leq \delta \leq \delta^{\max}, \qquad \sum_{t=1}^{T} \delta_t^2 \leq \kappa, \qquad \sum_{t=1}^{T-1} (\delta_{t+1} - \delta_t)^2 \leq \omega, \tag{9}$$

where $\delta^{\min}, \delta^{\max}$, κ, and ω are given parameters, and the first inequalities are elementwise. The first constraint limits the perturbation in yield for any t; the second limits the mean square perturbation, and the third is a smoothness constraint, which limits the roughness of the yield curve perturbation. This is of course just an example; one could add many further constraints, such as insisting that the perturbed yield have nonnegative slope, is concave, or that the perturbation is plausible under a statistical model of short term changes in yields.

3.3.5 Constraints

We can add any convex constraints in our description of $\mathcal{U}$. For example, we might add the constraints (9) to a factor model, or confidence ellipsoid. As an example of a constraint

related to spreads, we can require that the spreads are nonincreasing as a function of the bond rating, *i.e.*, we always have $s_i \leq s_j$ if bond i has a higher rating than bond j. This is a set of linear inequalities on the vector of spreads s.

4 Robust portfolio construction

In the worst case analysis problem described in §3, the portfolio is given as h^{nom}. Here we consider the case where the portfolio is to be chosen. We denote the new portfolio as h, with h^{nom} denoting the nominal or current portfolio. Our goal is to choose h, which we do by minimizing an objective function, subject to some constraints.

4.1 Nominal portfolio construction problem

We first describe the nominal bond portfolio construction problem. We are given a nominal objective function $\phi : \mathbf{R}^n \to \mathbf{R}$ which is to be minimized. The nominal objective function might include tracking error against a benchmark, a risk term, and possibly a transaction cost term if the portfolio is to be constructed from the existing portfolio h^{nom} (see, *e.g.*, [BBD+17]). We will assume that the nominal objective function is convex.

We also have a set of portfolio constraints, which we denote as $h \in \mathcal{H}$, where $\mathcal{H} \subset \mathbf{R}^n_+$. The constraint set includes the long-only constraint $h \geq 0$, as well as a budget constraint, such as $p^T h = p^T h^{\mathrm{nom}}$, which states that the new portfolio has the same value as the original one. (This can be extended to take into account transaction costs if needed.) The constraint set $\mathcal{H}$ can include constraints on exposures to regions or sectors, average ratings, duration, a limit on risk, and so on. We will assume that $\mathcal{H}$ is convex. The nominal portfolio construction problem is

$$\begin{array}{ll} \text{minimize} & \phi(h) \\ \text{subject to} & h \in \mathcal{H}, \end{array}$$

with variable h. This is a convex optimization problem.

4.2 Robust portfolio construction problem

To obtain the robust portfolio construction problem we add one more penalty term to the nominal objective function, which penalizes the worst case change in value over the given uncertainty set $\mathcal{U}$.

The term, which we refer to as the *robustness penalty*, is $-\lambda\Delta^{\mathrm{wc}}(h)$, where $\lambda > 0$ is a parameter used to trade off the nominal objective ϕ and the worst case change in log portfolio value $-\Delta^{\mathrm{wc}}(h)$. Here we write the worst case change in log value with argument h, to show its dependence on h.

We arrive at the optimization problem

$$\begin{array}{ll} \text{minimize} & \phi(h) - \lambda\Delta^{\mathrm{wc}}(h) \\ \text{subject to} & h \in \mathcal{H}, \end{array} \tag{10}$$

with variable h. We refer to this as the *robust bond portfolio construction problem*. The objective is convex since ϕ is convex and Δ^{wc} is a concave function of h. This means that the robust bond portfolio construction problem is convex.

However, the robustness penalty term $-\Delta^{\text{wc}}(h)$ is not directly amenable to standard convex optimization, since it involves a minimization (over y and s) itself. We will address the question of how to tractably handle the robustness penalty term below using methods for convex-concave saddle point optimization.

A different framing of the robust bond portfolio construction problem is to minimize ϕ subject to a constraint on Δ^{wc}. This is readily handled, but we defer the discussion to §6.

4.3 Linearized robust portfolio construction problem

As in the worst case analysis problem we can use the linearized approximation of the worst case log value instead of the true log portfolio value, which gives the problem

$$\begin{array}{ll} \text{minimize} & \phi(h) - \lambda\hat{\Delta}^{\text{wc}}(h) \\ \text{subject to} & h \in \mathcal{H}, \end{array} \tag{11}$$

with variable h. Here $\hat{\Delta}^{\text{wc}}$ is the worst case change in log portfolio value predicted by the linearized approximation, *i.e.*, the optimal value of (8), as a function of h. We note that $\hat{\Delta}^{\text{wc}}$ is, like Δ^{wc}, a concave function of h.

4.4 Convex-concave saddle point formulation

We can write the robust portfolio construction problem (10) as

$$\underset{h\in\mathcal{H}}{\text{minimize}} \ \max_{(y,s)\in\mathcal{U}} \ \left(\phi(h) - \lambda\Delta(h,y,s)\right). \tag{12}$$

(Maximizing $-\lambda\Delta(h,y,s)$ over $(y,s) \in \mathcal{U}$ gives $-\lambda\Delta^{\text{wc}}(h)$.) The objective in (12) is convex in h and concave in (y,u), so this is a convex-concave saddle point problem. Replacing Δ with $\hat{\Delta}$ yields the saddle point version of the linearized robust portfolio construction problem.

Sion's minimax theorem [Sio58] tells us that if $\mathcal{H}$ is compact, when we reverse the order of the minimization and maximization we obtain the same value, which implies that there exists a saddle point $(h^\star, y^\star, s^\star)$, which satisfies

$$\phi(h^\star) - \lambda\Delta(h^\star,y,s) \le \phi(h^\star) - \lambda\Delta(h^\star,y^\star,s^\star) \le \phi(h) - \lambda\Delta(h,y^\star,s^\star)$$

for all $h \in \mathcal{H}$ and $(y,s) \in \mathcal{U}$. The left hand inequality shows that $\phi(h^\star) - \lambda\Delta(h^\star,y,s)$ is maximized over $(y,s) \in \mathcal{U}$ by $(y^\star,s^\star)$; the right hand inequality shows that $\phi(h) - \lambda\Delta(h,y^\star,s^\star)$ is minimized over $h \in \mathcal{H}$ by $h^\star$. It follows that $\phi(h^\star) - \lambda\Delta(h^\star,y^\star,s^\star)$ is the optimal value of the robust portfolio construction problem, $h^\star$ is an optimal portfolio, and $(y^\star,s^\star)$ is a worst case yield curve and spread.

5 Duality based saddle point method

The robust bond portfolio construction problem (12) is convex, but unfortunately not immediately representable in a DSL. In this section we use a well known trick to transform the problem to one that can be handled directly in a DSL. Using duality we will express Δ^{wc} as the *maximum* of a concave function over some variables that lie in a convex set. This method of transforming an inner minimization is not new; it has been used since the 1950s when Von Neumann proved the minimax theorem using strong duality in his work with Morgenstern on game theory [VNM53].

We now describe the dualization method for the case when $\mathcal{U}$ is a polyhedron of the form

$$\mathcal{U} = \{(y, s) \mid A(y, s) \leq b\},$$

with $A \in \mathbf{R}^{p\times(T+n)}$ and $b \in \mathbf{R}^p$, and the inequality is elementwise, but similar derivations can be carried out for other descriptions of $\mathcal{U}$.

5.1 Dual form of worst case change in log value

We assume Slater's condition hold, since any uncertainty set in practice will have a nonempty relative interior. From strong duality, it follows that

$$\Delta^{\text{wc}}(h) = \max_{\mu\geq 0,\, \nu} \; g(h, \mu, \nu), \tag{13}$$

where

$$g(h,\mu,\nu) = \begin{cases} -\log(p^T h) - \mu^T b - \sum_{i=1}^{n}\sum_{t=1}^{T} \zeta(c_{i,t}h_i, \frac{\nu_{i,t}}{t}) & \text{if } A^T\mu - F^T\nu = 0,\ \mathbf{1}^T\nu = 1,\ \nu \geq 0 \\ -\infty & \text{otherwise,} \end{cases} \tag{14}$$

with

$$\zeta(x,t) = -t\log(x/t) = t\log(t/x) = t\log(t) - t\log(x),$$

which is the relative entropy. Here $\mu \in \mathbf{R}^p$ and $\nu \in \mathbf{R}^{nT}$ are variables, and the matrix $F \in \mathbf{R}^{nT\times(T+n)}$ will be defined below. The relative entropy is convex (see, *e.g.*, [BV04, §3.2.6]), which implies that g is jointly concave in (h, μ, ν). A full derivation of (13) is given in §C.

5.2 Single optimization problem form

Using (14) we can write the robust bond portfolio construction problem as a single optimization problem compatible with DSLs, with variables h, μ, and ν:

$$\begin{array}{ll} \text{minimize} & \phi(h) + \lambda\left(\mu^T b + \sum_{i=1}^{n}\sum_{t=1}^{T} \zeta(c_{i,t}, \frac{\nu_{i,t}}{t}) + \log(p^T h)\right) \\ \text{subject to} & \mu \geq 0, \quad A^T\mu - F^T\nu = 0, \quad \nu \geq 0, \quad \mathbf{1}^T\nu = 1 \\ & h \in \mathcal{H}. \end{array}$$

This is a convex optimization problem because the objective is convex and the constraints are linear equality and inequalities. This form is tractable for DSLs.

5.3 Automated dualization via conic representation

While for many uncertainty sets explicit dual forms can be derived by hand, this process can be tedious and error-prone. In recent work, Juditsky and Nemirovski [JN21] present a method for transforming general structured convex-concave saddle point problems to a single minimization problem via a generalized conic representation of convex-concave functions. Similar to disciplined convex programming (DCP) [GBY06], the method introduces some basic atoms of known convex-concave saddle functions, as well as a set of rules for combining them to form composite problems, making it extremely general.

Juditsky and Nemirovski define a general notion of conic representability for convex-concave saddle point problems

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \psi(x, y). \tag{15}$$

If the convex-concave ψ can be written in this general form, then (15) can be written as a single minimization problem, with variables comprising x together with additional variables. See [JN21] for details on conic representability, which is beyond the scope of this paper. The set of conically representable convex-concave functions is large and includes generalized inner products of the form $F(x)^T G(y)$ where F is elementwise convex and nonnegative and $G(y)$ is elementwise concave and nonnegative, and special atoms like weighted log-sum-exp, $\log(\sum_i x_i \exp(y_i))$, which appears in the robust bond portfolio construction problem.

The authors have developed a package for disciplined saddle point programming called DSP, described in [SLB23]. DSP automates the dualization, following the ideas of Juditsky and Nemirovski, and allows users to easily express and then solve convex-concave saddle point problems, including as a special case the robust bond portfolio construction problem. Roughly speaking DSP hides the complexity of dualization from the user, who expresses the saddle point problem using a natural description. We refer the reader to [SLB23] for a (much) more detailed description of DSP and its associated domain specific language.

5.4 DSP specification

To illustrate the use of DSP for robust bond portfolio construction, we give below the code needed to formulate and solve it. We assume that several objects have already been defined: `C` is the cash flow matrix, `H` and `U` are DCP compliant descriptions of the portfolio and yield/spread uncertainty set, `phi` is a DCP compliant convex nominal objective function, and `lamb` is a positive parameter.

```
1  import cvxpy as cp
2  import dsp
3
4  y = cp.Variable(T)
5  s = cp.Variable(n)
6  h = cp.Variable(n, nonneg=True)
7
8  exponents = []
9  weights = []
10 for i in range(n):
11     for t in range(T):
12         if C[i, t] > 0:
13             exponents.append(-(t + 1) * (y[t] + s[i]))
14             weights.append(h[i] * C[i, t])
15
16 Delta = dsp.weighted_log_sum_exp(cp.hstack(exponents), cp.hstack(weights))
17
18 obj = dsp.MinimizeMaximize(phi - lamb * Delta)
19
20 constraints = H + U
21
22 saddle_problem = dsp.SaddleProblem(obj, constraints)
23 saddle_problem.solve()
```

In lines 8–16 we construct an expression for Δ, where in line 16 we use the convex-concave DSP atomic function `weighted_log_sum_exp`. In line 20 the addition symbol concatenates `H` and `U`, which are lists of CVXPY constraints that define $\mathcal{H}$ and $\mathcal{U}$, respectively. In line 22 we construct the saddle point problem, and in line 23 we solve it. The optimal portfolio can then be found in `h.value`, and the worst case yield and spreads in `y.value` and `s.value`, respectively.

6 Variations and extensions

Periodically compounded growth. To handle periodically compounded interest, simply observe that in this case,

$$y_t = p_t^{-1/t} - 1, \quad t = 1, \ldots, T,$$

and the portfolio value is $f(y) = \sum_{t=1}^{T} c_t(1 + y_t)^{-t}$. This is a convex function of y because $c_t \geq 0$ and x^a is convex for any $a < 0$ and nonnegative argument.

Multiple reference yield curves. We can immediately extend to the case where each bond has its own reference yield curve $y^i \in \mathbf{R}^T$ for $i = 1, \ldots, n$. This effectively means the

spread can be time varying for each bond. All the convexity properties are preserved; in fact this just corresponds to an unconstrained z variable in §C.

Constrained form. It is natural and interpretable to pose (10) in constrained form, that is,

$$\begin{array}{ll} \text{minimize} & \phi(h) \\ \text{subject to} & \Delta^{\text{wc}}(h) \geq -\eta \\ & h \in \mathcal{H}, \end{array}$$

for some $\eta > 0$. For example, one could consider minimizing the tracking error to a reference bond portfolio, subject to the constraint that the worst case change in bond portfolio value does not exceed a given tolerance. Using the conic representability method in §5.3, we can immediately include $\Delta^{\text{wc}}(h) \geq -\eta$ as a DCP compliant constraint. See our recent paper on DSP [SLB23] for details.

7 Examples

In this section we illustrate worst case analysis and robust portfolio construction with numerical examples. The examples all use the data constructed as described below. We emphasize that we consider a simplified small problem only so the results are interpretable, and not due to any limitation in the algorithms used to carry out worst case analysis or robust portfolio construction, which readily scale to much larger problems.

The full source code and data to re-create the results shown here is available online at

`https://github.com/cvxgrp/robust_bond_portfolio`.

7.1 Data

We work with a simpler and smaller universe of bonds that is derived from, and captures the main elements of, a real portfolio.

Bond universe. We start with the bonds in the iShares Global Aggregate Bond UCITS ETF (AGGG), which tracks the Bloomberg Global Aggregate Bond Index [Blone], a market-cap weighted index of global investment grade bonds. As of 2022-09-12, AGGG held 10,564 bonds, which we partition into 20 groups by rating and maturity. We consider the four ratings AAA, AA, A, and BBB, and five buckets of maturities, 0–3, 3–5, 5–10, 10–20, and 20–30 years. From each of the 20 rating-maturity groups, we select the bond in AGGG with the highest market capitalization. These 20 bonds constitute the universe we consider. They are listed in table 1, with data as of 2022-09-12.

For each bond we construct its cash flow based on the coupon rate, maturity, and frequency of coupon payments, assuming the cash flows are paid at the end of each period. All bonds in the universe distribute either semi-annual or annual coupons, so we use a period length of six months. The longest term to maturity in our universe is 30 years, so we take

Ticker	Rating	Term to maturity	Coupon rate	Distribution	Price
T 2 5/8 03/31/25	AAA	2.46	2.625	semi-annual	101.86
T 1 1/4 12/31/26	AAA	4.21	1.250	semi-annual	92.63
T 0 5/8 12/31/27	AAA	5.21	0.625	semi-annual	85.43
T 3 1/4 05/15/42	AAA	19.58	3.250	semi-annual	128.96
T 3 08/15/52	AAA	29.84	3.000	semi-annual	130.55
FHLMC 0 3/8 09/23/25	AA	2.94	0.375	semi-annual	89.13
NSWTC 3 05/20/27	AA	4.60	3.000	semi-annual	106.66
WATC 3 1/4 07/20/28	AA	5.77	3.250	semi-annual	110.63
NSWTC 2 1/4 05/07/41	AA	18.56	2.250	semi-annual	99.64
BGB 3 3/4 06/22/45	AA	22.69	3.750	annual	88.60
JGB 0.4 09/20/25 #340	A	2.93	0.400	semi-annual	88.35
JGB 0.1 09/20/27 #348	A	4.93	0.100	semi-annual	79.62
JGB 0.1 06/20/31 #363	A	8.68	0.100	semi-annual	67.39
JGB 1 12/20/35 #155	A	13.18	1.000	semi-annual	72.75
JGB 1.7 09/20/44 #44	A	21.94	1.700	semi-annual	80.12
SPGB 3.8 04/30/24	BBB	1.54	3.800	annual	96.39
SPGB 2.15 10/31/25	BBB	3.05	2.150	annual	90.01
SPGB 1.45 10/31/27	BBB	5.05	1.450	annual	81.71
SPGB 2.35 07/30/33	BBB	10.79	2.350	annual	75.07
SPGB 2.9 10/31/46	BBB	24.05	2.900	annual	66.14

Table 1: The 20 bond universe used for numerical examples. The prices are computed as of 2022-09-12.

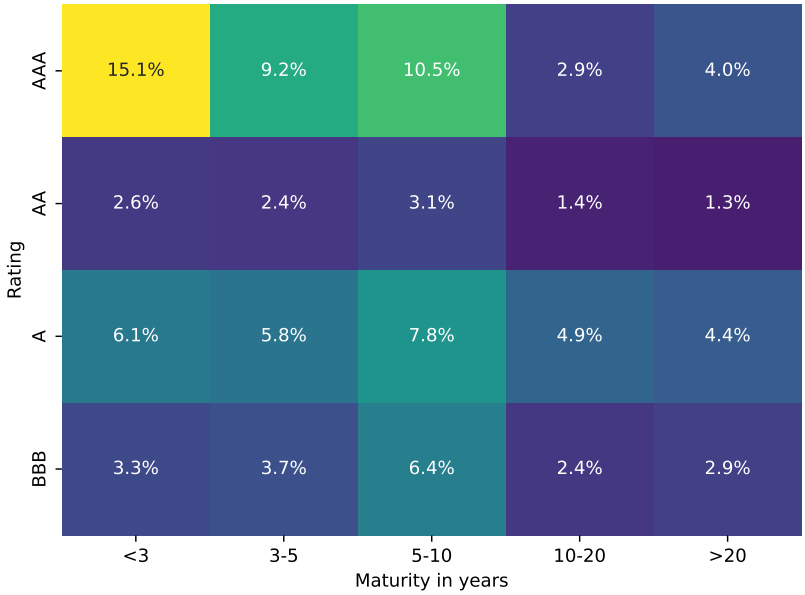


Figure 1: The nominal portfolio weights.

$T = 60$. We assemble the cash flows into a matrix $C \in \mathbf{R}^{20\times 60}$. We price the bonds according to (1), using US treasury yield curve data and spreads that depend on the rating, with data as of 2022-09-12, as listed in table 1.

Nominal portfolio. Our nominal portfolio h^{nom} puts weight on each of our 20 bonds equal to the total weight of all bonds in the corresponding rating-maturity group in AGGG. The weights are shown in figure 1.

7.2 Uncertainty sets

We create uncertainty sets using historical daily yield curves and spreads. Data for the yield curve is obtained from the US Treasury [U.Sne], and the spreads are obtained from the Federal Reserve Bank of St. Louis [Fedne], spanning the period 1997-01-02 to 2022-09-12, for a total of 5,430 observations. As our period length is 6 months, we only consider the 9 key rate durations of 6 months, as well as 1, 2, 3, 5, 7, 10, 20, and 30 years. Joining the 9 rates with the 4 ratings, our total data is represented as a $5,430 \times 13$ matrix. The mean value of each column, denoted μ^{hist}, as well as the nominal, *i.e.*, most recent, yields and spreads are shown in figure 2. We use simple linear interpolation to obtain the full yield curve $y \in \mathbf{R}^{60}$.

We model the uncertainty set as a (degenerate) ellipsoid. We compute the empirical mean and covariance of the historical data, denoted μ^{hist} and Σ^{hist}, respectively. We define

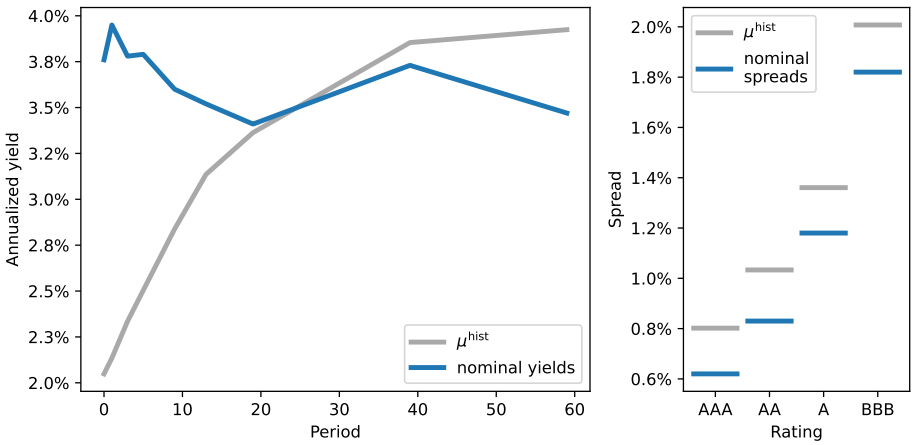


Figure 2: Historical mean values of (annualized) yield and spreads. The nominal yield and spreads, date 2022-9-12, are also shown.

$Z \in \mathbf{R}^{80\times 13}$ as the matrix that maps the key rates and ratings, $(y^{\text{k}}, s^r) \in \mathbf{R}^{13}$ to the yields and spreads, $(y, s) \in \mathbf{R}^{80}$, *i.e.*,

$$(y, s) = Z(y^{\text{k}}, s^r).$$

The matrix Z encodes linear interpolation between key rates; other linear mappings like spline interpolation could also be used.

Our uncertainty set is then defined in terms of key rates and ratings,

$$\mathcal{U} = \{Z(y^{\text{k}}, s^r) \mid \left((y^{\text{k}}, s^{\text{r}}) - \mu^{\text{hist}}\right)^T \left(\Sigma^{\text{hist}}\right)^{-1} \left((y^{\text{k}}, s^{\text{r}}) - \mu^{\text{hist}}\right) \leq F^{-1}(1-\alpha)\},$$

where F is the CDF of a χ^2 random variable with 13 degrees of freedom, and $\alpha \in (0, 1)$ is a confidence level. This uncertainty set is a degenerate ellipsoid, with affine dimension 13.

To represent a modest uncertainty set, we use confidence levels 50%. We also consider a more extreme uncertainty set, with confidence level 99%. These two uncertainty sets are meant only to illustrate our method; in practice, we would likely create uncertainty sets that change over time, and are based on more recent yield curves and spreads, as opposed to a long history of yields and spreads.

7.3 Worst case analysis

Table 2 shows worst case change in portfolio value for the 50% and 99% confidence levels, each using both the exact and linearized methods. The values are given as relative change in portfolio value, *i.e.*, we have already converted from log returns. For $\alpha = 50\%$, the exact method gives a worst case change in portfolio value of around -29%, and the linearized method predicts a change in portfolio value around four percentage points lower. For $\alpha =$

α	Worst case	Linearized
50%	-29.34%	-33.43%
99%	-39.64%	-45.95%

Table 2: Worst case change in portfolio value, for two uncertainty sets, using both the exact and linearized methods.

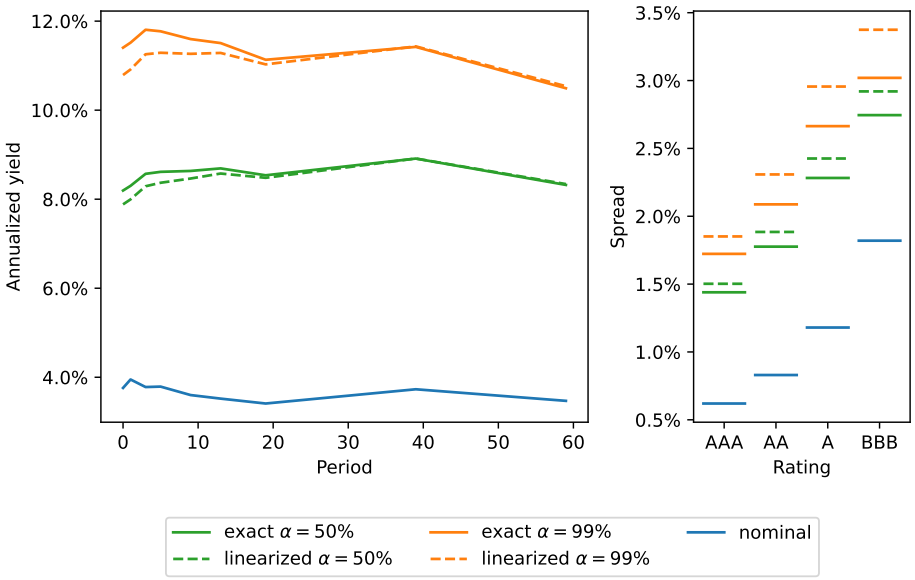


Figure 3: Worst (annualized) yield curve and spreads for the two uncertainty sets, using the exact and linearized methods.

99% the approximation error of the linearized method is greater, more than six percentage points. We can see that in both cases the linearized method is conservative, *i.e.*, predicts a change in value that is lower than the exact method.

Figure 3 shows the corresponding annualized worst case yields and spreads. For the modest uncertainty case, the linearized and exact methods produce similar results, while for the extreme case, the linearized method deviates more for shorter yields and across all spreads.

7.4 Robust portfolio construction

We now consider the case where instead of holding the nominal portfolio h^{nom} exactly, we try to track it, with a penalty term on the worst case change in portfolio value, as described in

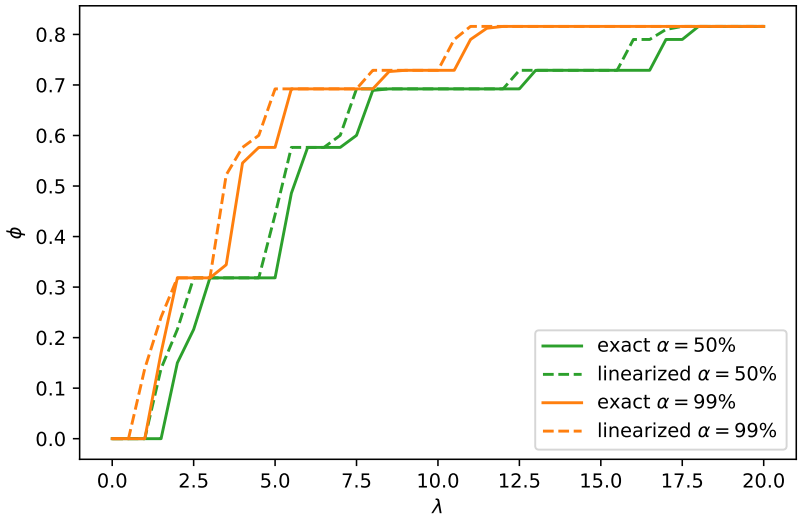


Figure 4: Turnover distance to the nominal portfolio, for two uncertainty sets, using both the exact and linearized methods.

§4.2. Our nominal objective is the turnover distance between our holdings and the nominal portfolio, given by

$$\phi(h) = (1/2)\|h - h^{\text{nom}}\|_1.$$

Figure 4 shows the turnover distance for both uncertainty sets and both the exact and linearized methods. For small values of λ, the nominal portfolio is held exactly. As expected, the turnover distance increases with λ, but more rapidly so for the extreme uncertainty set. We also see that the linearized method gives very similar results. For some values of λ, the resulting portfolio obtains the same turnover distance as the exact method, but for other values it is slightly worse due to the conservative linear approximation.

Figure 5 shows the resulting portfolio holdings for the two uncertainty sets across varying values of λ. For $\lambda = 1$, the weights are exactly the nominal weights. For $\lambda = 5$, the weights have shifted to shorter maturities, reducing the worst case change in portfolio. However, this shift is more pronounced for the extreme uncertainty set, where the weights are only allocated up to the 3–5 year bucket, whereas for the modest uncertainty set the weights include bonds up to the 5–10 year bucket. A similar observation can be made for $\lambda = 15$, where the optimization under the modest uncertainty set still allocates in bonds across all ratings in the bucket containing bonds with less than 3 years to maturity. In contrast, under the extreme uncertainty set the weights are only allocated to two bonds in this bucket. Specifically, the highest weight is assigned to the bond with BBB rating, and a smaller weight to the AAA bond. This is explained by the much shorter maturity of the BBB bond (see table 1), which outweighs the lower risk due to the higher rating of the AAA bond.

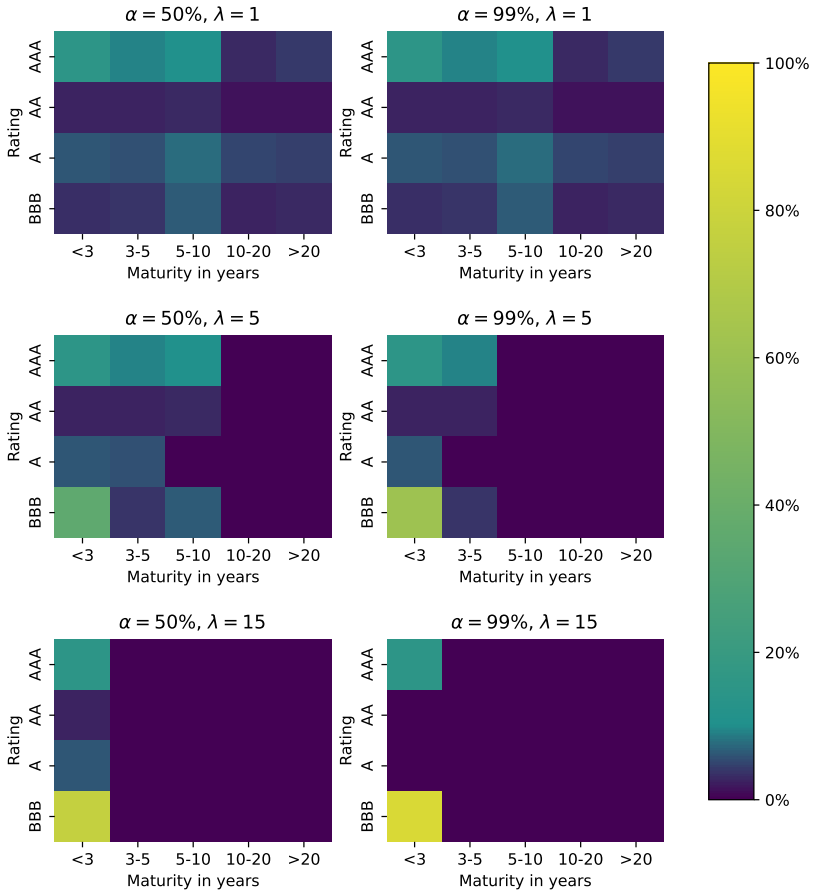


Figure 5: Portfolio holdings, for both uncertainty sets for $\lambda \in \{1, 5, 15\}$.

Indeed, when manually setting the maturities of these bonds to the same value, we find that weights would be assigned to the AAA bond for large values of λ instead.

8 Conclusions

We have observed that the greatest decrease in value of a long-only bond portfolio, over a given convex set of possible yield curves and spreads, can be found exactly by solving a tractable convex optimization problem that can be expressed in just a few lines using a DSL. Current practice is to estimate the worst case decrease in value using key rate durations, which is equivalent to finding the worst case change in portfolio value using a linearized approximation of the portfolio value. We show that this estimate is always conservative. Numerical examples show that it is a good approximation of the actual worst case value for modest changes in the yield curve and spread, but less good for large uncertainty sets.

We also show that the problem of constructing a long-only bond portfolio which includes the worst case value over an uncertainty set in its objective or constraints can be tractably solved by formulating it as a convex-concave saddle point problem. Such problems can also be specified in just a few lines of a DSL.

Acknowledgements

We thank Dr. Baruch Gliksberg for his thoughtful comments and suggestions.

This research was partially supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR, and by ONR N000142212121. P. Schiele is supported by a fellowship within the IFI program of the German Academic Exchange Service (DAAD).

References

[BBD+17] S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth. Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76, 2017.

[Blone] Bloomberg. Bloomberg fixed income indices fact sheets and publications, Nov. 16, 2022 [Online].

[BS16] K. Bredies and H. Sun. Accelerated Douglas-Rachford methods for the solution of convex-concave saddle-point problems. *arXiv preprint arXiv:1604.06282*, 2016.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CP05] J. Cochrane and M. Piazzesi. Bond risk premia. *American Economic Review*, 95(1):138–160, 2005.

[DB16] S. Diamond and S. Boyd. CVXPY: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

[EGBG09] E. Elton, M. Gruber, S. Brown, and W. Goetzmann. *Modern Portfolio Theory and Investment Analysis*. John Wiley & Sons, 2009.

[Fedne] Federal Reserve Bank of St. Louis. Data: [BAMLC0A1CAAA, BAMLC0A2CAA, BAMLC0A3CA, BAMLC0A4CBBB], Nov. 16, 2022 [Online].

[FM12] F. Fabozzi and S. Mann. *The Handbook of Fixed Income Securities*. McGraw-Hill Education, 2012.

[FNB20] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 94(14):1–34, 2020.

[FPY22] D. Filipović, M. Pelger, and Y. Ye. Stripping the discount curve-a robust machine learning approach. *Swiss Finance Institute Research Paper*, (22-24), 2022.

[GB14] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. 2014.

[GBY06] M. Grant, S. Boyd, and Y. Ye. *Disciplined Convex Programming*, pages 155–210. Springer US, Boston, MA, 2006.

[JN21] A. Juditsky and A. Nemirovski. On well-structured convex–concave saddle point problems and variational inequalities with monotone operators. *Optimization Methods and Software*, 0(0):1–36, 2021.

[KK06] O. Korn and C. Koziol. Bond portfolio optimization: A risk-return approach. *The Journal of Fixed Income*, 15(4):48–60, 2006.

[KKF14] J. Kim, W. Kim, and F. Fabozzi. Recent developments in robust portfolios with a worst-case approach. *Journal of Optimization Theory and Applications*, 161(1):103–121, 2014.

[LS91] R. Litterman and J. Scheinkman. Common factors affecting bond returns. *The Journal of Fixed Income*, 1(1):54–61, 1991.

[Mar52] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7:77–91, 1952.

[Nem04] A. Nemirovski. Prox-method with rate of convergence $o(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[NY78] A. Nemirovski and D. Yudin. Cesari convergence of the gradient method of approximating saddle points of convex-concave functions. In *Doklady Akademii Nauk*, volume 239, pages 1056–1059. Russian Academy of Sciences, 1978.

[NY83] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley-Interscience, 1983.

[Puh08] M. Puhle. *Bond Portfolio Optimization*, volume 605. Springer Science & Business Media, 2008.

[Sio58] M. Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1):171–176, 1958.

[SLB23] P. Schiele, E. Luxenberg, and S Boyd. Disciplined saddle point programming. *arXiv preprint arXiv:2301.13427*, 2023.

[TK04] R. Tütüncü and M. Koenig. Robust asset allocation. *Annals of Operations Research*, 132(1):157–187, 2004.

[UMZ+14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. *SC14 Workshop on High Performance Technical Computing in Dynamic Languages*, 2014.

[U.Sne] U.S. Department of the Treasury. Daily treasury par yield curve rates, Nov. 16, 2022 [Online].

[VN28] J. Von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[VNM53] J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

A Worst case analysis CVXPY code

```
import numpy as np
import cvxpy as cp

y = cp.Variable(T)
s = cp.Variable(n)

V = h @ p

exponents = []
for i in range(n):
    for t_idx in range(T):
        t = t_idx + 1  # account for 0-indexing
        w_it = h[i] * C[i,t_idx]
        if w_it > 0:
            exponents.append(-t * (y[t_idx] + s[i]) + np.log(w_it))

Delta = cp.log_sum_exp(cp.hstack(exponents)) - np.log(V)
obj = cp.Minimize(Delta)
prob = cp.Problem(obj, [A @ cp.hstack([y,s]) <= b])
prob.solve()
```

B Explicit dual portfolio construction CVXPY code

```
import numpy as np
import cvxpy as cp

F_1 = np.tile(np.eye(T), (n, 1))
F_2 = np.repeat(np.eye(n), repeats=T, axis=0)
F = np.hstack([F_1, F_2])

lam = cp.Variable(len(b), nonneg=True)
nu = cp.Variable(n * T, nonneg=True)

h = cp.Variable(n, nonneg=True)

B = 1

term = 0
for i in range(n):
    for t in range(1, T):
        nu_i_t = nu[i * T + t]
        term -= cp.rel_entr(C[i, t] * h[i], nu_i_t / t)

obj = cp.Maximize(-lam @ b + term - np.log(B))
constraints = [
    A.T @ lam == F.T @ nu,
    cp.sum(nu) == 1,
    p @ h == B,
]
prob = cp.Problem(obj, constraints)
prob.solve()
```

C Derivation of dual form

We now derive the dual form of the worst case portfolio construction problem for the case where the uncertainty set in polyhedral. We note that the worst case log change in portfolio value for a fixed h, $\Delta^{\mathrm{wc}}(h)$ is given by the optimal value of the optimization problem

$$\begin{array}{ll} \text{minimize} & \log\left(\sum_{i=1}^{n}\sum_{t=1}^{T} h_i c_{i,t}\exp(-t(y_t+s_i))\right) - \log(p^T h) \\ \text{subject to} & A(y,s) \preceq b. \end{array}$$

We have written this problem with (y, s) explicitly, instead of with x, to emphasize the objective's dependence on each component. We note that due to our budget constraint, $\log(V(y, s)) = \log(p^T h)$.

In order to obtain a closed form dual, we introduce a new variable $z \in \mathbf{R}^{nT}$, where we think of $z_{i,t}$ as corresponding to $y_t + s_i$. This is a very general formulation which allows each bond to be associated with its own yield curve $y_i \in \mathbf{R}^T$, with $z_{i,t}$ corresponding to the t'th entry of the i'th bond's yield curve. Since we model each bond as having its own yield curve, this formulation generalizes the earlier treatment with yields and spreads handled separately. We can recover the original structure with the linear constraints

$$z_{i,t} = y_t + s_i, \qquad t = 1, \ldots, T, \quad i = 1, \ldots, n, \tag{16}$$

which are representable as $z = Fx$ for an appropriate $F \in \mathbf{R}^{nT\times(T+n)}$. As such, the problem is equivalent to

$$\begin{array}{ll} \text{minimize} & \log\left(\sum_{i=1}^{n}\sum_{t=1}^{T} h_i c_{i,t}\exp(-t z_{i,t})\right) - \log(p^T h) \\ \text{subject to} & Ax \preceq b, \quad z = Fx. \end{array} \tag{17}$$

Strong duality tells us that $\Delta^{\mathrm{wc}}(h)$ is equal to the optimal value of the dual problem of (17) [BV04, §5.2].

Dual problem. We derive the dual of the problem

$$\begin{array}{ll} \text{minimize} & \log\left(\sum_{i=1}^{n}\sum_{t=1}^{T} h_i c_{i,t}\exp(-t z_{i,t})\right) - \log(p^T h) \\ \text{subject to} & Ax \preceq b, \quad z = Fx. \end{array} \tag{18}$$

First, with f the log-sum-exp function $f(x) = \log\left(\sum \exp x_i\right)$, we observe that our problem can be rewritten as

$$\begin{array}{ll} \text{minimize} & f(Cz+d) - \log(p^T h) \\ \text{subject to} & Ax \preceq b, \quad z = Fx. \end{array}$$

We define C to be the diagonal matrix with $C_{i,t} = -t$, where we are using unwound vectorized indexing for z, and $d \in \mathbf{R}^{nT}$ to be the vector with $d_{i,t} = \log(c_{i,t}h_i)$. Then, the Lagrangian is given by

$$L(z, x, \mu, \nu) = f(Cz + d) + \mu^T(Ax - c) + \nu^T(z - Fx) - \log(p^T h).$$

The Lagrange dual function is given by

$$g(\mu,\nu) = \inf_{z,x} L(z,x,\mu,\nu) = \inf_z \left(f(Cz+d) + \nu^T z\right) + \inf_x \left(\mu^T A x - \nu^T F x\right) - \mu^T c - \log(p^T h).$$

The second term is equal to $-\infty$ unless $A^T\mu + F^T\nu = 0$, so this condition will implicitly restrict the domain of g. Now, note that with $g(z) = f(Cz+d)$, the first term can be rewritten as

$$\begin{aligned}\inf_z \left(f(Cz+d) + \nu^T z\right) &= \inf_z \left(g(z) + \nu^T z\right) \\ &= -\max_z \left(-\nu^T z - g(z)\right) \\ &= -g^*(-\nu),\end{aligned}$$

where $g^*(y) = \max yy^T z - g(z)$ is the conjugate of g. [BV04, §3.3.1].

We now use two facts from [BV04, §3.3.2]. First, in general the conjugate of the linear precomposition $\phi(z) = \rho(Cz+d)$ can be written in terms of the conjugate of ρ as $\phi^*(y) = \rho^*(C^{-T}y) - d^T C^{-T} y$. Second, the dual of the log-sum-exp function f is

$$f^*(y) = \begin{cases} \sum_i y_i \log(y_i) & \text{if } y \geq 0, \quad \mathbf{1}^T y = 1 \\ \infty & \text{otherwise.} \end{cases}$$

Combining these two facts, and expanding terms, we find that

$$g(\mu,\nu) = \begin{cases} -\log(p^T h) - \mu^T b - \sum_{i=1}^n \sum_{t=1}^T \zeta(c_{i,t}h_i, \frac{\nu_{i,t}}{t}) & \text{if } A^T\mu - F^T\nu = 0,\ \mathbf{1}^T\nu = 1,\ \nu \geq 0 \\ -\infty & \text{otherwise,} \end{cases}$$

with

$$\zeta(x,t) = -t\log(x/t) = t\log(t/x) = t\log(t) - t\log(x).$$

Thus the robust bond portfolio problem can be written as

$$\begin{array}{lll} \text{minimize} & \phi(h) - \lambda \max\limits_{\mu,\nu} & g(\mu,\nu) \\ \text{subject to} & \mu \geq 0, \quad A^T\mu - F^T\nu = 0, \quad \mathbf{1}^T\nu = 1, \quad \nu \geq 0 \\ & h \in \mathcal{H}, \end{array}$$

where we have moved the implicit constraints in the definition of g to explicit constraints in the optimization problem. Note this equivalent optimization problem has new variables μ and ν. By using that $-\lambda \max_{\mu,\nu} g(\mu,\nu) = \min_{\mu,\nu} -\lambda g(\mu,\nu)$ and collecting the minimization over h, μ, and ν, we obtain the form in §5.2.

Part V.

Portfolio Optimization with Cumulative Prospect Theory Utility via Convex Optimization

This chapter is a reprint of:

Luxenberg, E., Schiele, P., Boyd, S. (2022). Portfolio Optimization with Cumulative Prospect Theory Utility via Convex Optimization. *Available at https://arxiv.org/abs/2209.03461*. Published in *Computational Economics*, *available at https://link.springer.com/article/10.1007/s10614-024-10556-x*.

The included version is the third revision of the essay.

Copyright information:

Author contributions:

This research project was initiated by Eric Luxenberg, who recognized that the cumulative prospect theory utility exhibits convexity properties that lend it to more efficient optimization methods. Philipp Schiele and Eric Luxenberg were responsible for the initial draft and software development, with Philipp Schiele independently deriving an epigraph representation of the weighted-ordered-sum function and implementing it in CVXPY, as well as deriving a specification of the convex term in the convex-concave procedure amenable to specification via disciplined convex programming. Subsequent revisions and supervision were provided by Stephen Boyd.

Supplementary material available at: https://github.com/cvxgrp/cptopt

Portfolio Optimization with Cumulative Prospect Theory Utility via Convex Optimization

Eric Luxenberg*[1], Philipp Schiele*[2], and Stephen Boyd[1]

[1]Department of Electrical Engineering, Stanford University
[2]Department of Statistics, Ludwig-Maximilians-Universität München

January 11, 2024

Abstract

We consider the problem of choosing a portfolio that maximizes the cumulative prospect theory (CPT) utility on an empirical distribution of asset returns. We show that while CPT utility is not a concave function of the portfolio weights, it can be expressed as a difference of two functions. The first term is the composition of a convex function with concave arguments and the second term a composition of a convex function with convex arguments. This structure allows us to derive a global lower bound, or minorant, on the CPT utility, which we can use in a minorization-maximization (MM) algorithm for maximizing CPT utility. We further show that the problem is amenable to a simple convex-concave (CC) procedure which iteratively maximizes a local approximation. Both of these methods can handle small and medium size problems, and complex (but convex) portfolio constraints. We also describe a simpler method that scales to larger problems, but handles only simple portfolio constraints.

*Equal contribution.

Contents

1 Introduction

1.1 Cumulative prospect theory

Analysis of decision-making under uncertainty has long been dominated by von Neumann-Morgenstern (VNM) utility maximization [vMR44], which takes rational behavior as a fundamental assumption. Kahneman and Tversky [KT79] observed that the VNM theory fails to explain actual human decision-making behavior in many settings. The subsequently introduced prospect theory (PT) formalizes loss aversion and the overweighting of small probability events, which are inconsistent with VNM utility maximization. To overcome a violation of first-order stochastic dominance in prospect theory, cumulative prospect theory (CPT) was introduced [TK92], which replaces probabilities of outcomes with their rank-dependent cumulative probability distribution. This change leads to an overweighting of extreme low-probability outcomes, instead of all low-probability outcomes. Maximizing CPT utility yields more realistic predictions of actual human decision-making behavior than maximizing VNM utility.

1.2 Portfolio optimization

Our focus is portfolio optimization, *i.e.*, choosing a mix of investments in a set of assets. One approach is based on VNM utility, where the expected value of a concave increasing utility function (which is also concave) is maximized subject to the constraints on the portfolio. Another approach, introduced by Markowitz [Mar52], poses the problem as a bi-criterion optimization problem, with the goal of trading off the maximization of expected return with minimization of risk, taken to be the variance of the portfolio return. The standard approach is to combine the return and risk, scaled by a risk aversion factor, into a risk-adjusted return, and maximize this concave quadratic objective subject to the constraints. For this reason Markowitz portfolio optimization is also referred to as mean-variance (MV) portfolio optimization. These two approaches are not the same, since the MV utility function is not increasing, but they are closely related. For example, with a Gaussian asset return model and exponential utility, VNM portfolio optimization is the same as MV portfolio optimization [Mer69]. In other cases, MV portfolio optimization was shown to be approximately optimal for other forms of utility functions [LM79].

One advantage of the MV formulation is that the objective can be expressed explicitly as a quadratic function, without an expectation over the asset returns. (The MV objective is the expected value of a function of the return, but one with a simple analytical expression.) This enables it to be solved analytically for special cases [GK99], and very efficiently using numerical methods for convex optimization when the constraints are convex [BV04]. Leveraging convex optimization, many extensions were developed, such as the inclusion of transaction and holding costs, or multi-period optimization [BBD+17],

VNM portfolio optimization generally uses sample based stochastic convex optimization [SDR21]. Here we use samples of the asset returns, which can be historical or generated from a stochastic model of asset returns (presumably fit to historical data). While these

methods solve the problem globally, they are substantially slower than MV methods for similar problems.

We now describe these portfolio optimization methods in more detail. We consider a set of n assets. A portfolio is characterized by its asset weights $w = (w_1, \ldots, w_n)$, where w_i is the fraction of the total portfolio value (assumed to be positive) invested in asset i, with $w_i < 0$ denoting a short position. The goal in portfolio optimization is to choose w. The general portfolio optimization problem is

$$\begin{array}{ll} \text{maximize} & U(w) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array} \tag{1}$$

with variable $w \in \mathbf{R}^n$. Here $U : \mathbf{R}^n \to \mathbf{R}$ is a utility function, $\mathcal{W} \subseteq \mathbf{R}^n$ is the set of allowable portfolio weights, and $\mathbf{1}$ is the vector with all entries one. The data in this problem are the utility function U and the portfolio constraint set $\mathcal{W}$, which we assume is convex.

When U is a concave function the portfolio optimization problem (1) is convex, and so readily solved globally [BV04]. When U is not concave, the problem is not convex, and in general difficult to solve globally. In this case, we typically resort to heuristic or local methods, which attempt to solve (1), but cannot guarantee that the globally optimal portfolio is found.

MV portfolio optimization uses the concave quadratic utility function

$$U^{\text{mv}}(w) = w^T \mu - \gamma w^T \Sigma w,$$

the risk-adjusted expected return, where μ is the expected asset return, Σ is the covariance matrix of the asset returns, and $\gamma > 0$ is the risk aversion parameter, used to control the trade-off of the mean and variance of the portfolio return. This yields a convex optimization problem that is efficiently solved.

VNM utility maximization uses a utility function of the form

$$U^{\text{vnm}}(w) = \mathbf{E}\, u(r^T w),$$

where r is the random asset return vector, and $u : \mathbf{R} \to \mathbf{R}$ is a concave increasing utility function. Since expectation preserves concavity, U is a concave function of w and this too leads to a convex portfolio optimization problem. In a few cases, the expectation can be worked out analytically, but in most cases one substitutes a sample or empirical average for the expected value, leading to the approximation

$$U^{\text{vnm}}(w) \approx \frac{1}{N} \sum_{i=1}^{N} u(r_i^T w),$$

where $r_1, \ldots, r_N$ are samples of returns. Maximizing this approximation results in a convex portfolio optimization problem.

1.3 This paper

We consider portfolio optimization under the CPT utility, which we denote U^{cpt}, defined later in §2. It is well known that CPT utility is not a concave function, so the problem of choosing portfolio weights so as to maximize it is not a convex optimization problem, as VNM utility maximization and MV portfolio optimization are. This makes it a challenge to carry out CPT utility maximization in practice.

While CPT utility is not concave, we will show that it does have some convexity structure. Specifically, it is the composition of a convex increasing function of concave functions for positive returns, and the composition of a concave increasing function of convex functions for negative returns. This observation allows us to construct a concave lower bound, or minorant, for the CPT utility, and leads immediately to a simple algorithm for maximizing it by repeatedly maximizing the constructed minorant (which is a convex problem, and thus readily solved). This simple minorization-maximization (MM) method leads to a local maximum of the CPT utility [LB15].

Our MM method scales to medium size problems, with perhaps tens of assets and hundreds of return samples. For larger problems, we give two other algorithms. One algorithm uses a simpler optimization of a minorant to the approximation given by fixing the probability weights that arise in CPT in each step, again relying on iterations that involve solving convex optimization problems. As a result, this method can handle complex portfolio constraints, as long as they are convex. The second additional algorithm scales to very large problems, but handles only simple portfolio constraints. It relies on modern frameworks for automatic differentiation and first-order optimization methods.

Open-source Python implementations of all three methods can be found in the code repository `https://github.com/cvxgrp/cptopt`.

We do not address questions such as whether or when one *should* choose a portfolio that maximizes CPT utility. We only address the question of *how* it can be done, algorithmically and computationally. We provide methods to solve the CPT portfolio optimization problem, but we emphasize that the conceptual framework behind the method is more general. Across computational economics and finance, nonconvex problems frequently arise. The framework for decomposing the problem into convex and concave parts can be extended to such other problems.

1.4 Previous and related work

Limited prior work exists on portfolio optimization with CPT utility. Analytical solutions exist for special cases such as single-period settings with one risk-free and one risky asset [BG10, HZ11, ZZ17] or for two-fund separation under elliptical distributions [PS12]. Extensions of these special cases to a multi-period setting are considered in [SCL15]. For the general multi-asset cases, heuristics such as particle swarm simulation [BCN20], or grid search methods [HM14] are employed, which have been extended to the multi-period case using dynamic programming [DL12, BH09]. While grid search can accommodate constraints, the particle swarm method used in [BCN20] cannot, and requires a hyperparameter to be

chosen to turn constraints into penalties.

The evaluation of CPT utility along the mean-variance frontier is a commonly used heuristic [LL04, HM14]. Some authors (*e.g.*, [SAM22]) use numerical methods to maximize CPT utility on small problems, do not explicitly mention the numerical solve method, suggesting the use of generic nonlinear optimizers. In contrast, we focus on custom methods that exploit the special structure of the CPT utility maximization problem.

After the initial release of this manuscript, Yan et al. [YJSC22] proposed a method for optimizing a portfolio using CPT utility based on the alternating direction method of multipliers (ADMM) (see, *e.g.*, [BPC+11]). This work is closely related to ours, in that they consider general multi-asset portfolios, and exploit convexity structure, although in a different way than we do. It is the only other method we are aware of that exploits the convexity structure of the CPT utility and can handle constraints. While we make an approximation about the monotonicity of the weights, they employ a method which does not globally solve one of their sub-problems in order to obtain tractable speeds. Carrying out a direct comparison of the methods is not immediately possible

1.5 Outline

We start in §2 by defining CPT utility, fixing our notation. The CPT utility extends prospect theory (PT) utility, described in §2.1, by adding a reweighting function, described in §2.2. In §2.3 we explore the convexity structure of CPT utility, followed by a description of the CPT utility portfolio optimization problem in §2.4. In §3 we describe algorithms that can be used to find a portfolio that maximizes CPT utility. The first method, presented in §3.1, is a minorization-maximization method that relies on the convexity structure described in the previous section. The second method, described in §3.2, uses the convex-concave procedure, a method for maximizing the sum of a convex and concave function, and iterates over the probability weights that appear in the CPT utility. The last method, given in §3.3, is a projected gradient type method, which can scale to large problem sizes. Numerical experiments are presented in §4, where we evaluate all methods on a toy problem with three assets, a medium-sized problem with more assets, and a large-scale problem, all based on historical asset class data. We give some conclusions in §5.

2 Cumulative prospect theory utility

2.1 Prospect theory utility

In this section we introduce PT utility, the first building block of CPT utility. Like VNM utility, it is monotonically increasing, but PT utility is not concave. PT utility has an inflection point at the origin, which represents a reference wealth. It is concave for positive arguments, *i.e.*, investors are risk averse for gains, and convex on for negative arguments, *i.e.*, investors are risk seeking for losses. Exponential utility functions are commonly used for both the convex and the concave sections of the PT utility function. We thus define the

positive and negative exponential utilities as

$$u_+(x) = 1 - \exp(-\gamma_+ x), \qquad u_-(x) = -1 + \exp(\gamma_- x),$$

where $\gamma_+, \gamma_- > 0$ are parameters. Here and throughout the paper, functions with a subscript plus sign are applied to gains, and functions with a subscript minus sign are applied to losses. Combining both functions yields the exponential prospect theory utility for a single return

$$u^{\text{prosp}}(x) = \begin{cases} u_+(x) & \text{if } x \geq 0 \\ u_-(x) & \text{otherwise} \end{cases},$$

which is S-shaped. Prospect theory further accounts for loss aversion, which requires $\gamma_- > \gamma_+$, *i.e.*, a marginal decrease in wealth would decrease the utility more than a marginal increase in wealth would increase the utility.

2.2 Probability reweighting

The second building block of CPT utility is a reweighting function that assigns higher weights to more extreme outcomes. As is common in the CPT literature, we first define the weighting functions $w(p) : [0,1] \to [0,1]$. We take the specific weighting functions

$$w_+(p) = \frac{p^{\delta_+}}{(p^{\delta_+} + (1-p)^{\delta_+})^{1/\delta_+}}, \qquad w_-(p) = \frac{p^{\delta_-}}{(p^{\delta_-} + (1-p)^{\delta_-})^{1/\delta_-}},$$

where $\delta_+, \delta_- > 0$ are parameters. We now specify the notion of extreme outcomes. Let $r_1, \ldots, r_N \in \mathbf{R}^n$ be the empirical distribution of realized returns on n assets. Consider a vector of portfolio weights $w \in \mathbf{R}^n$, with $\mathbf{1}^T w = 1$, where $\mathbf{1}$ is the vector with all entries one. The associated portfolio returns are $r_1^T w, \ldots, r_N^T w \in \mathbf{R}$. Without reweighting, all returns would have equal weight. Let N_- denote the number of negative returns, and N_+ the number of nonnegative returns, with $N_- + N_+ = N$. We let ρ_i denote the returns re-ordered or sorted by the portfolio returns, with index value $i = 1, \ldots, N$,

$$w^T\rho_1 \leq \cdots \leq w^T\rho_{N_-} < 0 \leq w^T\rho_{N_-+1} \leq \cdots \leq w^T\rho_N,$$

i.e., $w^T\rho_1$ is the largest loss and $w^T\rho_N$ is the largest gain. We define the positive and negative decision weights respectively as

$$\pi'_{+,j} = \begin{cases} w_+((N_+ - j + 1)/N) - w_+((N_+ - j)/N) & j = 1, \ldots, N_+ - 1 \\ w_+(1/N) & j = N_+, \end{cases}$$

$$\pi'_{-,j} = \begin{cases} w_-((N_- - j + 1)/N) - w_-((N_- - j)/N) & j = 1, \ldots, N_- - 1 \\ w_-(1/N) & j = N_-. \end{cases}$$

We would argue that π'_+ and π'_- should be nondecreasing, *i.e.*, we should put higher weight on more extreme events. This occurs for most reasonable choices of parameters, but there are choices for which monotonicity is (slightly) violated. Thus, we force monotonicity

by replacing $\pi'_{+,j}$ with $\min(\pi'_+)$ for all $j < \operatorname{argmin}(\pi'_+)$, and likewise for π'_-. We zero-pad π'_+ and π'_- from the left to be length N, *i.e.*, $\pi_+ = (0_{N_-}, \pi'_+)$ and $\pi_- = (0_{N_+}, \pi'_-)$, where the subscript on the vector zero denotes its dimension. We define for a monotone increasing probability vector π

$$f_\pi(x) = \sum_{i=1}^{N} \pi_i x_{(i)},$$

which is sometimes called the weighted-ordered-sum or dot-sort function. (The notation $x_{(i)}$ means the ith smallest element of the vector x.) Then, with

$$\phi_+(x) = \max(x, 0), \qquad \phi_-(x) = -\min(x, 0),$$

we have the total CPT utility given by

$$U^{\text{cpt}}(w) = f_{\pi_+}\left(\phi_+\left(u_+(Rw)\right)\right) - f_{\pi_-}\left(\phi_-\left(u_-(Rw)\right)\right).$$

2.3 Convexity properties

In this section we describe some convexity properties of the CPT utility function. PT utility is convex for negative arguments and concave for positive arguments by definition. CPT utility, *i.e.*, with reweighting, is a difference of two structured terms

$$U^{\text{cpt}}(w) = \underbrace{f_{\pi_+}(\phi_+(}_{\text{convex}}\underbrace{u_+(Rw)}_{\text{concave}}\underbrace{))}_{} - \underbrace{f_{\pi_-}(\phi_-(}_{\text{convex}}\underbrace{u_-(Rw)}_{\text{convex}}\underbrace{))}_{}.$$

The first term is a composition of dot-sort-positive, $f_\pi \circ \phi_+$, and the concave exponential utility for gains, u_+. The dot-sort-positive function is convex, because dot-sort is convex and increasing for positive weights π, and ϕ_+ is convex. The weighted sum in the CPT utility is consistent with dot-sort whenever the weights in the dot-sort function are monotone nondecreasing, *i.e.*, $\pi_1 \leq \pi_2 \cdots \leq \pi_N$. Similarly, $f_\pi \circ \phi_-$ is convex following the same reasoning, making $-f_\pi \circ \phi_-$ concave, which is in turn composed with the convex exponential utility for losses, u_-. We note that for each return, only one argument of the difference contributes to the CPT utility, as $\phi_+(x)\phi_-(x) = 0$. These convexity properties motivate principled algorithmic approaches to maximizing the CPT utility, which we explore in §3.1 and §3.2.

2.4 CPT utility portfolio optimization problem

The CPT utility portfolio optimization problem is

$$\begin{array}{ll} \text{maximize} & U^{\text{cpt}}(w) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array} \tag{2}$$

with variable w, where $\mathcal{W}$ is the set of feasible portfolio weights. It is not a convex optimization problem, so we will seek approximate solution methods.

We mention some simple methods for solving or approximately solving the CPT utility portfolio optimization problem (2). If the number of assets is very small (say, 3 or 4), we can solve it by brute force computation, by evaluating U^{cpt} over a fine grid of values.

A reasonable heuristic for approximately solving the CPT utility portfolio optimization problem, motivated by [LL04], leverages our ability to efficiently solve the MV portfolio optimization problem. We find the so-called efficient frontier, by solving the MV problem for a number of different values of the risk aversion parameter γ. (This gives the MV efficient frontier.) We evaluate the CPT utility of each of these portfolios, and choose the one with the largest value. While this does not in general solve the problem (2), it often produces a very good, *i.e.*, nearly optimal, portfolio. It can be used as an initial guess for the iterative methods described below. We refer to this method as the MV heuristic for CPT maximization.

3 Optimization methods

3.1 Minorization-maximization method

The CPT utility has the composition form

$$U^{\text{cpt}}(w) = (f_{\pi_+} \circ \phi_+)(u_+(Rw)) - (f_{\pi_-} \circ \phi_-)(u_-(Rw)).$$

We denote a general linearization of a convex (concave) function $h(w)$ at the point $\hat{w}$ as

$$\widehat{h}(w, \hat{w}) = h(\hat{w}) + g^T(w - \hat{w}),$$

where g is a subgradient (supergradient) of the function h. As all linearizations that follow occur at $\hat{w}$, we suppress the second argument.

At $\hat{w}$, we create a concave approximation of the first term of $U^{\text{cpt}}(w)$ by linearizing $(f_{\pi_+} \circ \phi_+)$. We approximate the second term in the difference by linearizing the inner convex utility u_-. To linearize dot-sort-positive, we observe that a subgradient is given by the vector g_x with entries

$$g_{x,i} = \begin{cases} 0 & \text{if } x_i < 0 \\ \pi_{+\sigma_x(i)} & \text{otherwise,} \end{cases}$$

where σ_x is the permutation which maps i to the rank of x_i in x. The minorant at $\hat{w}$ is therefore given by

$$\tilde{U}^{\text{cpt}}(w) = (\widehat{f_{\pi_+} \circ \phi_+})(u_+(Rw)) - (f_{\pi_-} \circ \phi_-)(\widehat{u_-}(Rw)).$$

The minorization-maximization (MM) algorithm (also called the majorization-minimization algorithm when solving a minimization problem) simply iterates between creating the minorant at the current iterate and then maximizing it to find the next iterate [HL00]. Our minorant is concave, so maximizing it is efficient. Here $\mathcal{W}$ can be any DCP convex constraint set, since each iteration requires solving a general convex optimization problem.

Algorithm 1 Minorization-maximization method

given w^0, let $\hat{w} := w_0$.
repeat:

1. Let w^{next} be a maximizer of $\tilde{U}^{\text{cpt}}(w)$, subject to $w \in \mathcal{W}$.
2. **break if** $w^{\text{next}} = \hat{w}$.
3. Update $\hat{w} = w^{\text{next}}$.

return $\hat{w}$.

3.2 Iterated convex-concave method

Though the CPT portfolio optimization objective is non-convex, we know the curvature and sign properties of the component functions which are composed to form the utility. In particular, PT utility is convex on the negative reals, and concave on the nonnegative reals. Thus, it is amenable to optimization via the convex-concave procedure (CCP) [LB15, SDGB16, YR03, LS09]. The convex-concave procedure for maximization iteratively linearizes the second term in the sum of a concave and a convex function, and maximizes this surrogate objective. While the PT utility has this clear curvature, the CPT utility does not, due to reweighting. Our heuristic approach is to fix the probability weights in each iteration, and then solve the fixed weight CPT utility optimization problem with the convex-concave procedure. Once the weights have been fixed, we can write the PT utility as a concave function

$$f^{\text{ccv}}(x) = \begin{cases} 1 - \exp(-\gamma_+ x) & \text{if } x \geq 0 \\ \gamma_- x & \text{otherwise,} \end{cases}$$

plus a convex function,

$$f^{\text{cvx}}(x) = \inf_{z \leq 0,\, z \leq x} \left(-1 + \exp(\gamma_- z) - \gamma_- z\right).$$

Here "cvx" and "ccv" denote convex and concave, respectively. (See §A for a derivation of these functions in disciplined convex programming (DCP) form.) Unlike the MM algorithm in §3.1 which maximizes a global lower bound, this approximation is only local, so we include a trust region constraint, which we omit from the algorithm description for brevity. Note that as before, $\mathcal{W}$ can be any DCP convex constraint set.

Algorithm 2 Convex-concave procedure

given w_0, let $\hat{w} := w_0$.
repeat:

1. Let π be the concatenation of the reversed vector π'_- followed by π'_+, where π'_- and π'_+ are the decision weights associated with $\hat{w}$ (see §2).
2. Let L_i^{cvx} be the linearization of f^{cvx} at $(R\hat{w})_{(i)}$
3. Let w^{next} be a maximizer of $\sum_i \pi_i \left(f^{\text{ccv}}\left(w^T\rho_i\right) - L_i^{\text{cvx}}\left(w^T\rho_i\right)\right)$, subject to $w \in \mathcal{W}$, where ρ_i is the row of R associated with $(R\hat{w})_{(i)}$.
4. **break if** $w^{\text{next}} = \hat{w}$.
5. Update $\hat{w} = w^{\text{next}}$.

return $\hat{w}$.

3.3 Projected gradient ascent

We first consider maximizing the CPT utility using gradient ascent (GA). While the CPT utility is not differentiable everywhere, we can use an automatic differentiation package such as PyTorch [PGM+19] to specify the computation chain for the problem and automatically compute the gradient at points where the utility is differentiable, and a reasonable surrogate for the gradient (such as a subgradient for convex functions) at points where it is not. Such libraries are extremely fast and optimized for use on GPUs. We can then perform gradient ascent, together with a method to enforce the portfolio constraints. Projected gradient ascent consists of the iterations

$$w^{k+1} = \Pi\left(w^k + \eta^k \nabla f(w^k)\right),$$

where k denotes iteration, $\eta^k > 0$ is a stepsize and Π is ℓ_2 or Euclidean projection onto the constraint set $\mathcal{W}$, *i.e.*, $\Pi(w) = \operatorname{argmin}_{w' \in \mathcal{W}} \|w' - w\|_2$.

Using these computation frameworks requires the projection to be expressed as a simple computation chain, which can be done in simple cases such as a long-only portfolio, *i.e.*, $\mathcal{W} = \mathbf{R}^n_+$. Another option to handle long-only portfolio constraints is to parametrize nonnegative portfolio weights via a multinomial logistic map,

$$w_i = \frac{\exp x_i}{\sum_j \exp x_j}, \quad i = 1, \ldots, n,$$

where x is an unconstrained variable.

Asset class	Region	GFD symbol
Equity	US	SPXTRD
Equity	Europe	STOXXER
Equity	Japan	TOPXDVD
Equity	Emerging markets	TRGFDEM
Government Bonds	US	TRUSG10M
Corporate Bonds	US	TRCCRBD
Government Bonds	Europe	TREUROGM
Government Bonds	Japan	TRJPNGVM
Bills	US	TRUSABIM
Bills	Europe	TREUROBM
Bills	Japan	TRJPNBIM
Commodities	Global	TRUSACOM
Gold	Global	XAU_BD
Silver	Global	XAG_HD

Table 1: Asset classes and regions in the data set.

4 Numerical examples

To evaluate the efficiency and performance of the proposed methods, we compare them in a series of numerical experiments with increasing data size. We first compile a data set consisting of $N = 600$ monthly returns, covering the 50-year period from 07–1972 to 06–2022. The $n = 14$ assets consist of equities and fixed income securities from different regions, as well as commodities, as displayed in table 1. The data was obtained from Global Financial Data (GFD) [Dat22]. The indices are total return indices, *i.e.*, they include dividends and interest payments. The GFD index methodology extends the index history back in time by combining multiple single indices where necessary. We provide the GFD symbol for each asset class for reference.

4.1 Toy example

Our first small example uses $n = 3$ assets: US stocks, 10-year US Treasury bonds, and 3-month US Treasury T-bills. We choose the CPT function with parameters

$$\gamma_+ = 8.4, \quad \gamma_- = 11.4, \qquad \delta_+ = 0.77, \quad \delta_- = 0.79,$$

which are reasonable, and at the same time exhibit clear non-convexity and even multimodality of the CPT utility. (Many other reasonable choices of the parameters lead to unimodal CPT utility, which makes the portfolio optimization problems easy; our goal is to evaluate the methods on more challenging problem instances.)

Figure 1 gives a plot of this utility function for $\mathcal{W} = \mathbf{R}^2_+$, *i.e.*, long-only portfolios. The horizontal axis is w_1, the fraction invested in stocks; the vertical axis is w_2, the fraction

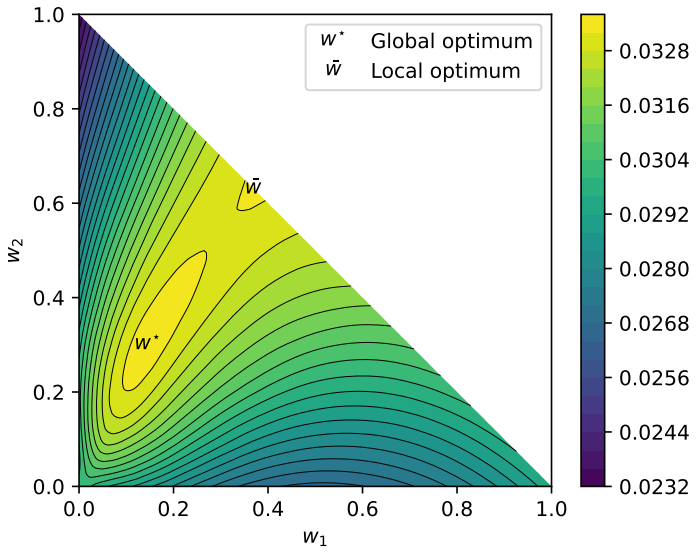


Figure 1: CPT utility surface for a long-only portfolios of stocks (w_1), bonds (w_2), and T-bills ($w_3 = 1 - w_1 - w_2$).

invested in bonds. The fraction invested in T-bills is $w_3 = 1 - w_1 - w_2$. Thus, the point $(0, 0)$ represents a portfolio fully invested in T-bills. Any portfolio on the diagonal connecting $(1, 0)$ and $(0, 1)$ represents portfolios invested in a convex combination of only stock and bonds. Since there are only two portfolio weights to optimize over, we can find the global maximum using brute-force evaluation of the utility over a fine grid. The global maximum is attained at $w^\star = (0.14, 0.3)$, yielding $U^{\text{cpt}}(w^\star) = 0.0334$. In addition, there is a local optimum with slightly lower utility at $\bar{w} = (0.37, 0.63)$, which yields $U^{\text{cpt}}(\bar{w}) = 0.0332$.

MV frontier. A simple heuristic is to evaluate U^{cpt} on portfolios along the mean-variance efficient MV frontier and choosing the maximizing portfolio among them. Based on the sample mean and covariance of the returns, we first find the return-maximizing and risk-minimizing portfolios, and then sample 100 points that are equidistant in volatility space along the efficient MV frontier. Figure 2 (a) shows the efficient MV frontier, and the portfolio with the highest CPT utility along it, w^{mv}, associated with risk aversion parameter $\gamma = 3.2$. It achieves CPT utility of $U^{\text{cpt}}(w^{\text{mv}}) = 0.0328$. Figure 2 (b) shows U^{mv} for the choice $\gamma = 3.2$. It should be noted that the MV frontier is independent of the parameter choice of the CPT utility function, and in general the MV optimum can be far away from a local optimum of CPT.

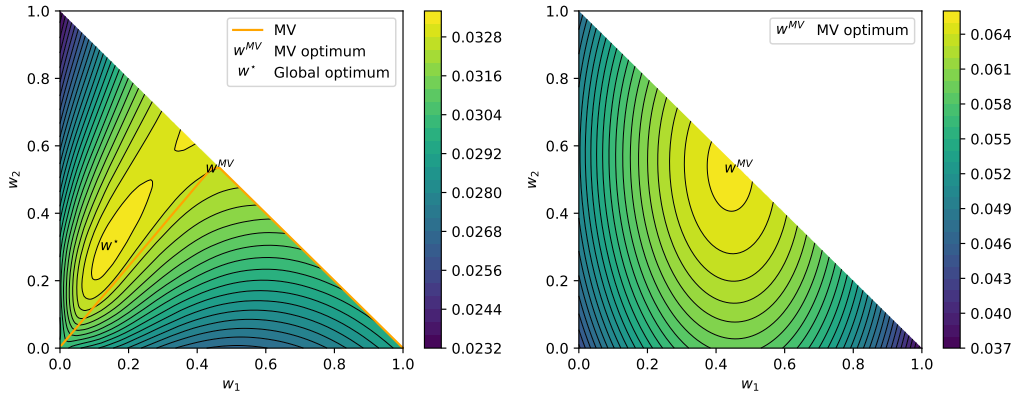


Figure 2: (a) Maximizing U^{cpt} along the MV frontier, resulting in w^{mv}. (b) Utility surface of U^{mv} for the choice of λ that results in w^{mv}.

Iterative methods. As all remaining methods depend on initialization, we compare the convergence from equal weights, three points close to a full investment in each single asset, as well as the MV optimum in figure 3. The MM algorithm terminates at a local maximum from all starting points within fewer than 30 iterations. Likewise, CC converges to a local optimum or a point where the numerical stopping criterion is reached in all cases within at most 11 iterations, albeit on a visually more erratic path. Lastly, the GA method also converges to a point close to a local optimum in all cases. Thus, all iterative methods appear to perform equally well on the toy example.

Diversification. To see the effect of the CPT utility on diversification, as well as to understand how the different methods alter the portfolio weights, we run a backtest using a sliding window of 100 observations along our previously described data set of 600 monthly returns. Following [GK08] and [YJSC22], we compute the sum of squared portfolio weights (SSPW), $\|w - \frac{1}{n}\|_2^2$, as a measure of diversification. We compare the MM, CC, and GA methods to the MV heuristic in figure 4. We observe that the MM and CC methods all result in a lower SSPW than the MV heuristic, *i.e.*, the portfolios are more diversified. In addition, we explore how the assumptions of probability reweighting and loss aversion that are inherent to the CPT utility affect the portfolio weights. For this, we change set $\gamma_+ = \gamma_- = 11.4$ in the no loss aversion setting, and $\delta_+ = \delta_- = 1$ in the no probability reweighting setting. We find that both cases result in a much higher median SSPW, indicating that the portfolio weights are less diversified.

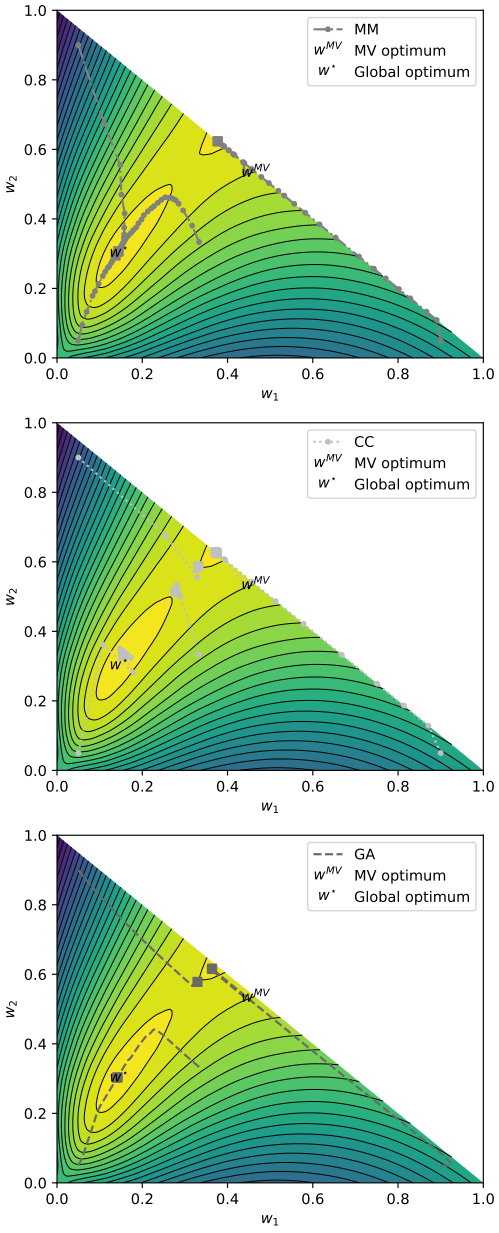


Figure 3: Convergence from different staring points for the MM (a), CC (b), and GA (c) methods.

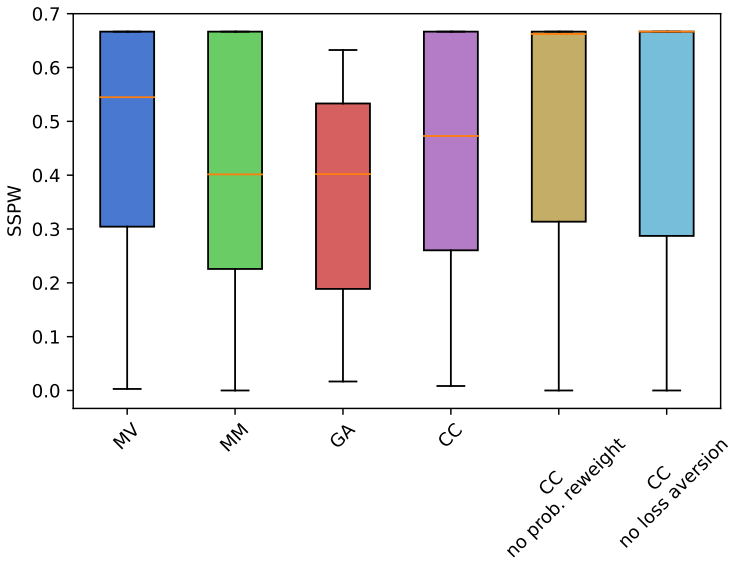


Figure 4: Comparison of the sum of squared portfolio weights across methods.

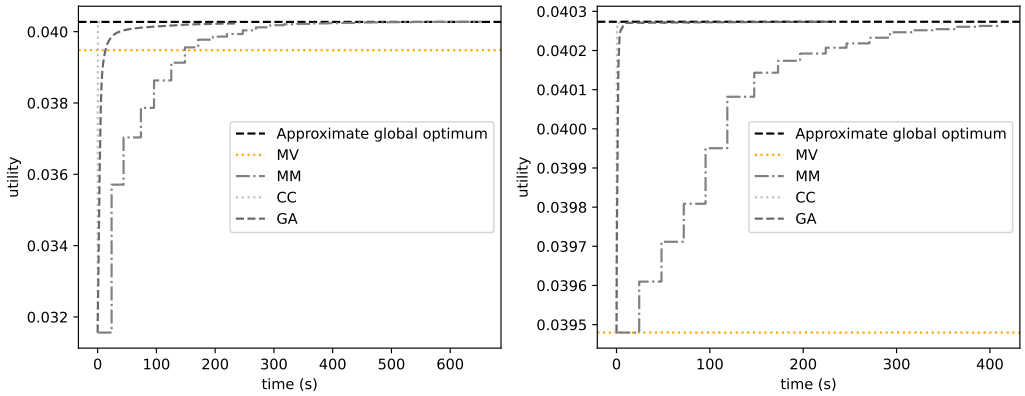


Figure 5: Comparison of wall-time across methods for the multi-asset example, started from (a) equal-weight portfolio and (b) the best MV portfolio.

4.2 Multi-asset example

We now extend our example to use all $n = 14$ assets, comparing the achieved utilities, as well as the required computation time. While comparing the absolute wall-times across different implementations can only approximate the computational efficiency of the algorithms, it is relevant to the practicality of the presented methods. The best portfolio on the efficient MV frontier attains a utility of 0.0395 in only 0.6 seconds. Starting all iterative methods from the equal-weight portfolio, CC terminates first, yielding a utility of 0.0403 in 4.1 seconds. MM also attains the same utility, but it takes substantially longer, terminating after about 650 seconds. GA also results in approximately the same utility, being slower than CC, but still dominating MM. When optimizing a single portfolio, we find that GA converges faster using the CPU. However, simultaneous optimization of many portfolios, which is naturally handled by this method, scales better when using a GPU. We use this observation to simultaneously optimize from 10,000 starting points. These weights are sampled from a symmetric Dirichlet distribution with concentration parameter $\alpha = 1$, *i.e.*, $w_0 \sim \mathrm{Dir}(\mathbf{1}^n)$, which is equivalent to a uniform distribution over the open standard $(n-1)$-simplex. The best resulting utility is denoted as the approximate global optimum, and is not higher than the utilities achieved by all iterative methods when started from equal weights. All methods, as well as the approximate global optimum, are visualized in figure 5 (a). In practice, it would likely be beneficial to leverage the fast computation of the MV portfolio as a starting point for the iterative methods. Indeed, as shown in figure 5 (b), this reduces the time to convergence substantially for all methods, with the GA method now converging in approximately ten seconds. MM is also faster, but still takes about 400 seconds to converge.

Investigating the sensitivity to the starting point, we sample 30 starting weights and compare the attained utilities. We find that MM and CC converge to a utility of 0.0403

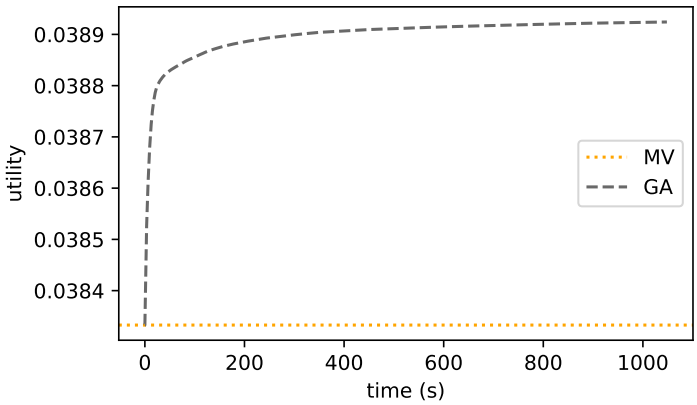


Figure 6: Convergence of the GA method for $N = 6,000$ starting from w^{mv}.

in all cases. GA, however, displays a higher variance. Its best utility is close to the values obtained by MM and CC. The median utility is 0.0401, which is higher than MV at 0.0395. The worst case utility is 0.0386, which is worse than all other methods.

4.3 Scaling to many return samples

To investigate the scalability of the methods to more return samples, we extend the 600 observations of our data set with synthetic returns. For this, we sample from a Gaussian mixture model with three components that was fit to the return data. We find that GA scales best, handling data sets of hundreds of thousands of observations. For such large data sets, MM and CC did not converge in a reasonable time. Further, as the GA implementation naturally handles optimizing multiple starting points simultaneously, the problem of high variance observed in §4.2 is alleviated. Figure 6 shows an example where we extend the original data set by a factor of ten, *i.e.*, we consider the case where $N = 6,000$. We choose w^{mv} as a principled starting point for the GA method. We observe that GA does improve over its starting point, however, looking at the axis scale reveals that the improvement over MV is marginal. We observe that the numerical value of the highest utility is different compared to the original data set in §4.2. This is expected, however, as the observed data only makes up 10% of the extended data set, and the data generating process for the synthetic returns is only an approximation of the true data generating process, which may not be fully described by any single distribution.

5 Conclusions

While the CPT utility is nonconvex and can even be multimodal, we identify some simple convexity properties. Specifically, the CPT utility is a difference of two structured functions, with the first term given by a composition of a convex function with concave arguments and the second term given by a composition of a convex function with convex arguments. This structure allows us to construct locally tangent concave minorants, which we use to develop a minorization-maximization algorithm to maximize the CPT utility numerically. While the analysis was restricted to the CPT utility, we believe that it motivates similar analyses for other nonconvex objectives commonly used in finance and economics. We provide several practical methods to maximize the CPT utility, including one massively scalable method, and two methods which can easily handle arbitrary convex portfolio constraints. To the best of our knowledge, previous work on maximizing CPT utility considered only simple analytical cases or small problem instances with generic nonlinear optimizers.

As a practical matter, for small problems with arbitrary convex constraints, the MM method has shown smooth convergence and is thus the recommended default method. If this method is too slow, but the portfolio constraints are complex, the CC method should be used instead. For large problems with simple constraints, the GA method appears to be the best choice. As there is low scaling overhead, one should optimize many portfolios simultaneously, including the MV optimal portfolio, an equal weight portfolio, as well as randomly sampled starting points. As all methods are readily available in the accompanying code, it is easy to experiment for the given use case.

Lastly, it is worth noting that the simple method of approximately maximizing the CPT utility by restricting the feasible set to the MV frontier seems to closely approximate the optimal CPT utility in many problem instances.

Declarations

Funding. P. Schiele is supported by a fellowship within the IFI program of the German Academic Exchange Service (DAAD). This research was partially supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR, and by ONR grant N00014-22-1-2121.

Competing interests. The authors have no competing interests to declare that are relevant to the content of this article.

Data availability

The code is available at https://github.com/cvxgrp/cptopt. The dataset analyzed during the current study are not publicly available due to licensing reasons but are available from the corresponding author on reasonable request.

References

[BBD+17] S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth. Multi-period trading via convex optimization. *Foundations and Trends® in Optimization*, 3(1):1–76, 2017.

[BCN20] D. Barro, M. Corazza, and M. Nardon. Cumulative prospect theory portfolio selection. *University Ca'Foscari of Venice, Dept. of Economics Research Paper Series No*, 26, 2020.

[BG10] C. Bernard and M. Ghossoub. Static portfolio choice under cumulative prospect theory. *Mathematics and Financial Economics*, 2(4):277–306, 3 2010.

[BH09] N. Barberis and M. Huang. Preferences with frames: A new utility specification that allows for the framing of risks. *Journal of Economic Dynamics and Control*, 33(8):1555–1576, 2009.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004.

[Dat22] Global Financial Data. Available: Global Financial Data [Online] https://globalfinancialdata.com/, 2022.

[DL12] E. De Giorgi and S. Legg. Dynamic portfolio choice and asset pricing with narrow framing and probability weighting. *Journal of Economic Dynamics and Control*, 36(7):951–972, 2012.

[GK99] R. Grinold and R. Kahn. *Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Selecting Superior Returns and Controlling Risk*. McGraw-Hill Library of Investment and Finance. Mcgraw-hill, New York, 1999.

[GK08] W. Goetzmann and A. Kumar. Equity portfolio diversification. *Review of Finance*, 12(3):433–463, 03 2008.

[HL00] D. Hunter and K. Lange. Quantile regression via an MM algorithm. *Journal of Computational and Graphical Statistics*, 9(1):60–77, 2000.

[HM14] T. Hens and J. Mayer. Cumulative prospect theory and mean variance analysis: A rigorous comparison. *Swiss finance institute research paper*, (14-23), 2014.

[HZ11] X. He and X. Zhou. Portfolio choice under cumulative prospect theory: An analytical treatment. *Management Science*, 57(2):315–331, 2011.

[KT79] D. Kahneman and A. Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979.

[LB15] T. Lipp and S. Boyd. Variations and extension of the convex–concave procedure. *Optimization and Engineering*, 17(2):263–287, 11 2015.

[LL04] H. Levy and M. Levy. Prospect theory and mean-variance analysis. *The Review of Financial Studies*, 17:1015–1041, 10 2004.

[LM79] H. Levy and H. Markowitz. Approximating expected utility by a function of mean and variance. *The American Economic Review*, 69(3):308–317, 1979.

[LS09] G. Lanckriet and B. Sriperumbudur. On the convergence of the concave-convex procedure. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22, New York, 2009. Curran Associates, Inc.

[Mar52] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7:77–91, 1952.

[Mer69] R. Merton. Lifetime portfolio selection under uncertainty: The continuous-time case. *The Review of Economics and Statistics*, 51(3):247–257, 1969.

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., New York, 2019.

[PS12] T. Pirvu and K. Schulze. Multi-stock portfolio optimization under prospect theory. *Mathematics and Financial Economics*, 6(4):337–362, 2012.

[SAM22] S. Srivastava, A. Aggarwal, and A. Mehra. Portfolio selection by cumulative prospect theory and its comparison with mean-variance model. *Granular Computing*, 2022.

[SCL15] Y. Shi, X. Cui, and D. Li. Discrete-time behavioral portfolio selection under cumulative prospect theory. *Journal of Economic Dynamics and Control*, 61:283–302, 2015.

[SDGB16] X. Shen, S. Diamond, Y. Gu, and S. Boyd. Disciplined convex-concave programming. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1009–1014, 2016.

[SDR21] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on stochastic programming: modeling and theory*. SIAM, Philadelphia, PA, 2021.

[TK92] A. Tversky and D. Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, 5(4):297–323, 1992.

[vMR44] J. von Neumann, O. Morgenstern, and A. Rubinstein. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press, Princeton, NJ, 1944.

[YJSC22] Y. Yan, R. Jiang, Y. Shi, and X. Cui. Portfolio optimization under cumulative prospect theory: An alternating direction method of multipliers, 2022.

[YR03] A. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15(4):915–936, 04 2003.

[ZZ17] B. Zou and R. Zagst. Optimal investment with transaction costs under cumulative prospect theory in discrete time. *Mathematics and Financial Economics*, 11(4):393–421, 2017.

A DCP form of CCP objective

A.1 DCP form of f^{ccv}

To obtain the piecewise definition of f^{ccv}, we split up its argument into a positive and negative part,

$$x = x^+ + x^-, \qquad x^+ \geq 0, \quad x^- \leq 0.$$

Now, we observe that

$$f^{\text{ccv}}(x) = 1 - \exp(-\gamma_+ x^+) + \gamma_- x^-$$

is concave, because $\gamma_- > \gamma_+ \geq \frac{\partial(1-\exp(-\gamma_+ x))}{\partial x}$ for $x \geq 0$. We implement f^{ccv} in DCP form via its hypograph,

$$\{(x,t) \mid f(x) \geq t\} = \{(x,t) \mid \exists\, x^+ \geq 0,\ x^- \leq 0,\ x = x^+ + x^-,\ 1 - \exp(-\gamma_+ x^+) + \gamma_- x^- \geq t\},$$

which in practice means that to add this function to an optimization problem, we introduce new variables t, x^+, and x^-, replace f^{ccv} with t, and add the constraints

$$t \leq 1 - \exp(-\gamma_+ x^+) + \gamma_- x^-, \qquad x = x^+ + x^-, \qquad x^+ \geq 0, \qquad x^- \leq 0.$$

A.2 DCP form of f^{cvx}

To see that f^{cvx} is convex, we can equivalently represent it as a partial minimization of the convex function (of z and x jointly)

$$-1 + \exp(\gamma_- z) - \gamma_- z + \mathbf{1}\{z \leq x\}$$

over the convex set $\{(z,x) \mid z \leq 0\}$. The function can be used in DCP frameworks that provide the indicator function and partial minimization. Alternatively, the indicator function can be omitted when adding the explicit constraints

$$z \leq x, \qquad z \leq 0.$$

B Code snippets

```
from scipy.stats import multivariate_normal as normal

from cptopt.optimizer import *
from cptopt.utility import CPTUtility

# Generate returns
corr = np.array([
    [1, -.2, -.4],
    [-.2, 1, .5],
    [-.4, .5, 1]
])
sd = np.array([.01, .1, .2])
Sigma = np.diag(sd) @ corr @ np.diag(sd)

np.random.seed(0)
r = normal.rvs([.03, .1, .193], Sigma, size=100)

# Define utility function
utility = CPTUtility(
    gamma_pos=8.4, gamma_neg=11.4,
    delta_pos=.77, delta_neg=.79
)

initial_weights = np.array([1/3, 1/3, 1/3])

# Optimize
mv = MeanVarianceFrontierOptimizer(utility)
mv.optimize(r, verbose=True)

mm = MinorizationMaximizationOptimizer(utility)
mm.optimize(r, initial_weights=initial_weights, verbose=True)

cc = ConvexConcaveOptimizer(utility)
cc.optimize(r, initial_weights=initial_weights, verbose=True)

ga = GradientOptimizer(utility)
ga.optimize(r, initial_weights=initial_weights, verbose=True)
```

Part VI.

ARMA Cell: A Modular and Effective Approach for Neural Autoregressive Modeling

This chapter is a reprint of:

Schiele, P., Berninger, C., Rügamer, D. (2021) ARMA Cell: A Modular and Effective Approach for Neural Autoregressive Modeling. *Available at https://arxiv.org/abs/2208.14919*. Planned submission (after revising according to submission guidelines): *Scientific Reports*.

The included version is the second revision of the essay.

Author contributions:

The idea to combine ARMA models with neural networks was conceived by Philipp Schiele after discussing semi-structured neural network methods with David Rügamer and Christoph Berninger. Philipp Schiele played a key role in refining this idea into the ARMA cell and developing the corresponding initial draft of the essay, which was later revised by David Rügamer and Christoph Berninger. Philipp Schiele also developed the software for the ARMA cell and conducted the numerical experiments. David Rügamer supervised the project throughout.

Supplementary material for reproducing the results available at:
https://github.com/phschiele/armacell_paper

Standalone software package available at:
https://github.com/phschiele/armacell

ARMA Cell: A Modular and Effective Approach for Neural Autoregressive Modeling

Philipp Schiele[1], Christoph Berninger[1], and David Rügamer[1]

[1]Department of Statistics, Ludwig-Maximilians-Universität München

January 12, 2024

Abstract

The autoregressive moving average (ARMA) model is a classical, and arguably one of the most studied approaches to model time series data. It has compelling theoretical properties and is widely used among practitioners. More recent deep learning approaches popularize recurrent neural networks (RNNs) and, in particular, Long Short-Term Memory (LSTM) cells that have become one of the best performing and most common building blocks in neural time series modeling. While advantageous for time series data or sequences with long-term effects, complex RNN cells are not always a must and can sometimes even be inferior to simpler recurrent approaches. In this work, we introduce the ARMA cell, a simpler, modular, and effective approach for time series modeling in neural networks. This cell can be used in any neural network architecture where recurrent structures are present and naturally handles multivariate time series using vector autoregression. We also introduce the ConvARMA cell as a natural successor for spatially-correlated time series. Our experiments show that the proposed methodology is competitive with popular alternatives in terms of performance while being more robust and compelling due to its simplicity.

Contents

1 Introduction

Despite the rapidly advancing field of deep learning (DL), linear autoregressive models remain popular for time series analysis among academics and practitioners. Especially in economic forecasting, datasets tend to be small and signal-to-noise ratios low, making it difficult for neural network approaches to effectively learn linear or non-linear patterns. Although research in the past has touched upon autoregressive models embedded in neural networks (*e.g.*, [CAM91, CMA94]), existing literature in fields guided by linear autoregressive models such as econometrics mainly focuses on hybrid approaches (see Section 2). These hybrid approaches constitute two-step procedures with suboptimal properties and often cannot even improve over the pure linear model. The DL community took a different route for time-dependent data structures, popularizing recurrent neural networks (RNNs), as well as adaptions to RNNs to overcome difficulties in training and the insufficient memory property [HS97] of simpler RNNs. In particular, methods like the Long Short-Term Memory (LSTM) cell are frequently used in practice, whereas older recurrent approaches such as Jordan or Elman networks seem to have lost ground in the time series modeling community [Jor86, Elm90]. This can be attributed to the more stable training and the insensitivity to information lengths in the data of more recent recurrent network approaches such as the LSTM cell.

While often treated as a gold standard, we argue that these more complex RNN cells (such as the LSTM cell) are sometimes used only because of the lack of modular alternatives and that their long-term dependencies or data-driven forget mechanisms might not always be required in some practical applications. For example, in econometrics, including a small number of lagged time series values or lagged error signals in the model is usually sufficient to explain most of the variance of the time series. Similar, sequences of images (*i.e.*, tensor-variate time series) such as video sequences often only require the information of a few previous image frames to infer the pixel values in the next time step(s). In addition, current optimization routines allow practitioners to train classical RNN approaches without any considerable downsides such as vanishing or exploding gradients.

Our contributions. In this work, we propose a new type of RNN cell (cf. Figure 1) that can be seen as a natural connection between the classical time series school of thoughts and DL approaches. To analyze how the ARMA modeling philosophy can improve neural network predictions, we

- embed ARMA models in a neural network cell, which has various advantages over classical approaches (see Section 4);
- further exemplify how this proposal can be extended to convolutional approaches to model tensor-variate time series such as image sequences (Section 4.4);
- demonstrate through various numerical experiments that our model is on par with or even outperforms both its classical time series pendant as well as the LSTM cell

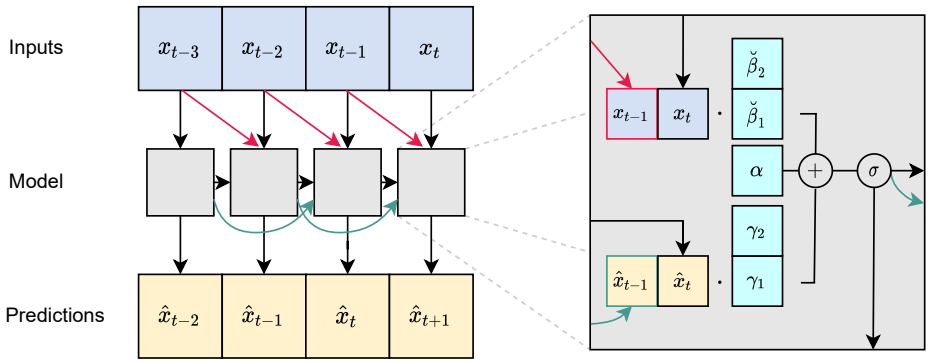


Figure 1: Left: Graphical visualizations of how predictions are computed in a univariate ARMA(2,2) cell using the time series values x from the current and previous time points as well as past model predictions $\hat{x}$. Right: Zooming in on the rightmost model cell from the left picture to show the computations of the ARMA cell with parameters as defined in (3).

in various settings, with architectures ranging from shallow linear to deep non-linear time series models;

- provide a fully-tested, modular and easy-to-use `TensorFlow` [ABC+16] implementation with a high-level syntax almost identical to existing RNN cells to foster its usage and systematic comparisons. It is available at https://github.com/phschiele/armacell.

The goal of this paper is further to make practitioners aware of an alternative to commonly used RNN cells, highlight that short-term recurrence can be sufficient in various time series applications, and that a simpler parameterized lag structures can even outperform data-driven forget mechanisms.

We start by discussing related literature in the following section. A short mathematical background is given in Section 3, followed by our proposed modeling approach in Section 4. We investigate practical aspects of our method in Section 5 and summarize all ideas and results in Section 6.

2 Related literature

Recent advancements in (deep) time series modeling merge statistical autoregressive approaches with DL, specifically in building larger modeling frameworks with multiple components, including appropriate preprocessing [SFGJ20, HZL+19]. We emphasize literature synthesizing classical time series analysis with fundamental building blocks of RNN modeling.

Traditional autoregressive approaches. Building upon the foundational contributions of Box et al. [BJRL15], Autoregressive Integrated Moving Average (ARIMA) models have been popularized due to their simple design coupled with practical effectiveness. They remain widely used in statistics, econometrics, and many other fields. In instances where the mean stationarity of a time series is attained without necessitating an integration step, the Autoregressive Moving Average (ARMA) model, a special case of ARIMA, becomes applicable. A spectrum of specialized variants has been developed to adeptly model certain nonlinear series. Notable examples include the bilinear models by Granger & Andersen [GA78, Rao81] and threshold models by Tong & Lim [TL09]. In addition, Seasonal ARIMA (SARIMA) models can include seasonal patterns, and the ARIMAX models have been extended to incorporate exogenous variables [BJRL15]. Analogies between ARMA models and RNNs have been discussed by various authors [CAM91, CMA94, Sax97], which we will discuss in the following.

Recurrent neural network approaches. RNNs naturally extend traditional models by incorporating previous states into current predictions. Notable variants such as Elman [Elm90] and Jordan [Jor86] networks have been developed, but suffer from issues like vanishing or exploding gradients (*e.g.*, [GBC16]) Advances like Gated Recurrent Units (GRU) [CMBB14] and Long Short-Term Memory cells (LSTM) [HS97] have been instrumental in mitigating these challenges, facilitating the modeling of long-term dependencies.

Combining classical time series approaches with neural networks. Hybrid models combining classical and neural network approaches have been explored, often involving a two-stage process where a neural network refines the residuals of a preceding AR(I)MA model [Zha03]. While several combinations, such as state space models with RNNs, have been proposed, they tend to lack modularity and general applicability [RP96].

Recurrent convolutional approaches. Spatio-temporal sequence forecasting benefits from the incorporation of convolutional operations into RNN cells, as seen in models like ConvLSTM and convolutional GRU which handle spatially distributed information efficiently [SCW+15, TLY+19].

3 Background and notation

In the following, we introduce our notation and the general setup for modeling time series. We will address univariate time series $x_t \in \mathbf{R}$ for time points $t \in \mathbf{Z}$ as well as multi- and tensor-variate time series, which we denote as $\boldsymbol{x}_t$ and $\boldsymbol{X}_t$, respectively.

ARMA model. The ARMA(p, q) model [BJRL15] for $p, q \in \mathbf{N}_0$ is defined as

$$x_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j} + \varepsilon_t, \tag{1}$$

where x_t represents the variable of interest defined for $t \in \mathbf{Z}$ and is observed at time points $t = 1, \ldots, T$, $T \in \mathbf{N}$[1]. Here, α, $\beta_1, \ldots, \beta_p$, $\gamma_1, \ldots, \gamma_q$ are real valued parameters and $\varepsilon_t \overset{iid}{\sim} \mathcal{F}(\sigma^2)$ is an independent and identically distributed (iid) stochastic process with pre-specified distribution $\mathcal{F}$ and variance parameter $\sigma^2 > 0$. By setting $q = 0$ or $p = 0$, the ARMA model comprises the special cases of a pure autoregressive (AR) and a pure moving average (MA) model, respectively. The class of ARMA models is, in turn, a special case of ARIMA models, where differencing steps are applied to obtain a stationary mean function before fitting the ARMA model. As stationarity is also a fundamental assumption for RNNs to justify parameter sharing [GBC16], we focus on the class of ARMA models in this work, *i.e.*, assume that differencing has already been applied to the data. A stationary time series is characterized by a constant mean and variance and a time invariant autocorrelation structure. $\alpha_0, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_q$ are model parameters and p and q characterize the number of lags of the dependent variable and the forecasting errors included in the model, respectively.

VARMA model. The univariate ARMA model can be generalized to a multivariate version – the vector autoregressive moving average (VARMA) model – by adapting the principles of the ARMA model for multivariate time series. The VARMA(p, q) model [TB81] for $p, q \in \mathbf{N}_0$ is defined as

$$\boldsymbol{x}_t = \boldsymbol{\alpha} + \sum_{i=1}^{p} \boldsymbol{B}_i \boldsymbol{x}_{t-i} + \sum_{j=1}^{q} \boldsymbol{\Gamma}_j \boldsymbol{\varepsilon}_{t-j} + \boldsymbol{\varepsilon}_t \tag{2}$$

where $\boldsymbol{x}_t, t \in \mathbf{Z}$ represents a vector of time series observed at time points $t = 1, \ldots, T$. $\boldsymbol{B}_i$ and $\boldsymbol{\Gamma}_j$ are time-invariant $(k \times k)$-matrices, where $k \in \mathbf{N}$ represents the number of individual time series. $\boldsymbol{\varepsilon}_t$ is a k-dimensional iid stochastic process with pre-specified k-dimensional distribution $\mathcal{F}(\boldsymbol{\Omega})$ and covariance matrix $\boldsymbol{\Omega}$. Furthermore, the error term $\boldsymbol{\varepsilon}_t$ needs to satisfy the following three conditions:

1. $E[\boldsymbol{\varepsilon}_t] = \mathbf{0}$ for all $t \in \mathcal{T}$
2. $E[\boldsymbol{\varepsilon}_t \boldsymbol{\varepsilon}_t^T] = \boldsymbol{\Omega}$ is a (k x k) positive semi-definite covariance matrix
3. $E[\boldsymbol{\varepsilon}_t \boldsymbol{\varepsilon}_{t-c}^T] = \mathbf{0}$ for any non-constant $c \in \mathbf{N}$.

By setting $q = 0$, the VARMA model comprises the special cases of a pure autoregressive (VAR) model, which is the most common VARMA model used in applications. Similar to the ARMA model being a special case of the ARIMA model class, the VARMA model is a special case of the VARIMA model class, representing only stationary time series.

[1] It is common to define a time series for time points $t = 1, \ldots, T$ to describe its current value and recent history, while time series dynamics are assumed to originate from time points prior to $t = 1$, hence $t \in \mathbf{Z}$

4 ARMA-based neural network layers

ARMA models have been successfully used in many different fields and are a reasonable modeling choice for time series in many areas. This section introduces a neural network cell version of the ARMA mechanism. While very similar to Elman or Jordan networks, the proposed cell exactly resembles the ARMA computations and can be used in a modular fashion in any neural network architecture where recurrent structures are present. Emulating the ARMA logic in a recurrent network cell has various advantages. It allows to 1) recover estimated coefficients of classical ARMA software (see Supplementary Material B.1 for an empirical investigation of the convergence), but can also be used to fit ARMA models for large-scale or tensor-variate data (which is otherwise computationally infeasible), 2) modularly use the ARMA cell in place for any other RNN cell, 3) combine ARMA approaches with other features from neural networks such as regularization and thereby seamlessly extend existing time series models, and 4) model hybrid linear and deep network models that were previously only possible through multi-step procedures. As shown in our numerical experiments section, an ARMA cell can further lead to comparable or even better prediction performance compared to modern RNN cells.

4.1 ARMA cell

An alternative formulation of the ARMA model can be derived by incorporating the observed (or estimated) residual term $\hat{\varepsilon}$ through the predictions $\hat{x}_t := x_t - \hat{\varepsilon}_t, t \in \mathbf{Z}$. Thus, (1) can be defined in terms of its intercept, the model predictions $\hat{x}_t$ and the actual time series values x_t as

$$\hat{x}_t = \alpha + \sum_{i=1}^{\max(p,q)} \breve{\beta}_i x_{t-i} - \sum_{j=1}^{q} \gamma_j \hat{x}_{t-j}, \tag{3}$$

where

$$\breve{\beta}_i = \begin{cases} \beta_i + \gamma_i & \text{for } i \leq \min(p,q), \\ \beta_i & \text{for } i > q \text{ and } p > q, \\ \gamma_i & \text{for } i > p \text{ and } p < q. \end{cases}$$

A detailed derivation of (3) is given in A. Using (3), we can implement the ARMA functionality as an RNN cell. More specifically, the recurrent cell processes the p lagged time series values as well as the q predicted outputs of the previous time steps and computes a linear combination with parameters $\breve{\beta}_i$ and γ_j. After adding a bias term, the final output $\hat{x}_t$ is given by a (non-linear) activation function σ of the sum of all terms. Figure 1 gives both a higher-level view of how predictions are computed in the ARMA cell as well as a description of how the cell is defined in detail. In addition to the classical ARMA computations in the cell, the activation function σ allows to straightforwardly switch between a linear ARMA model and a non-linear version.

The above-mentioned ARMA cell has the same hypothesis space as the classical ARMA model when using a single-unit ARMA cell with a linear activation function. While using

a non-linear activation for the outputs, in this case, is equivalent to using a link function (as done in generalized linear models) for the classical ARMA model, extensions using multiple units or stacking ARMA cells (see below) increase the model's expressiveness. As for regular multi-layer perceptrons, where each node is a simple regression model with an activation function and the combination of multiple units makes the models more expressive, these extensions combine simpler ARMA models and therefore allow modeling more complex relationships.

Advantages and comparison to other cells. Modeling a classical ARMA model in a neural network can be more stable in the estimation of coefficients due to the use of a stochastic first-order method (less vulnerable to ill-conditioning and numerical instabilities), which is also confirmed in our numerical experiments in numerous settings. Training the ARMA model using mini-batch optimization, further allows scaling to large data sets, which is especially beneficial when modeling multivariate time series where the complexity of classical models increases substantially with the number of parameters and the multivariate time series dimension.

In contrast to the standard RNN cell, the ARMA cell internally can access multiple previous states and lagged features, making it potentially easier to learn time dependencies and recurrences. The standard RNN cell, in contrast, only relies on the current input and the previous cell state. In other words, the ARMA cell allows for a more complex autoregressive structure and, in contrast to the simple RNN, provides a way to model moving averages. This can also be explained using Figure 1, where the standard RNN cell can represent the black arrows, but not the red and green connections.

Last but not least, the ARMA cell can be used to seamlessly model hybrid models end-to-end in one holistic network, which historically has always been implemented using two-step approaches [Zha03], yielding potentially inferior performance as the models in both steps are not jointly optimized. This is also confirmed by our numerical results in Section 5.3.

4.2 Training procedure

The ARMA cell is trained as follows. For a given sequence, it creates predictions by recursively applying (3). This is done in one forward pass. To also allow predictions for the first q time points in each sequence, we need to pad the sequence of previous predictions with 0-values. Further details on the input sizes of the ARMA cell can be found in Supplementary Material. We then differentiate the loss of these outputs given the current weights back through the whole sequence, *i.e.*, the network is trained exactly as done for the LSTM, GRU, and simple RNN cell via backpropagation through time. Note that our ARMA cell also supports returning sequences, which we can use to stack cells or for training a model on multiple steps simultaneously.

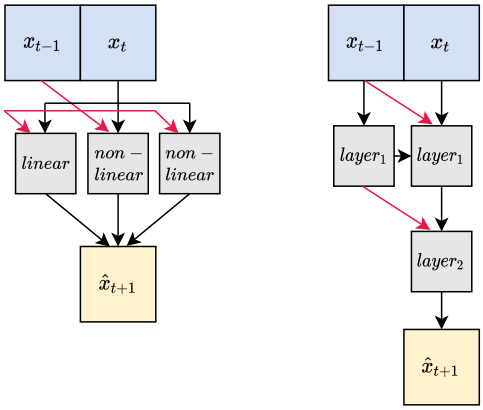


Figure 2: Visualization of an ARMA cell with multiple units representing a mixture of linear and non-linear ARMA models by using different activation functions (left) and a network with stacked ARMA cells creating a more complex model class by transforming inputs by subsequent ARMA cells (right).

4.3 Extensions

The ARMA cell in Figure 1 can be used in a modular fashion similar to an LSTM or GRU cell. In the following, we will thus present how this idea can be used to generate more complex architectures using multiple units or by stacking cells. Both options also allow bypassing the linearity assumptions of ARMA models.

Multi-unit ARMA cell. Similar to feedforward neural networks, an RNN layer can also contain multiple units. Each unit receives the same input but can capture different effects due to the random initialization of weights. The outputs of each unit are then concatenated. A multi-unit architecture allows combining different activation functions, *e.g.*, to simultaneously capture linear and non-linear effects, and is depicted in Figure 2 (left). Using a multi-unit ARMA cell thereby seamlessly provides the possibility to combine a linear with a non-linear ARMA model. We refer to models having a single hidden ARMA layer with one or more units as *ShallowARMA* models.

Stacked ARMA. To allow for higher levels of abstraction and increased model complexity, the ARMA modeling strategy does not only allow for multiple units in a single layer, but users can also stack multiple layers in series, as shown in Figure 2 (right). This is achieved by returning a sequence of lagged outputs from the previous layer. Models with more than one hidden ARMA layer are referred to as *DeepARMA* models in the following.

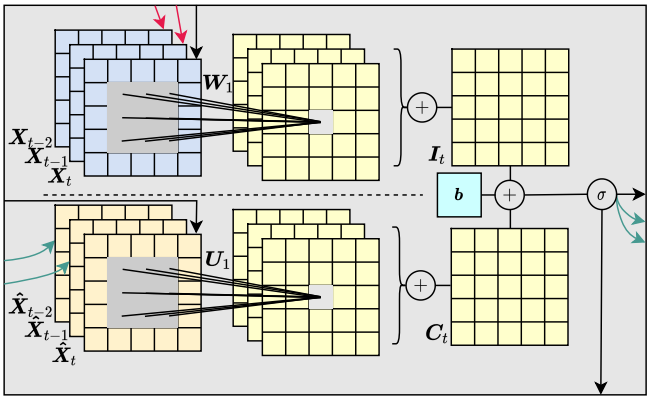


Figure 3: Exemplary visualization of a single-filter ConvARMA cell processing matrix-variate time series (with a single channel) with three lags (upper left) and matrix-variate predictions with three lags (bottom left) using convolutions and combining the results into a single matrix prediction (bottom/top right) with additional bias term b and activation function σ (center right).

4.4 ConvARMA

Similar to the ConvLSTM network [SCW+15], it is possible to model spatial dependencies and process tensor-variate time series $\boldsymbol{X}_t \in \mathbf{R}^{n_1 \times \ldots \times n_d}, n_1, \ldots, n_d \in \mathbf{N}, d \in \mathbf{N}$ by using convolution operations within an ARMA cell. The resulting ConvARMA(p, q) cell for $p, q \in \mathbf{N}_0$ and $t \in \mathbf{Z}$ is defined as

$$\begin{aligned} \boldsymbol{I}_t &= \sum_{i=1}^{p} \boldsymbol{W}_i * \boldsymbol{X}_{t-i}, \\ \boldsymbol{C}_t &= \sum_{j=1}^{q} \boldsymbol{U}_j * \hat{\boldsymbol{X}}_{t-j}, \\ \hat{\boldsymbol{X}}_t &= \sigma\left(\boldsymbol{I}_t + \boldsymbol{C}_t + \boldsymbol{b}\right), \end{aligned} \tag{4}$$

where $*$ represents the convolution operator, $\boldsymbol{W}_i \in \mathbf{R}^{k_1 \times \ldots \times k_{d-1} \times n_d \times c}, i = 1, \ldots, p$ and $\boldsymbol{U}_j \in \mathbf{R}^{k_1 \times \ldots \times k_{d-1} \times c \times c}, j = 1, \ldots, q$ are the model's kernels of size $k_1 \times \ldots \times k_{d-1}$, $\boldsymbol{b} \in \mathbf{R}^c$ is a bias term broadcasted to dimension $n_1 \times \ldots \times n_{d-1} \times c$ and σ an activation function. By convention, the last dimension of the input represents the channels, and c denotes the number of filters of the convolution. The inputs of the convolution are padded to ensure that the spatial dimensions of the prediction $\hat{\boldsymbol{X}}_t$ and the state remain unchanged. In other words, the ConvARMA cell resembles the computations of an ARMA model, but instead of simple multiplication of the time series values with scalar-valued parameters, a convolution operation is applied. Figure 3 shows an abstract visualization of the computations in a ConvARMA cell. To follow the AR(I)MA modeling logic in the spatial dimensions, a ConvARMA cell

can further incorporate spatial differences in all directions. A possible extension of the cell proposed in (4) could further be to allow for non-linear recurrent activations as done for, *e.g.*, the ConvLSTM cell.

As for the ConvLSTM or ConvGRU cell, the ConvARMA cell can be included in an autoencoder architecture for sequence-to-sequence modeling or extended to *e.g.*, allow for warping, rotation, and scaling [SGL+17].

4.5 Limitations

As for other autoregressive approaches, our approach is limited in its application if the time series are very short or if a large number of lags p is required to approximate the underlying data generating process well. We note, however, that due to the model's recurrent definition, past time points $t - i$ for $i > p$ also influence the model's predictions. It is therefore often not necessary to define a large lag value p, even if autocorrelation is high. Despite the ARMA cell's simplicity, this also shows that its predictions are not always straightforward to interpret.

5 Numerical experiments

In this section, we examine the performance of our ARMA cell in a variety of synthetic and benchmark experiments. We examine how it compares to classical time series approaches as well as to similar complex neural architectures. Note that our experiments are not designed to be a benchmark comparison with current state-of-the-art time series forecasting frameworks. These rather complex architectures include many different components, such as automated pre-processing and feature generation, and thus do not necessarily allow making a statement about the performance of a single recurrent cell therein. Instead, we aim for a comparison with other fundamental modeling building blocks [2]. Yet, in order to emphasize our cell's modularity and demonstrate its efficacy when used as part of a larger state-of-the-art network, we also present results on real-world benchmark data sets when replacing RNN cells within a DeepAR model [SFGJ20] with the ARMA cell.

Methods. For (multivariate) time series, we compare a shallow and a deep variant of the ARMA cell against the respective (V)ARMA model and neural models. For the latter, we consider LSTM, GRU, and simple RNN cells, again each in their shallow and deep variants. Hyperparameter optimization is done using a grid search with predefined parameter spaces for the number of units for all network layers and lags for ARMA-type models. All other hyperparameters of network layers are kept fixed with defaults that do not favor one or the other method. Further details on the specification of the architectures can be found in the Supplementary Material.

[2] We provide code to reproduce all experiments at https://github.com/phschiele/armacell_paper

Table 1: Comparisons of different methods (rows) and different data generating processes (columns) using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

	Univariate					Multivariate		
	ARMA	TAR	SGN	NAR	Heterosk.	VARMA	EXP	SQ
(V)ARMA	2.04±0.35	2.92±3.20	2.36±2.67	2.67±4.92	1.19±0.15	**1.00±0.03**	3.35±0.69	1.90±0.14
ShallowARMA	**1.96±0.09**	**1.09±0.12**	1.10±0.07	*1.02±0.05*	**1.11±0.06**	*1.00±0.03*	*3.14±0.74*	**1.75±0.12**
DeepARMA	1.97±0.10	*1.24±0.47*	**1.05±0.06**	1.02±0.04	*1.11±0.06*	1.01±0.03	**3.10±0.75**	*1.76±0.11*
LSTM	1.98±0.10	1.38±0.42	1.19±0.16	1.02±0.04	1.15±0.09	1.01±0.04	3.26±0.73	1.83±0.16
DeepLSTM	2.02±0.10	1.46±0.55	1.17±0.13	1.02±0.04	1.16±0.08	1.03±0.04	3.35±0.68	1.86±0.14
GRU	*1.96±0.10*	1.28±0.31	*1.09±0.08*	**1.02±0.04**	1.13±0.07	1.02±0.04	3.19±0.75	1.80±0.13
DeepGRU	1.99±0.09	1.24±0.36	1.09±0.12	1.02±0.04	1.12±0.06	1.01±0.04	3.22±0.80	1.82±0.18
Simple	1.99±0.09	1.29±0.31	1.14±0.09	1.04±0.04	1.13±0.08	1.02±0.03	3.29±0.70	1.80±0.12
DeepSimple	2.01±0.11	1.47±0.57	1.15±0.10	1.03±0.04	1.16±0.10	1.02±0.03	3.30±0.84	1.83±0.13

Performance measures. We compare time series predictions using the root mean squared error (RMSE) for both the uni- and multivariate time series forecasts. We provide further performance measures for our comparisons in the Supplementary Material.

Comparison between classical and first-order optimization. In the case where the data generating process is in fact a (V)ARMA process, we expect the classical (V)ARMA model and the ARMA cell to perform similarly, but note that the optimization using stochastic gradient descent can sometimes yield better estimations of this process and hence outperform these classical models despite having the exact same hypothesis space.

5.1 Simulation study.

We start with a variety of synthetic data examples using time series models defined in Lee et al. [LWG93]. Simulations include linear and non-linear, as well as uni- and multivariate time series. All time series are of length 1000 and split into 70% train and 30% test data. The data generating processes follow Lee et al. [LWG93] and include an ARMA process (ARMA), a threshold autoregressive model (TAR), an autoregressive time series which is transformed using the sign operation (SGN), a non-linear autoregressive series (NAR), a heteroscedastic MA process (Heteroscedastic), a vector ARMA (VARMA), a non-linear multivariate time series with quadratic lag structure (SQ), and an exponential multivariate autoregressive time series (EXP). The exact specification of the data generating processes can be found in the Supplementary Material.

Results. The results in Table 1 suggest that the ShallowARMA approach emulating an ARMA model in a neural network works well for all linear- and non-linear datasets. In terms of robustness, the lower RMSE and high standard deviation of the ARMA model on the ARMA process shows that fitting an ARMA model in a neural network with stochastic gradient descent can, in fact, be more robust than the standard software [HK08, SP10].

While the classical ARMA did match the performance of its neural counterpart it some cases, the average RMSE is worse, as it did not converge in all runs, even for the linear time series. The performance of the DeepARMA approach is slightly worse compared to the ShallowARMA in most cases. The performance of LSTM, GRU, and the Simple RNN are all similar, with all methods matching the ARMA cells in some cases, and falling slightly behind in others. As expected, the classical ARMA approach does not work well for non-linear data generating processes (TAR, SGN, NAR) and yields unstable predictions underpinned by the large standard deviations in RMSE values.

For multivariate time series results of the simulation are summarized in Table 6. The results again show that the ShallowARMA model matches the performance of the classical VARMA model for a dataset that is also based on a VARMA process. For other types of data generation, the ShallowARMA model and DeepARMA model work similarly well. Both outperform the other neural cells, which in turn yield better results than the VARMA baseline.

In summary, the findings suggest that ARMA cells work well both for simpler linear as well as non-linear data generating processes while being much more stable than a classical ARMA approach. In the Supplementary Material, we further study the empirical convergence of a single unit single hidden layer ARMA cell, which is mathematically equivalent to an ARMA model for given values of p and q, and present another comparison showing the equivalence of the ARMA cell and an Elman network.

5.2 Ablation studies

To explore the validity of our presented results, we perform a series of ablation studies. Two important influence factors on the performance of time series models are the length of the time series and the forecasting horizon, which we subsequently assess in the controlled setting of our simulation study.

5.2.1 Time series length

In order to investigate the influence of the time series length on the performance reported in previous simulations, we vary $T \in \{200, 1000, 10000\}$ and re-run the experiments reported in Table 1. The full results are given in the Supplementary Material. In summary, the rank of the different methods is similar to the aforementioned results, and ShallowARMA yields the best results in most settings. There is, however, a clear trend in that the performance differences between the different cell types become irrelevant for an increase in T. For example, for the multivariate time series study setting SQ, the ShallowARMA yields a notably better MSE for $T = 200$ compared to the DeepSimple cell (1.97 ± 0.41 vs. 3.00 ± 3.95), the performances are almost identical for $T = 10,000$ observations (1.78 ± 0.05 vs. 1.80 ± 0.07).

	ARMA	TAR	SGN	NAR	Heteroskedastic
End2End	2.021 ± 0.105	**1.134 ± 0.127**	**1.128 ± 0.055**	**1.002 ± 0.044**	**1.142 ± 0.062**
Hybrid	**2.014 ± 0.112**	1.264 ± 0.811	1.138 ± 0.051	3.009 ± 8.869	1.147 ± 0.062

Table 2: Comparisons of the End2End and Hybrid approach on different data generating processes (columns) for univariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best-performing method is highlighted in bold.

5.2.2 Forecasting horizon

Similar to the previous ablation study, we reran the experiments but now alter the forecasting horizon by comparing a one-step, 10-step, and 20-step forecast for $T = 1000$. The full results can again be found in the Supplementary Material. In the univariate case for forecasting horizons greater one, the different ARMA variations do not outperform other approaches anymore and DeepLSTM, GRU, or DeepGRU yield the best results in many cases. The performance values, however, are in most cases within one standard deviation of those by the Shallow- or DeepARMA approach. For the multivariate case, the classical VARMA model provides the best forecast for all multi-step ahead forecast scenarios, closely followed by the Shallow- and DeepARMA models.

5.3 Comparison to hybrid models

We now investigate the differences between a standard hybrid approach following [Zha03] and an end-to-end approach using the ARMA cell. The hybrid model first trains a classical model, in this case, an ARMA(2,2) model. Then, an LSTM model is fit on the residual. The final prediction is obtained as the sum of the ARMA and LSTM predictions. We also implement an end-to-end version of this model using the ARMA cell, which we refer to as End2End. Here, we train a linear ARMA cell, also specified with $p = 2$ and $q = 2$, and sum its output with the output from an LSTM cell. Both model approaches have the same hypothesis space, however, training a single model simplifies the training process and optimized the parameters jointly. We run both approaches on the simulated univariate time series, as shown in Table 2. We see that the End2End model performs similarly to the hybrid model in most cases. However, for the NAR dataset, we find that the two-step hybrid approach does not converge in all cases, leading to a substantially worse average RMSE and a high corresponding standard deviation, indicating that this approach is less robust compared to the End2End model.

5.4 Benchmarks

In order to investigate the performance of our approach for real-world time series with a potentially more complex generating process, we compare the previously defined models on various time series benchmark datasets.

Table 3: Comparison of different univariate and multivariate forecasting approaches (rows) for different datasets (columns) based on the average RMSE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

		M4	TRAFFIC	ELEC.
UNIV.	ARMA	1.58±0.00	0.98±0.00	1.19±0.00
	SHALLOWARMA	**1.57±0.01**	0.97±0.00	1.14±0.01
	DEEPARMA	*1.57±0.01*	**0.94±0.01**	**1.10±0.02**
	LSTM	1.71±0.14	*0.96±0.01*	1.15±0.07
	DEEPLSTM	1.96±0.38	0.97±0.02	1.12±0.05
	GRU	1.61±0.03	0.97±0.02	*1.11±0.02*
	DEEPGRU	1.61±0.02	0.97±0.02	1.11±0.03
	SIMPLE	1.72±0.15	1.00±0.01	1.12±0.01
	DEEPSIMPLE	1.76±0.17	1.00±0.02	1.11±0.02
MULTIV.	ARMA	1.72±0.00	*1.06±0.00*	1.46±0.00
	SHALLOWARMA	*1.68±0.01*	**1.06±0.00**	1.37±0.03
	DEEPARMA	**1.67±0.01**	1.08±0.01	1.32±0.03
	LSTM	1.92±0.15	1.15±0.01	2.07±1.12
	DEEPLSTM	2.11±0.25	1.15±0.02	1.26±0.05
	GRU	1.91±0.25	1.15±0.00	1.25±0.04
	DEEPGRU	1.88±0.12	1.15±0.01	*1.23±0.02*
	SIMPLE	1.89±0.06	1.16±0.00	1.25±0.03
	DEEPSIMPLE	1.90±0.08	1.16±0.01	**1.22±0.01**

5.4.1 Univariate and multivariate time series

We use the m4 [MSA18], traffic [YRD16], and electricity [YRD16] dataset, all openly accessible and commonly used in time series forecast benchmarks. Further background on every dataset and details on pre-processing can be found in the Supplementary Material. As all datasets come with multiple time series, we use these datasets both for testing the performance on univariate and multivariate time series. For univariate time series, this is done by training a local model for every dimension and averaging the results over the different multivariate dimensions. The multivariate comparison is based on the predictions of a single global model.

Univariate time series. Results of univariate benchmarks are summarized in Table 3. The comparisons suggest that ARMA cells, either in the shallow or deep variant, outperform on all studied datasets. The classical ARMA model is competitive for the m4 dataset, but again worse than its neural pendant on Traffic, and Electricity.

Multivariate time series. For the multivariate time series benchmarks, we observe that model performance is in general worse than when performing hyperparameter optimization

and model training for each time series individually, as done for the univariate time series benchmark. Finding architectures better suited to the individual time series seems to outweigh the additional information from observing the comovement of multiple time series simultaneously. In the comparison of different forecasting approaches for multivariate dimensions, the performance of the ARMA cells is either notably better than the other neural cells but on par with the classical ARMA model (Traffic), or outperforms all other approaches (m4). Only for the Electricity dataset, the ARMA cells yield a slightly worse MSE compared to the DeepSimple cell.

5.4.2 Integration with state-of-the-art forecasting frameworks

To demonstrate the modularity aspect of the ARMA cell, we use it to replace the LSTM cell in a DeepAR model [SFGJ20]. We then train a larger global model on the previously studied benchmark data sets for multivariate time series (as this is the application area where DeepAR model excels in performance) and examine to what extent the change in RNN cell influences the results. The experimental details can be found in Supplementary Material B.12.3. Results (Table 4) indicate that it is possible to successfully replace the

Table 4: Comparison of different DeepAR-based models (rows) for different datasets (columns) based on the average negative log-likelihood ± its standard deviation of 30 independent runs. The best performing method is highlighted in bold.

	m4	Traffic	Elec.
DeepAR ARMA single	2.444 ± 0.137	**1.323 ± 0.033**	**5.236 ± 0.398**
DeepAR LSTM single	**2.306 ± 0.056**	1.362 ± 0.054	10.236 ± 9.242
DeepAR ARMA stacked	2.513 ± 0.157	**1.332 ± 0.056**	**5.639 ± 0.461**
DeepAR LSTM stacked	**2.266 ± 0.049**	1.381 ± 0.053	9.607 ± 5.630

LSTM with an ARMA cell in the DeepAR model and to receive a similar performance. We further observe that the LSTM-based DeepAR does not always converge for the electricity dataset, indicating that the training of the ARMA cell is more robust.

6 Conclusion and Outlook

We provided a modular and flexible neural network cell to model time series in a simply parameterized fashion and as an alternative to commonly used RNN cells such as the LSTM cell. We further extended this approach to vector autoregression and autoregressive models for tensor-variate applications. Our numerical experiments show that the ARMA cell 1) performs well on univariate, multivariate, and tensor-variate time series; 2) matches or even outperforms the LSTM, GRU, and a simple RNN cell in linear and non-linear settings, and;

3) shows more robust convergence for classical ARMA formulations compared to a standalone implementation.

Outlook. An interesting future research direction is to make use of the theoretical results for ARMA models known from classical statistical literature and transfer these to the application of ARMA as a cell with multiple units or in its stacked variant. A directly available result, *e.g.*, would be last-layer uncertainty quantification (see, *e.g.*, [IKB21]) in a stacked RNN model where the last cell is an ARMA cell with one unit. Although this neglects the variance in previous layers, it allows a first assessment of the RNN's uncertainty. Further, when merging multiple linearly activated ARMA cells, the combination is an ensemble of ARMA models, for which some form of uncertainty quantification method could be derived.

References

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 265–283, 2016.

[BJRL15] G. Box, G. Jenkins, G. Reinsel, and G. Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.

[CAM91] J. Connor, L. Atlas, and D. Martin. Recurrent networks and NARMA modeling. In *Advances in Neural Information Processing Systems*, volume 4, 1991.

[CMA94] J. Connor, R. Martin, and L. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994.

[CMBB14] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[Elm90] J. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[GA78] C. Granger and A. Andersen. On the invertibility of time series models. *Stochastic Processes and their Applications*, 8(1):87–92, 1978.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[HK08] R. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008.

[HS97] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HZL+19] Z. Han, J. Zhao, H. Leung, K. Ma, and W. Wang. A review of deep learning models for time series prediction. *IEEE Sensors Journal*, 21(6):7833–7848, 2019.

[IKB21] A. Immer, M. Korzepa, and M. Bauer. Improving predictions of bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics*, pages 703–711. PMLR, 2021.

[Jor86] M. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, Hillsdale, NJ: Erlbaum, 1986.

[KB14] D. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[LWG93] T.-H. Lee, H. White, and C. Granger. Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests. *Journal of Econometrics*, 56(3):269–290, 1993.

[MSA18] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808, 2018.

[Rao81] T. Rao. On the theory of bilinear time series models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 43(2):244–255, 1981.

[RP96] I. Rivals and L. Personnaz. Black-box modeling with state-space neural networks. In *Neural Adaptive Control Technology*, pages 237–264. World Scientific, 1996.

[Sax97] H. Saxén. On the equivalence between ARMA models and simple recurrent neural networks. In *Applications of Computer Aided Time Series Modeling*, pages 281–289. Springer, 1997.

[SCW+15] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[SFGJ20] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[SGL+17] X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W. K. Wong, and W. C. Woo. Deep learning for precipitation nowcasting: A benchmark and a new model, 2017.

[SP10] S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[TB81] G. Tiao and G. Box. Modeling multiple time series with applications. *Journal of the American Statistical Association*, 76(376):802–816, 1981.

[TL09] H. Tong and K. Lim. Threshold autoregression, limit cycles and cyclical data. In *Exploration Of A Nonlinear World: An Appreciation of Howell Tong's Contributions to Statistics*, pages 9–56. World Scientific, 2009.

[TLY+19] L. Tian, X. Li, Y. Ye, P. Xie, and Y. Li. A generative adversarial gated recurrent unit model for precipitation nowcasting. *IEEE Geoscience and Remote Sensing Letters*, 17(4):601–605, 2019.

[Tri15] A. Trindade. Electricityloaddiagrams20112014, 2015.

[YRD16] H-F. Yu, N. Rao, and I. Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In *NIPS*, pages 847–855, 2016.

[Zha03] G. Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, 2003.

A Derivation of ARMA reparametrization

This shows how to rewrite the ARMA model. We start with

$$x_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j} + \varepsilon_t$$

and use the definition of $\hat{x}_t \coloneqq x_t - \varepsilon_t$ to get

$$\hat{x}_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j}.$$

We now replace each ε_{t-j} with $x_{t-j} - \hat{x}_{t-j}$

$$\hat{x}_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j (x_{t-j} - \hat{x}_{t-j}) = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{i=1}^{q} \gamma_i x_{t-i} - \sum_{j=1}^{q} \gamma_j \hat{x}_{t-j}.$$

We see that for all indices $i \leq \min(p, q)$ the common factor of x_{t-i} is $\beta_i + \gamma_i$, if $p > q$ and $i > q$ the factor is β_i and if $q > p$ and $i > p$ then the factor is γ_i, yielding the desired result.

B Additional experimental details

B.1 ARMA parameter recovery

In order to investigate if the implemented cell recovers parameters of an arbitrary ARMA model with coefficients estimated in a standard ARMA software [SP10], we simulate (V)ARMA processes for $25,000$ time steps and all possible combinations of $p, q \in \{0, 1, \ldots, 5\}$. We then train a neural network defined by a single linear ARMA cell on the data and check the convergence against the values obtained by maximum likelihood estimation. Results confirm that the ARMA cell can recover the coefficients for different values of p and q, and also in the multivariate setting. Figure 4 visualizes one exemplary learning process.

B.2 Elman parameter recovery

We now demonstrate the equivalence of a network based on the ARMA cell and an Elman network when only one MA lag is considered, *i.e.*, when the ARMA cell is restricted to $q = 1$. For this, we take the ARMA(1,1) time series process used also in our simulation studies and fit linearly activated single-unit models based on both the ARMA cell and the Elman network. As shown in Figure 5, both models converge to the same parameters when applying the ARMA coefficient reparametrization as in (3).

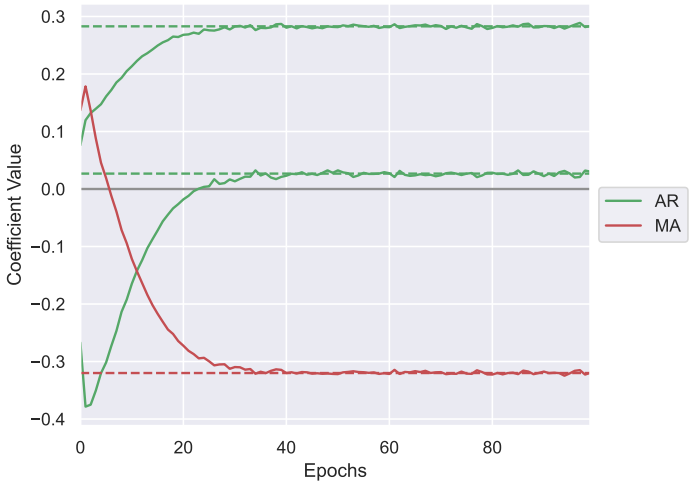


Figure 4: Exemplary optimization paths for a single linear ARMA(2,1) cell using stochastic gradient descent. After around 30 iterations, the models converge to the maximum likelihood coefficients.

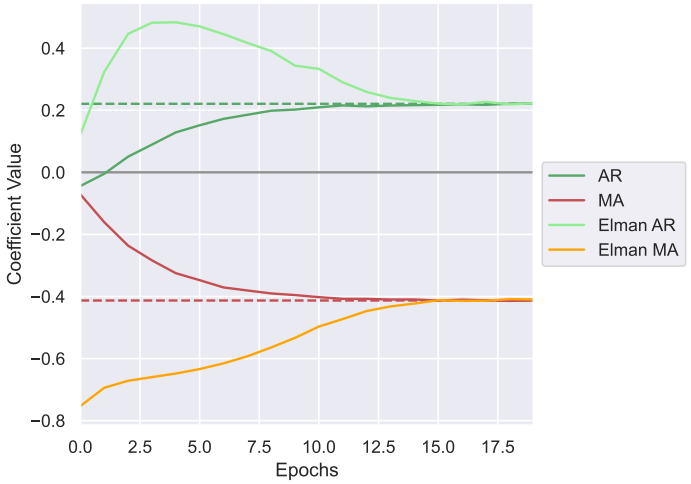


Figure 5: The ARMA cell and the Elman model converging to the same coefficients for an ARMA(1,1) process.

B.3 Simulation study

Comparison between classical and first-order optimization. In the case where the data generating process is in fact a (V)ARMA process, we expect the classical (V)ARMA model and the ARMA cell to perform similarly, but note that the optimization using stochastic gradient descent can sometimes yield better estimations of this process and hence outperform these classical models despite having the exact same hypothesis space.

We start with a variety of synthetic data examples using time series models defined in [LWG93]. Simulations include linear and non-linear, as well as uni- and multivariate time series. All time series are of length 1000 and split into 70% train and 30% test data. The data generating processes follow [LWG93] and include an ARMA process (ARMA), a threshold autoregressive model (TAR), an autoregressive time series which is transformed using the sign operation (SGN), a non-linear autoregressive series (NAR), a heteroscedastic MA process (Heteroscedastic), a vector ARMA (VARMA), a non-linear multivariate time series with quadratic lag structure (SQ), and an exponential multivariate autoregressive time series (EXP). The exact specification of the data generating processes are given below.

Description of simulated data generating processes. All error terms are a Gaussian white noise $\varepsilon_t \sim \mathcal{N}(0, 1)$. The data generating processes were defined as follows:

- ARMA(2,1)
$$x_t = 0.1x_{t-1} + 0.3x_{t-2} - 0.4\varepsilon_{t-1} + \varepsilon_t$$
- Threshold autoregressive (TAR)
$$x_t = \begin{cases} 0.9x_{t-1} + \varepsilon_t & \text{for } |x_{t-1}| \leq 1, \\ -0.3x_{t-1} + \varepsilon_t & \text{for } |x_{t-1}| > 1 \end{cases}$$
- Sign autoregressive (SGN)
$$x_t = sgn(x_{t-1}) + \varepsilon_t,$$
with
$$sgn(x) = \begin{cases} 1 \text{ for } x > 0, \\ 0 \text{ for } x = 0, \\ -1 \text{ for } x < 0 \end{cases}$$
- Non-linear autoregressive (NAR)
$$x_t = \frac{0.7|x_{t-1}|}{|x_{t-1} + 2|} + \varepsilon_t$$
- Heteroskedastic MA(2)
$$x_t = \varepsilon_t - 0.4\varepsilon_{t-1} + 0.3\varepsilon_{t-2} + 0.5\varepsilon_t\varepsilon_{t-2}$$

Table 5: Comparisons of different methods (rows) and different data generating processes (columns) for univariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

MODEL	ARMA	TAR	SGN	NAR	HETEROSKEDASTIC
ARMA	2.04±0.35	2.92±3.20	2.36±2.67	2.67±4.92	1.19±0.15
SHALLOWARMA	**1.96±0.09**	**1.09±0.12**	1.10±0.07	*1.02±0.05*	**1.11±0.06**
DEEPARMA	1.97±0.10	*1.24±0.47*	**1.05±0.06**	1.02±0.04	*1.11±0.06*
LSTM	1.98±0.10	1.38±0.42	1.19±0.16	1.02±0.04	1.15±0.09
DEEPLSTM	2.02±0.10	1.46±0.55	1.17±0.13	1.02±0.04	1.16±0.08
GRU	*1.96±0.10*	1.28±0.31	*1.09±0.08*	**1.02±0.04**	1.13±0.07
DEEPGRU	1.99±0.09	1.24±0.36	1.09±0.12	1.02±0.04	1.12±0.06
SIMPLE	1.99±0.09	1.29±0.31	1.14±0.09	1.04±0.04	1.13±0.08
DEEPSIMPLE	2.01±0.11	1.47±0.57	1.15±0.10	1.03±0.04	1.16±0.10

- VARMA

$$\begin{bmatrix} x_{t,1} \\ x_{t,2} \end{bmatrix} = \begin{bmatrix} 0.1 & -0.2 \\ -0.2 & 0.1 \end{bmatrix} \begin{bmatrix} x_{t-1,1} \\ x_{t-1,2} \end{bmatrix} + \begin{bmatrix} -0.4 & 0.2 \\ 0.2 & -0.4 \end{bmatrix} \begin{bmatrix} \varepsilon_{t-1,1} \\ \varepsilon_{t-1,2} \end{bmatrix} + \begin{bmatrix} \varepsilon_{t,1} \\ \varepsilon_{t,2} \end{bmatrix}$$

- Square multivariate (SQ)

$$\begin{aligned} x_{t,1} &= 0.6x_{t-1} + \varepsilon_{t,1} \\ x_{t,2} &= x_{t,1}^2 + \varepsilon_{t,2} \end{aligned}$$

- Exponential multivariate (EXP)

$$\begin{aligned} x_{t,1} &= 0.6x_{t-1} + \varepsilon_{t,1} \\ x_{t,2} &= \exp(x_{t,1}) + \varepsilon_{t,2} \end{aligned}$$

For the multivariate time series (VARMA, SQ, EXP), the second index of $x_{t,i}$, $i \in \{1, 2\}$, refers to the individual components.

Results. The results in Table 5 suggest that the ShallowARMA approach emulating an ARMA model in a neural network works well for all linear- and non-linear datasets. In terms of robustness, the lower RMSE and high standard deviation of the ARMA model on the ARMA process shows that fitting an ARMA model in a neural network with stochastic gradient descent can, in fact, be more robust than the standard software [HK08, SP10]. While the classical ARMA did match the performance of its neural counterpart it some cases, the average RMSE is worse, as it did not converge in all runs, even for the linear

Table 6: Comparisons of different methods (rows) and different data generating processes (columns) for multivariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

	VARMA	EXP	SQ
VARMA	**1.00±0.03**	3.35±0.69	1.90±0.14
ShallowARMA	*1.00±0.03*	*3.14±0.74*	**1.75±0.12**
DeepARMA	1.01±0.03	**3.10±0.75**	*1.76±0.11*
LSTM	1.01±0.04	3.26±0.73	1.83±0.16
DeepLSTM	1.03±0.04	3.35±0.68	1.86±0.14
GRU	1.02±0.04	3.19±0.75	1.80±0.13
DeepGRU	1.01±0.04	3.22±0.80	1.82±0.18
SIMPLE	1.02±0.03	3.29±0.70	1.80±0.12
DeepSimple	1.02±0.03	3.30±0.84	1.83±0.13

time series. The performance of the DeepARMA approach is slightly worse compared to the ShallowARMA in most cases. The performance of LSTM, GRU, and the simple RNN are all similar, with all methods matching the ARMA cells in some cases, and falling slightly behind in others. As expected, the classical ARMA approach does not work well for non-linear data generating processes (TAR, SGN, NAR) and yields unstable predictions underpinned by the large standard deviations in RMSE values.

For multivariate time series results of the simulation are summarized in Table 6. The results again show that the ShallowARMA model matches the performance of the classical VARMA model for a dataset that is also based on a VARMA process. For other types of data generation, the ShallowARMA model and DeepARMA model work similarly well. Both outperform the other neural cells, which in turn yield better results than the VARMA baseline.

In summary, findings suggest that ARMA cells work well for simpler linear and non-linear data generating processes while being much more stable than a classical ARMA approach.

B.4 Ablation studies

To explore the validity of our presented results, we perform a series of ablation studies. Two important influence factors on the performance of time series models are the length of the time series and the forecasting horizon, which we subsequently assess in the controlled setting of our simulation study.

B.4.1 Time series length

In order to investigate the influence of the time series length on the performance reported in previous simulations, we vary $T \in \{200, 1000, 10000\}$ and re-run the experiments reported in Table 5 and 6. The results are given in Tables 7 and 8. In summary, the rank of the

different methods is similar to the aforementioned results, and ShallowARMA yields the best results in most settings. There is, however, a clear trend in that the performance differences between the different cell types become irrelevant for an increase in T. For example, for the multivariate time series study setting SQ, the ShallowARMA yields a notably better MSE for $T = 200$ compared to the DeepSimple cell (1.97 ± 0.41 vs. 3.00 ± 3.95), the performances are almost identical for $T = 10,000$ observations (1.78 ± 0.05 vs. 1.80 ± 0.07).

	ARMA	TAR	SGN	NAR	Hetero
ARMA	2.19±0.97	1.77±0.86	1.80±1.54	1.04±0.15	1.38±0.56
ShallowARMA	**2.00±0.21**	**1.51±0.46**	**1.29±0.10**	*1.03±0.13*	**1.17±0.15**
DeepARMA	*2.04±0.22*	1.71±0.43	*1.32±0.16*	1.03±0.13	*1.21±0.17*
LSTM	2.08±0.24	2.05±0.98	1.40±0.16	1.03±0.14	1.22±0.17
DeepLSTM	2.09±0.24	2.03±0.51	1.46±0.17	1.03±0.13	1.24±0.17
GRU	2.08±0.24	*1.71±0.49*	1.34±0.16	1.03±0.13	1.23±0.17
DeepGRU	2.08±0.22	1.77±0.43	1.38±0.15	**1.03±0.13**	1.23±0.18
SIMPLE	2.32±0.69	2.14±0.91	1.43±0.22	1.16±0.53	1.34±0.36
DeepSIMPLE	2.30±0.87	2.34±1.47	1.70±0.81	1.10±0.29	1.25±0.18

(a) 200 observations

	ARMA	TAR	SGN	NAR	Hetero
ARMA	2.07±0.29	1.98±1.98	1.71±1.80	1.94±3.46	1.48±1.60
ShallowARMA	**2.02±0.11**	**1.12±0.14**	1.12±0.05	**1.00±0.05**	**1.11±0.06**
DeepARMA	*2.04±0.11*	*1.18±0.29*	**1.07±0.05**	*1.00±0.05*	*1.12±0.06*
LSTM	2.06±0.12	1.22±0.25	1.16±0.10	1.00±0.05	1.15±0.06
DeepLSTM	2.11±0.14	1.18±0.20	1.16±0.11	1.00±0.05	1.17±0.09
GRU	2.06±0.13	1.29±0.30	1.14±0.09	1.00±0.05	1.13±0.06
DeepGRU	2.08±0.13	1.21±0.26	*1.11±0.09*	1.00±0.05	1.13±0.07
SIMPLE	2.07±0.13	1.31±0.25	1.18±0.11	1.01±0.05	1.15±0.08
DeepSIMPLE	2.09±0.12	1.43±0.45	1.16±0.11	1.01±0.06	1.15±0.08

(b) 1000 observations

	ARMA	TAR	SGN	NAR	Hetero
ARMA	2.04±0.23	1.60±0.94	1.94±3.10	1.05±0.14	1.16±0.11
ShallowARMA	**2.00±0.03**	**1.01±0.01**	1.06±0.03	**1.00±0.01**	1.06±0.02
DeepARMA	2.00±0.04	1.06±0.23	**1.02±0.03**	*1.00±0.01*	1.05±0.02
LSTM	*2.00±0.03*	1.06±0.23	*1.03±0.01*	1.00±0.01	1.04±0.02
DeepLSTM	2.02±0.06	1.11±0.33	1.06±0.12	1.01±0.01	1.04±0.04
GRU	2.01±0.04	1.19±0.45	1.07±0.12	1.00±0.01	*1.04±0.02*
DeepGRU	2.01±0.04	1.19±0.44	1.04±0.10	1.00±0.01	**1.03±0.02**
SIMPLE	2.02±0.04	*1.02±0.02*	1.08±0.08	1.00±0.01	1.06±0.02
DeepSIMPLE	2.02±0.06	1.15±0.37	1.04±0.08	1.00±0.01	1.05±0.02

(c) 10000 observations

Table 7: Comparisons of different methods (rows) and different data generating processes (columns) across different time series lengths (200 (a), 1000 (b), 10000 (c)) for univariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

	VARMA 200	EXP 200	SQ 200
VARMA	1.02±0.08	3.11±1.67	2.07±0.44
ShallowARMA	**1.01±0.06**	**2.97±1.68**	**1.97±0.41**
DeepARMA	1.02±0.07	*3.02±1.68*	*2.04±0.47*
LSTM	*1.01±0.06*	3.24±1.77	2.17±0.57
DeepLSTM	1.02±0.06	3.12±1.70	2.16±0.56
GRU	1.02±0.07	3.12±1.72	2.06±0.47
DeepGRU	1.02±0.06	3.15±1.73	2.14±0.55
SIMPLE	1.09±0.13	3.60±2.04	2.79±1.59
DeepSIMPLE	1.06±0.09	3.32±1.83	3.00±3.95

(a) 200 observations

	VARMA 1k	EXP 1k	SQ 1k
VARMA	**1.00±0.03**	2.92±0.61	1.88±0.17
ShallowARMA	*1.01±0.04*	**2.70±0.63**	**1.75±0.16**
DeepARMA	1.01±0.04	*2.71±0.63*	*1.75±0.15*
LSTM	1.02±0.04	2.80±0.65	1.81±0.17
DeepLSTM	1.04±0.04	2.86±0.63	1.84±0.17
GRU	1.02±0.04	2.76±0.61	1.78±0.17
DeepGRU	1.02±0.04	2.81±0.66	1.81±0.18
SIMPLE	1.03±0.04	3.18±1.67	1.82±0.19
DeepSIMPLE	1.04±0.04	2.89±0.72	1.85±0.18

(b) 1000 observations

	VARMA 10k	EXP 10k	SQ 10k
VARMA	**1.00±0.01**	3.04±0.39	1.94±0.06
ShallowARMA	*1.00±0.01*	**2.80±0.41**	*1.78±0.05*
DeepARMA	1.00±0.01	*2.81±0.41*	**1.78±0.05**
LSTM	1.00±0.01	2.84±0.46	1.79±0.06
DeepLSTM	1.01±0.01	2.90±0.46	1.81±0.07
GRU	1.00±0.01	2.82±0.41	1.79±0.06
DeepGRU	1.00±0.01	2.87±0.43	1.80±0.06
SIMPLE	1.00±0.01	2.83±0.42	1.79±0.05
DeepSIMPLE	1.01±0.01	2.86±0.41	1.80±0.07

(c) 10000 observations

Table 8: Comparisons of different methods (rows) and different data generating processes (columns) across different time series lengths (200 (a), 1000 (b), 10000 (c)) for multivariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

B.4.2 Forecasting horizon

Similar to the previous ablation study, we reran the experiments but now alter the forecasting horizon by comparing a one-step, 10-step, and 20-step forecast for $T = 1000$. The results can be found in Table 9 and 10. In the univariate case for forecasting horizons greater one, the different ARMA variations do not outperform other approaches anymore and DeepLSTM, GRU, or Deep GRU yield the best results in many cases. The performance values, however, are in most cases within one standard deviation of those by the Shallow- or DeepARMA approach. For the multivariate case, the classical VARMA model provides the best forecast for all multi-step ahead forecast scenarios, closely followed by the Shallow- and DeepARMA models.

B.5 Description of benchmark datasets

M4. Stemming from the Makridakis Competitions [MSA18] (see https://en.wikipedia.org/wiki/Makridakis_Competitions for more information), the M4 dataset contains 414 time series of hourly data. Every time series has a different starting point and a length of 748 hours. To allow for multivariate prediction, we take a subset of ten times series starting at the same time and ending at the same time. We further take differences with a period of one and 24 hours to improve stationarity and reduce seasonal effects, respectively.

Traffic. The traffic dataset can be downloaded from https://archive.ics.uci.edu/ml/datasets/PEMS-SF. It consists of 963 car lane occupancy rates with values between 0 and 1 taken from freeways in the San Francisco Bay Area. Time series start on the first of January 2008 and last until March 30 2009 with an observation frequency of 10 minutes. To condense the information, an hourly aggregation is used [YRD16], yielding time series of length 10,560. We use the first ten time series and observations until '2008-06-22 23:00:00', yielding a total of 4,167 observations per lane. We further apply seasonal differencing with a seasonal period of 24 hours and take first differences to reduce non-stationary behavior.

Electricity. The electricity dataset can be downloaded from https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014. The dataset consists of electricity consumption (kWh) time series of 370 customers [Tri15]. Values correspond to electricity usage in a frequency of 15 minutes. In our benchmarks, we aggregate the values to hourly consumption (see also [YRD16] for justification of this approach). We use a subset of ten customers and a time range from '2014-01-01 00:00:00' to '2014-09-07 23:00:00', yielding a total of 6,000 observations per customer . We further apply seasonal differencing with a period of 24 hours to reduce seasonal effects and take the first differences for stationarity reasons.

B.6 Architectures and search space

For uni- and multivariate time series, all neural networks contain one to two RNN layers of the respective RNN cell, yielding the shallow and deep versions of the models, respectively. The cells in each layer contain one to five units with a rectified linear activation function. In the ShallowARMA model, one cell is activated linearly as shown in Figure 2, resembling a hybrid model. A final fully connected layer with linear activation and appropriate output shape is used to match the dimensions of the time series. The lag values p and q are chosen from the interval $[1, 4]$. The loss function of all models is the mean squared error function. For training, the Adam [KB14] optimizer is used in combination with an early stopping callback to prevent overfitting. For all other model properties, the default values are used. For tensor-variate time series, batch normalization layers are added between the RNN layers, and adaptive learning is added to improve convergence. Each layer contains 64 filters, so a 2D convolution is added to reduce the number of channels appropriately.

B.7 Computational environment

All experiments and benchmarks were carried out on an internal cluster. Uni- and multivariate time series were trained on a server with 10 vCPUs, running on an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz physical CPU and 48Gb allocated memory. Tensor-variate time series were trained on a server with 16 vCPUS, running on an Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 32Gb allocated memory, and a Nvidia GeForce RTX 2080 Ti (11Gb).

B.8 Additional details on experimental setup

We now give additional details about our experimental setup, clarifying the preprocessing steps as well as the training routines.

B.9 General setup and optimization

Across all simulations and benchmarks, we use the Adam [KB14] optimizer with a learning rate of 1e-3, and momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use a batch size of 32, early stopping with patience 10 iterations, and run the model for a maximum of 100 epochs.

B.10 Competitor architectures

The following abbreviations are used for the methods we compare the ARMA cell against:

- LSTM/GRU/SIMPLE: A single LSTM/GRU/SIMPLE cell with ReLU activation function and a sigmoid recurrent activation function. The kernel, recurrent, and bias initializers are chosen to be uniform, orthogonal, and zero, respectively. The number of units is chosen via hyperparameter optimization in the range of one to five.
- DeepLSTM/DeepGRU/DeepSIMPLE: This setup stacks two of the cells in their non-deep counterparts, with the first layer set to return sequences.

B.11 Input and out formats of RNNs

The input of the RNN cells are subsequences of the time series $\boldsymbol{X}$. For a sequence length s, the input shape for a non-ARMA cell RNN is given by $s \times k$. The input of the ARMA cell has an additional dimension for the lagged inputs, *i.e.*, $s \times k \times p$.

The non-ARMA cell RNN with u units returns an $s \times u$ time series. In order to allow the ARMA cell to be stacked, we have an additional dimension in the output. Specified by the keyword argument `return_lags`, we either return $s \times (d * u) \times 1$ if `False`, or $s \times (d * u) \times q$ if `True`. The factor d ensures that a single unit ARMA cell returns the same number of dimensions as its input, allowing it to capture a classical VARMA model.

B.12 Specific setups

B.12.1 Simulation study

For the simulated time series, no preprocessing was necessary, as the data generating processes are all stationary and also of the same magnitude.

B.12.2 Benchmarks

For the real-world datasets, preprocessing was performed to improve the convergence of all models. In particular, we first apply differencing to remove trends in the time series. We define Δ_k to be the difference to the k'th lag of the time series. For M4, traffic, and electricity, we use $\Delta_1\Delta_{24}\boldsymbol{X}$. We then standardize the resulting time series with their empirical mean and variance.

B.12.3 Integration with state-of-the-art forecasting frameworks

For the DeepAR benchmark, we use a larger dataset by looking at the first 100 columns of the M4, traffic, and electricity datasets.

Architectures. We use the following two types of DeepAR architectures:

- SINGLE: The SINGLE DeepAR model consists of one RNN block (either LSTM or ARMA) with $4(1 + log(d))$ units for the LSTM and one unit for the ARMA cell, a dropout rate of 0.2, and returns a sequence that is further processed by a fully-connected layer with $s(1+log(d))$ number of units where $s \in \{1, 4\}$ is a hyperparameter and tanh activation. The resulting output is then fed into a Gaussian distribution layer that multiplies the number of input units by two to define both a mean and standard deviation for all output units by multiplying the inputs with a weight matrix of respective size. For the ARMA cell-based DeepAR model, we have the additional hyperparameters p and q, which we optimize over the range from 1 to 4, again only considering $p = q$ for computational reasons.

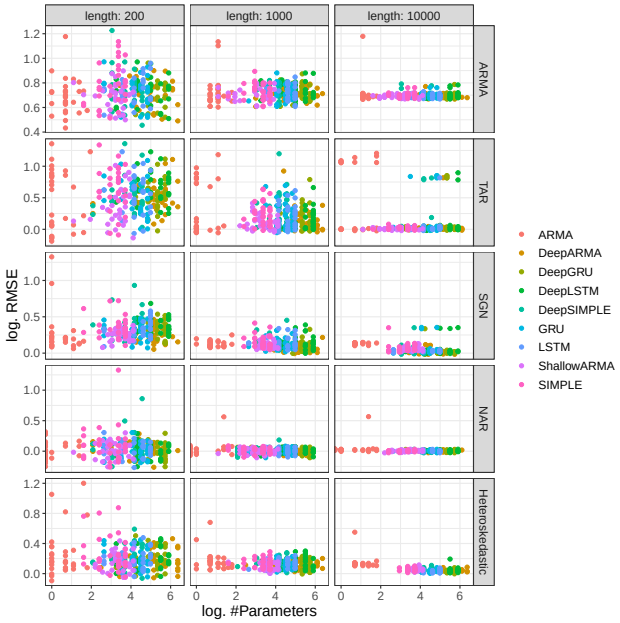


Figure 6: Logarithmic RMSE values (y-axis) for different models (colors) and their respective parameter numbers (x-axis) separated by the time series length (columns) and data sets (rows).

- STACKED: The STACKED model uses the same architecture as SINGLE but combines two of the RNN cells specified by the SINGLE model.

B.13 Investigation of parameter influence

In the following, we investigate the influence of the number of parameters on the performance and provide a summary of the resulting hyperparameter optimization results.

Relation between number of parameters and performance. In Figure 6 we compare the logarithmic RMSE values of different models to the (logarithmic) number of parameters in the case of a one-step forecasting horizon. Whereas performance values are very similar for all models for the linear time series (ARMA, TAR), the ShallowARMA model yields better results than the SIMPLE, GRU, and LSTM models while having fewer parameters compared to the latter two architectures. The DeepARMA model often yields a similar or larger number of parameters compared to the other deep architectures, while also yielding smaller RMSE values, in particular for non-linear time series (NAR, SGN).

A similar result can be observed for different forecasting horizons (Figure 7). When comparing only the two ARMA-variants (Figure 8), the improvement using a deep instead of a shallow ARMA becomes apparent, but – as expected – only on the non-linear data sets

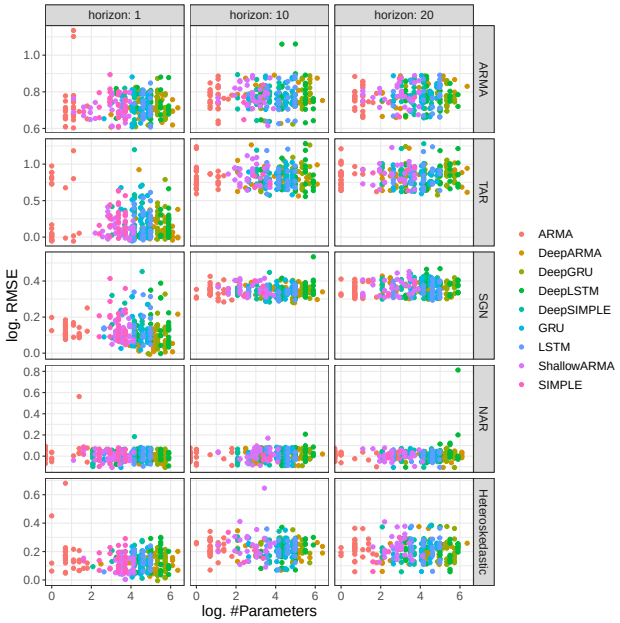


Figure 7: Logarithmic RMSE values (y-axis) for different models (colors) and their respective parameter numbers (x-axis) separated by the forecasting horizon (columns) and data sets (rows).

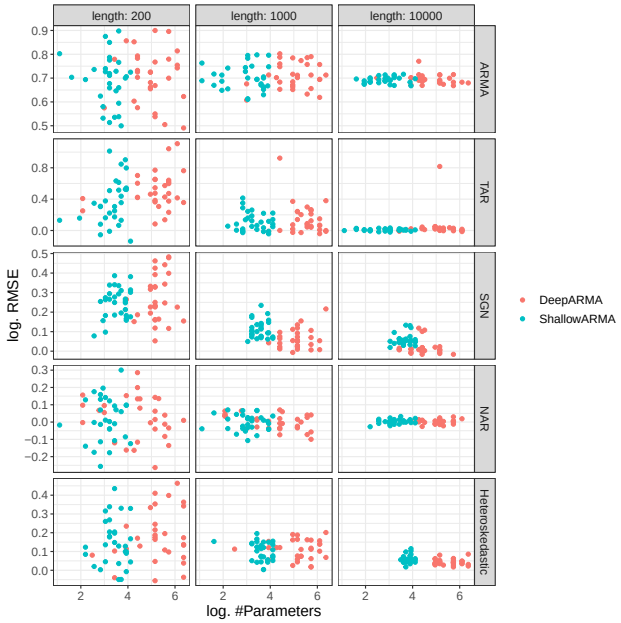


Figure 8: Logarithmic RMSE values (y-axis) for the two different ARMA models (colors) and their respective parameter numbers (x-axis) separated by the time series length (columns) and data sets (rows).

(TAR, SGN) and larger time series (1000, 10000).

We now further investigate the chosen number of lags p and q in the hyperparameter optimization routine. Figure 9 summarizes the result of this analysis by plotting the number of runs in which a given value for p or q was chosen. As we have the simplifying assumption of only considering $p = q$ in the ShallowARMA and DeepARMA models, only one bar is given for each model. We find that the classical ARMA model chooses the correct value of $p = 2$ in the majority of cases. We further find that the DeepARMA model tends to use a lower number of lags compared to the ShallowARMA model, indicating that the second layer facilitates more complex model spaces that are otherwise captured by a longer lag structure.

Influence of the non-linearity. To assess the influence of the non-linear activation, we check how often a purely linear activation was chosen by the hyperparameter optimization for the ShallowARMA model in the univariate simulations. We find that the hyperparameter optimization opted to use non-linear activations in almost all cases, only using linear models 3 times for NAR, 1 time for Heteroskedastic, and 0 times for TAR and SGN. For the ARMA time series, however, a purely linear activation was selected more often, in 9 out of 30 cases. This indicates that the non-linearity contributes to the performance improvement of the cell. By contrast, we did not find substantial differences based on the number of units, as shown

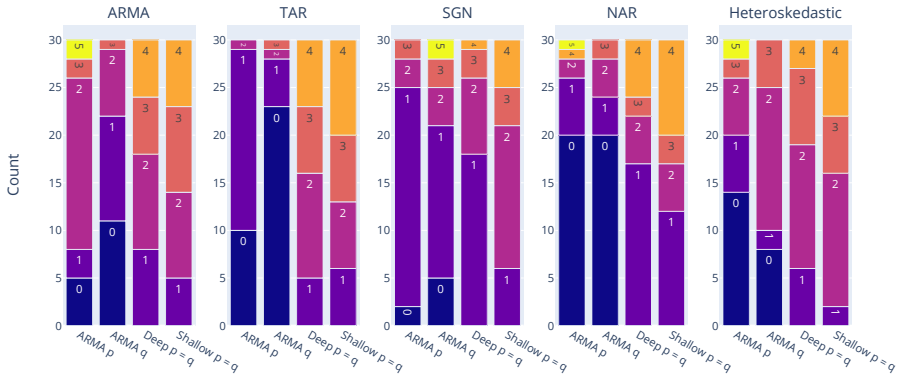


Figure 9: Aggregated lag choices (number of counts on the y-axis for p and q for the classical ARMA model, as well as the ShallowARMA and DeepARMA for each of the 5 data sets (different plots)).

in Figures 6-8. Note that while these figures show the number of parameters, this implicitly groups them by the number of units.

B.14 Additional simulation and benchmark results

In the following, we provide additional results on numerical experiments by including comparisons based on the mean absolute error (MAE).

Results. The results for simulated data suggest that either the ShallowARMA or DeepARMA cell perform best in most cases while on par with the GRU cell for the ARMA and NAR dataset. For the simulated multivariate time series, none of the existing neural methods outperforms the ARMA cells. For the time series benchmark datasets, the ARMA approaches outperform all other RNN approaches on M4 and Traffic. On the Electricity dataset the ARMA cells remain competitive for the univariate case, but yield larger RMSE values compared to GRU and Simple in the multivariate setting.

Overall the rankings of methods do not change notably when using the MAE instead of the RMSE as comparison measure.

	ARMA	TAR	SGN	NAR	Hetero
ARMA	2.07±0.29	1.98±1.98	1.71±1.80	1.94±3.46	1.48±1.60
ShallowARMA	**2.02±0.11**	**1.12±0.14**	1.12±0.05	**1.00±0.05**	**1.11±0.06**
DeepARMA	*2.04±0.11*	*1.18±0.29*	**1.07±0.05**	*1.00±0.05*	*1.12±0.06*
LSTM	2.06±0.12	1.22±0.25	1.16±0.10	1.00±0.05	1.15±0.06
DeepLSTM	2.11±0.14	1.18±0.20	1.16±0.11	1.00±0.05	1.17±0.09
GRU	2.06±0.13	1.29±0.30	1.14±0.09	1.00±0.05	1.13±0.06
DeepGRU	2.08±0.13	1.21±0.26	*1.11±0.09*	1.00±0.05	1.13±0.07
SIMPLE	2.07±0.13	1.31±0.25	1.18±0.11	1.01±0.05	1.15±0.08
DeepSIMPLE	2.09±0.12	1.43±0.45	1.16±0.11	1.01±0.06	1.15±0.08

(a) 1 step ahead

	ARMA	TAR	SGN	NAR	Hetero
ARMA	**2.12±0.06**	2.58±0.64	1.44±0.08	1.03±0.04	1.28±0.06
ShallowARMA	2.16±0.14	2.33±0.34	1.41±0.04	1.00±0.05	1.24±0.08
DeepARMA	2.17±0.13	*2.26±0.33*	*1.41±0.04*	**1.00±0.05**	1.24±0.08
LSTM	2.17±0.13	2.48±1.22	1.41±0.04	1.00±0.05	1.24±0.08
DeepLSTM	2.16±0.13	2.30±0.37	**1.41±0.04**	1.00±0.05	*1.24±0.08*
GRU	2.17±0.13	**2.25±0.30**	1.41±0.05	1.00±0.05	1.24±0.08
DeepGRU	*2.16±0.13*	2.29±0.35	1.41±0.04	*1.00±0.05*	**1.24±0.08**
SIMPLE	2.18±0.14	2.28±0.28	1.42±0.05	1.01±0.06	1.29±0.15
DeepSIMPLE	2.22±0.23	2.34±0.42	1.43±0.07	1.02±0.06	1.25±0.08

(b) 10 step ahead

	ARMA	TAR	SGN	NAR	Hetero
ARMA	**2.10±0.07**	3.25±1.38	1.43±0.08	1.02±0.02	1.24±0.05
ShallowARMA	2.17±0.13	2.33±0.32	**1.42±0.05**	1.00±0.04	1.24±0.09
DeepARMA	2.17±0.13	2.32±0.33	*1.42±0.05*	1.00±0.04	*1.23±0.09*
LSTM	2.17±0.13	2.59±1.45	1.43±0.06	1.00±0.04	1.24±0.09
DeepLSTM	*2.17±0.13*	2.33±0.35	1.43±0.05	**1.00±0.04**	**1.23±0.09**
GRU	2.17±0.13	**2.32±0.31**	1.44±0.06	1.01±0.04	1.24±0.09
DeepGRU	2.17±0.13	*2.32±0.32*	1.43±0.06	*1.00±0.04*	1.24±0.09
SIMPLE	2.18±0.14	2.44±0.45	1.44±0.06	1.01±0.04	1.26±0.10
DeepSIMPLE	2.18±0.13	2.42±0.43	1.44±0.07	1.07±0.23	1.24±0.09

(c) 20 step ahead

Table 9: Comparisons of different methods (rows) and different data generating processes (columns) across different forecasting horizons (1 (a), 10 (b), 20 (c)) for univariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

	VARMA	EXP	SQ
VARMA	**1.00±0.03**	2.92±0.61	1.88±0.17
ShallowARMA	*1.01±0.04*	**2.70±0.63**	**1.75±0.16**
DeepARMA	1.01±0.04	*2.71±0.63*	*1.75±0.15*
LSTM	1.02±0.04	2.80±0.65	1.81±0.17
DeepLSTM	1.04±0.04	2.86±0.63	1.84±0.17
GRU	1.02±0.04	2.76±0.61	1.78±0.17
DeepGRU	1.02±0.04	2.81±0.66	1.81±0.18
SIMPLE	1.03±0.04	3.18±1.67	1.82±0.19
DeepSIMPLE	1.04±0.04	2.89±0.72	1.85±0.18

(a) 1 step ahead

	VARMA	EXP	SQ
VARMA	**1.05±0.03**	**2.98±0.80**	**1.89±0.19**
ShallowARMA	1.05±0.03	3.03±0.81	*1.91±0.19*
DeepARMA	1.05±0.04	*3.02±0.81*	1.91±0.20
LSTM	1.05±0.04	3.08±0.83	1.93±0.20
DeepLSTM	*1.05±0.04*	3.06±0.81	1.93±0.21
GRU	1.05±0.03	3.06±0.82	1.92±0.20
DeepGRU	1.05±0.04	3.06±0.83	1.91±0.19
SIMPLE	1.06±0.04	3.09±0.83	1.98±0.22
DeepSIMPLE	1.06±0.04	3.19±1.01	1.95±0.20

(b) 10 step ahead

	VARMA	EXP	SQ
VARMA	**1.05±0.04**	**2.88±0.88**	**1.97±0.20**
ShallowARMA	1.06±0.04	*2.91±0.90*	*2.00±0.20*
DeepARMA	1.05±0.04	2.91±0.89	2.00±0.20
LSTM	1.05±0.04	2.93±0.90	2.04±0.27
DeepLSTM	*1.05±0.04*	2.97±0.95	2.01±0.20
GRU	1.05±0.04	2.98±1.06	2.01±0.20
DeepGRU	1.05±0.04	2.93±0.88	2.01±0.21
SIMPLE	1.06±0.04	3.06±0.93	2.09±0.27
DeepSIMPLE	1.06±0.04	2.98±0.89	2.11±0.50

(c) 20 step ahead

Table 10: Comparisons of different methods (rows) and different data generating processes (columns) across different forecasting horizons (1 (a), 10,(b) 20 (c)) for multivariate time series using the average RMSE ± the standard deviation of 30 independent runs. The best performing method is highlighted in bold, the second-best in italics.

Table 11: Comparisons of different methods (rows) and different data generating processes (columns) for univariate time series using the average MAE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

model	ARMA	TAR	SGN	NAR	Heteroskedastic
ARMA	1.62±0.29	2.39±2.66	1.98±2.35	2.28±4.42	0.95±0.11
ShallowARMA	**1.55±0.07**	**0.84±0.06**	0.88±0.06	0.81±0.04	**0.88±0.05**
DeepARMA	1.57±0.08	0.96±0.38	**0.84±0.05**	0.81±0.04	*0.88±0.05*
LSTM	1.57±0.07	1.08±0.36	0.96±0.15	*0.81±0.04*	0.92±0.08
DeepLSTM	1.60±0.08	1.12±0.42	0.94±0.12	0.81±0.04	0.92±0.07
GRU	*1.56±0.09*	0.98±0.22	0.88±0.07	**0.81±0.04**	0.90±0.06
DeepGRU	1.58±0.07	*0.95±0.26*	*0.87±0.10*	0.81±0.04	0.90±0.05
Simple	1.58±0.08	0.99±0.22	0.91±0.08	0.83±0.04	0.90±0.06
DeepSimple	1.60±0.08	1.15±0.46	0.92±0.08	0.82±0.04	0.92±0.08

Table 12: Comparisons of different methods (rows) and different data generating processes (columns) for multivariate time series using the average MAE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

model	VARMA	EXP	SQ
VARMA	**0.80±0.02**	1.61±0.13	1.22±0.07
ShallowARMA	*0.80±0.03*	*1.48±0.13*	**1.19±0.06**
DeepARMA	0.81±0.03	**1.48±0.13**	*1.20±0.06*
LSTM	0.81±0.03	1.55±0.13	1.25±0.09
DeepLSTM	0.82±0.03	1.63±0.16	1.29±0.10
GRU	0.81±0.03	1.53±0.14	1.23±0.07
DeepGRU	0.81±0.03	1.54±0.13	1.23±0.09
SIMPLE	0.82±0.02	1.56±0.15	1.23±0.07
DeepSimple	0.82±0.03	1.56±0.17	1.25±0.08

Table 13: Comparison of different univariate and multivariate forecasting approaches (rows) for different datasets (columns) based on the average MAE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

		M4	TRAFFIC	ELECTRICITY
UNIV.	ARMA	**0.82±0.00**	0.48±0.00	0.77±0.00
	SHALLOWARMA	*0.83±0.00*	0.48±0.00	0.74±0.01
	DEEPARMA	0.84±0.01	**0.45±0.01**	*0.70±0.02*
	LSTM	0.88±0.03	*0.45±0.00*	0.72±0.03
	DEEPLSTM	0.98±0.10	0.45±0.00	**0.69±0.02**
	GRU	0.86±0.02	0.45±0.01	0.70±0.02
	DEEPGRU	0.86±0.02	0.45±0.01	0.70±0.01
	SIMPLE	0.91±0.06	0.46±0.00	0.71±0.01
	DEEPSIMPLE	0.93±0.07	0.46±0.01	0.70±0.01
MULTIV.	ARMA	*0.82±0.00*	0.48±0.00	0.77±0.00
	SHALLOWARMA	**0.82±0.01**	0.53±0.01	0.78±0.01
	DEEPARMA	0.83±0.00	0.53±0.02	0.77±0.03
	LSTM	0.97±0.04	0.49±0.01	0.93±0.20
	DEEPLSTM	1.06±0.08	0.49±0.02	0.74±0.03
	GRU	0.97±0.09	0.48±0.00	0.73±0.02
	DEEPGRU	0.97±0.06	0.48±0.01	*0.72±0.02*
	SIMPLE	0.99±0.03	**0.47±0.01**	0.73±0.02
	DEEPSIMPLE	1.01±0.05	*0.47±0.01*	**0.70±0.01**

Contributing Publications

Boyd, S., Johansson, K., Kahn, R., Schiele, P., Schmelzer, T. (2024). Markowitz Portfolio Construction at Seventy. *Available at* https://arxiv.org/abs/2401.05080. To appear in the *Journal of Portfolio Management*.

Schiele, P., Luxenberg, E., Boyd, S. (2023). Disciplined Saddle Programming. *Available at* https://arxiv.org/abs/2301.13427. Published in *Transactions on Machine Learning Research*, *available at* https://openreview.net/forum?id=KhMLfEIoUm.

Luxenberg, E., Schiele, P., Boyd, S. (2022). Robust Bond Portfolio Construction via Convex-Concave Saddle Point Optimization. *Available at* https://arxiv.org/abs/2212.02570. Published in the *Journal of Optimization Theory and Applications*, *available at* https://link.springer.com/article/10.1007/s10957-024-02436-z.

Luxenberg, E., Schiele, P., Boyd, S. (2022). Portfolio Optimization with Cumulative Prospect Theory Utility via Convex Optimization. *Available at* https://arxiv.org/abs/2209.03461. Published in *Computational Economics*, *available at* https://link.springer.com/article/10.1007/s10614-024-10556-x.

Schiele, P., Berninger, C., Rügamer, D. (2021) ARMA Cell: A Modular and Effective Approach for Neural Autoregressive Modeling. *Available at* https://arxiv.org/abs/2208.14919. Planned submission (after revising according to submission guidelines): *Scientific Reports*.

Eidesstattliche Versicherung (Affidavit)

(Siehe Promotionsordnung vom 12. Juli 2011, § 8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 12.01.2024 Philipp Schiele