FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Bayesian Portfolio Selection Using Markov Chain Monte Carlo Methods

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Lukas Steinbrecher
Matrikelnummer 01129465

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.Univ.Prof. Mag. Dr. Dr. Thomas Dangl

Wien, 30. April 2018

Lukas Steinbrecher	Thomas Dangl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Bayesian Portfolio Selection Using Markov Chain Monte Carlo Methods

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Lukas Steinbrecher
Registration Number 01129465

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Mag. Dr. Dr. Thomas Dangl

Vienna, 30th April, 2018

Lukas Steinbrecher	Thomas Dangl

Erklärung zur Verfassung der Arbeit

Lukas Steinbrecher
Wopfing 358, 2754 Waldegg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. April 2018

Lukas Steinbrecher

Danksagung

Ich danke meiner ganzen Familie, die mich während meines gesamten Studiums unterstützt hat. Insbesondere danke ich meinen Eltern und Großeltern.

Kurzfassung

Diese Diplomarbeit untersucht den Bayesschen Ansatz zur Portfolioselektion. Wir benutzen ein nicht-hierarchisches und ein hierarchisches multivariat-Gaussches Bayessches Modell mit zwei verschiedenen Prior-Verteilungen. Aufgrund der mathematischen Unlösbarkeit solcher komplizierten Bayesschen Modelle benutzen wir eine probabilistische Programmiersprache namens Stan, welche den numerischen Markov-Chain Monte Carlo (MCMC) Ansatz zur Inferenz dieser Modelle benutzt. Wir betrachten zwei verschiedene Portfoliooptimierungsmethoden: die klassische Mean/Variance Optimierung und einen direkten Erwartungsnutzen-Maximierungsansatz basierend auf der CRRA-Nutzenfunktion. Wir führen eine Backtest-Analyse, basierend auf einem Datensatz der 10 Industriesektoren des S&P 500 zwischen 1989 und 2016, durch. Wir vergleichen dabei die Bayessschen Modelle und Optimierungsmethoden mit dem klassischen Mean/Variance Ansatz mit Maximum-Likelihood-Schätzer. Obwohl die Baysschen Modelle keine besseren Punktvorhersagefähigkeiten besitzen, war der Einsatz dieser fortgeschrittenen Methoden, aufgrund der Nutzengewinne von 1-2% Zusatzrendite des Sicherheitsäquivalents pro Jahr, gerechtfertigt. Alle zusätzlichen Gewinne konnten durch das Fama-French-Carhart Vier-Faktorenmodell erklärt werden.

Abstract

This thesis investigates the Bayesian approach to portfolio selection. We use a non-hierarchical and a hierarchical Multivariate Gaussian Bayesian model with two different prior distributions. Due to the mathematical intractability of such advanced Bayesian models, we use a probabilistic programming language called Stan, which uses the numerical Markov Chain Monte Carlo (MCMC) approach to do inference on these models. We consider two different portfolio optimization methods: The classical Mean/Variance optimization and an direct expected utility maximization approach based on the CRRA utility function. We conduct a historical backtesting analysis based on a dataset consisting of the 10 industry sectors of the S&P 500 index between 1989 and 2016. We compare the Bayesian models and optimization methods to the classical Markovitz Mean/Variance approach with maximum likelihood estimators. Although the Bayesian models do not provide better point prediction forecast ability, the use of these advanced methods was justified, by providing an utility gain which ranges up to 1-2% additional certainty equivalent return per year. All additional returns can be explained by the Fama-French-Carhart four factor model.

Contents

Notation

x	A scalar, e.g. one possible assignment for $\mathbf{x}$
X	A vector or matrix, e.g. one possible assignment for $\mathbf{X}$
$\mathbf{x}, \mathbf{X}$	A random variable, vector or matrix
$x^{(i)}$	A particular sample from $\mathbf{x}$
X_{ij}	When X is a 2-dimensional matrix, X_{ij} is the element at row i and column j
X_i	When X is a 2-dimensional matrix, X_i is the row vector at row i, i.e. all values at row i
$p(x)$, $p_{\mathbf{x}}(x)$	The probability density function of $\mathbf{x}$
$p(x \mid y)$, $p_{\mathbf{x} \mid \mathbf{y}}(x \mid y)$	The conditional probability density function of $\mathbf{x}$ given $\mathbf{y} = y$
$P(E)$	Probability of event E happening
X^T	The transpose of matrix X
$\mathbb{1}$	Ones vector, vector with all components being 1 (usually of size S)

Abbreviations

ACF	Autocorrelation function
CAPM	Capital asset pricing model
CDF	Cumulative distribution function
CER	Certainty equivalent return
CRRA	Constant relative risk aversion
DEU	Direct expected utility
DSL	Domain specific language
HMC	Hamilton Monte Carlo
iid	independent and identically distributed
MCMC	Markov chain Monte Carlo
ML	Maximum likelihood
MLE	Maximum likelihood estimators
MSE	Mean squared error
NUTS	No-U-turn sampler
PPL	Probabilistic programming language
RMSE	Root mean squared error
RNG	Random number generator
SQP	Sequential quadratic programming

1 Introduction

Portfolio selection deals with the problem of selecting the optimal weights for a portfolio of risky financial assets based on investors preferences and constraints. Markowitz (1952) introduced the classical Mean-Variance framework, which is widely recognized as the cornerstone of modern portfolio theory. The Mean-Variance model uses portfolio variance as a measure of risk, and assumes that an investor wants to maximize its utility and therefore maximizing the portfolio return based on a given amount of risk, which he is willing to take. All portfolios which maximize the expected return given an amount of risk are known as efficient portfolios. The Markowitz approach assumes that the expected returns and variances/covariances are given, although the real value of these measures are typically unknown and have to be estimated. One way to estimate this values is to use the maximum likelihood estimators (MLE) on historical returns.

The **Bayesian** approach is an alternative approach to portfolio selection and tries to deal with practical issues, which have been identified in the classical Mean/Variance approach with ML estimators.
The classical approach which uses ML to estimate the expected returns and the covariance-matrix doesn't take into account parameter uncertainty (Jorion (1986), Markowitz (1952), Kalymon (1971)) which can lead to suboptimal and volatile weight choices. This problem was already recognized by Markowitz in his original paper, and different solutions have been proposed to incorporate this uncertainty (e.g., see Dangl and Weissensteiner, 2017; Kan and Zhou, 2007, for recent non-Bayesian approaches). The problem of parameter uncertainty is also naturally incorporated by the Bayesian approach by treating the model parameters $\boldsymbol{\Theta}$ as probability distributions instead of constants.

The Mean/Variance optimization approach uses the first two moments for calculating the optimal weights for the investor. When using different assumptions about investors utility (as compared to the Mean/Variance approach) additional information might be used (e.g. higher moments in the return distribution).

Also well known market anomalies like the momentum effect (e.g. Jegadeesh and Titman (1993)) or external variables which can predict future asset returns (e.g. Dangl and Halling (2012)) may be incorporated in the model to improve prediction accuracy. Although using the described techniques are not Bayesian exclusive, the use of Bayesian numerical methods make it relatively easy and allow the use of highly complex models and therefore the (mathematical) complexity of the model is only a minor issue.

The Bayesian portfolio approach follows this procedure (Bade et al. (2009)): First

the probability distributions of the returns (the likelihood) and the parameter prior distributions are defined ("the model"). Then the posterior probability distributions are obtained, either by directly solving the Bayesian formula or by using MCMC. The posterior probability distributions incorporate the observed data, e.g. historical returns.

In the last step the weights of the portfolio are calculated either by obtaining the predictive moments (means and variance-covariances) from the predictive (posterior) returns and by using the classical mean-variance-formula or by directly optimizing the investors expected utility based on a different utility function (e.g. CRRA utility) using the predictive returns.

Markov chain Monte Carlo (MCMC) refers to a set of algorithms which allow to draw samples from the posterior probability distribution of a Bayesian model.
If one wants to do Bayesian inference on a model (i.e. calculating the posterior probability distribution) this can be done with the well known Bayesian formula $p(\Theta \mid X) = \frac{p(\Theta)p(X \mid \Theta)}{p(X)} = \frac{p(\Theta)p(X \mid \Theta)}{\int p(X \mid \Theta)p(\Theta)d\Theta}$, where $p(\Theta)$ is the prior probability density, $p(\Theta \mid X)$ the posterior probability density, $p(X \mid \Theta)$ the likelihood of the data, and in the denominator $p(X)$ the evidence (also called marginal likelihood), which is a constant. Often it is very hard, or even impossible to calculate the posterior using the Bayesian formula because it involves solving the integral of the evidence. Historically this problem has been solved by using simple models or by using special "conjugate" priors which behave "nicely" when applied on the Bayesian formula.
Markov chain Monte Carlo algorithms in contrast avoid this problem by constructing a Markov chain whose equilibrium distribution is the probability distribution of the posterior. This provides a numerical method to directly sample from the posterior without solving the Bayesian formula analytically. Although already theoretically described in the 50's (Metropolis et al. (1953)), MCMC only started to gain popularity in recent times because such algorithms are computationally intensive (Robert and Casella (2011)). In recent times not only advanced MCMC algorithms (e.g. Hamilton Monte Carlo based NUTS, Hoffman and Gelman (2011)), which allow the usage of highly complex models with a high number of parameters, have been discovered, also the increasing power of computers allows the practical use of MCMC methods.

Probabilistic Programming Languages (PPL) refer to a group of domain specific languages which allow the declarative description of statistical models and provide Markov chain Monte Carlo algorithms to do inference on these models. Popular PPL include Stan (Carpenter et al. (2016)), WinBUGS (Lunn et al. (2000)) and JAGS (Plummer et al. (2003)).

1.1 Research question and methodology

In this thesis we are investigating the Bayesian approach to portfolio selection. We use Bayesian models, in particular a non-hierarchical and a hierarchical multivariate Gaussian model with two different priors. We are considering two different portfolio optimization methods: The classical Mean/Variance optimization and an expected utility maximization approach using the isoelastic utility function. We want to investigate whether such advanced Bayesian models and optimization methods are superior to the classical Mean/Variance approach with ML estimators, and therefore the usage of such advanced models and optimization methods is justified. To provide evidence to the asked question, we conduct a historical backtesting analysis based on a dataset consisting of monthly percentage returns of the 10 industry sectors of the S&P 500 index between October 1989 and August 2016. We then analyze and compare the constructed portfolios to the portfolio constructed by the classical approach and answer the following questions:

1. Are the estimates of the returns of the Bayesian models superior compared to the maximum likelihood estimator, based on the Mean squared prediction error measure?
   The MSE (Mean squared error) is a statistical measure of the average squared error of the predicted returns and the realized returns. The MSE is compared to the MSE of the benchmark model using the $R^2 = 1 - \frac{MSE(\text{model})}{MSE(\text{benchmark})}$ measure. The benchmark model is the maximum likelihood estimator.

2. Do the Bayesian portfolios provide a higher certainty equivalent return compared to the portfolios constructed by the classical framework?
   The certainty equivalent return is the (expected) return an investor must receive on a portfolio, given his utility function (which usually incorporates his risk preferences) in order to be indifferent between the portfolio and a risk free return.

   $$u(r_{\text{rf}}) = \mathbb{E}\, u(\mathbf{r_p})$$

   We use this formulation for an ex-post analysis of the realized certainty equivalent return by calculating the average ex-post certainty equivalent return of the realized portfolio returns. We compare the average ex-post certainty equivalent return of the portfolio of the classical model with the portfolios of the Bayesian models.

3. Can all returns of the Bayesian-constructed portfolios be explained by the economically meaningful factors of the Fama-French-Carhart model?

The factor model of Fama-French-Carhart, Carhart (1997), tries to explain excess portfolio returns using regression analysis and is given by

$$R_{Pt} - R_{RFt} = \alpha + \beta_{RMRF} RMRF_t + \beta_{HML} HML_t + \beta_{SMB} SMB_t + \beta_{MOM} MOM_t + \epsilon_t$$

where R_{Pt} is the portfolio return and R_{RFt} the risk free rate at time t. $RMRF_t$ represents the equity premium factor, and SMB_t, HML_t, MOM_t the size, book-to-market and one year momentum premium factor at time t respectively.

1.2 State of the Art

The Bayesian based approaches to portfolio selection can be divided into two groups. The first group treats the asset returns as independent and identically distributed (iid), and concentrates on modeling the return distribution only conditional on the historical returns (e.g. by assuming a Multivariate Gaussian distribution and estimating the μ and Σ parameters). The other group treats the asset returns as time dependent and therefore predictable. External economic variables can be incorporated into the model, e.g., the current stage in the business cycle.

To do inference on those models there are two methods: the analytical and the numerical approach.

In the analytical approach, the posterior probability distributions are derived analytically. As said, this can be hard or even intractable. Only tractable models can be used, this can be done by using conjugate priors, however this may also add some additional unwanted assumptions on the model.

The numerical approach uses MCMC algorithms to sample from the posteriors without deriving an analytical solution. In this approach, the mathematical complexity of the models is not a constraint. Most papers which use this approach use a custom implementation of the model, and do not use a declarative approach using a probabilistic programming language, which can make it hard to reproduce and to compare the results.

Avramov and Zhou (2010) provides an extensive survey of the literature of Bayesian based portfolio optimization. Here we provide an excerpt of recent papers of Bayesian based portfolio optimization:

Harvey et al. (2010) use a skew normal distribution (Sahu et al., 2003) and also incorporate higher moments (skewness and coskewness) in the portfolio selection process. A MCMC algorithm is used for solving the model and they use direct expected utility maximization to calculate the weights.

Jacquier and Polson (2010) describe a general method how simulation techniques and in particular MCMC algorithms can be used in the portfolio selection process. They describe a MCMC algorithm for expected utility maximization.

Qian (2009) use a Gaussian mixture model. Both a classical and a utility maximization approach for calculating the portfolio weights is presented. A custom MCMC routine is developed to obtain the posteriors. They conclude that the mixture model outperforms the classical approach in their simulations.

Dangl and Halling (2012) use a dynamic linear regression model for predicting asset returns. They use regressors like dividend yield, price-earnings ratio and book to market ratio as well as macroeconomic variables like inflation and the treasury bond yield. Bayesian model selection is used to select the most significant predictors and in their empirical evaluation they show a positive utility gain compared to the benchmark model.

Bade et al. (2009) use a orthogonal MGARCH model and hierarchical informative priors. A MCMC algorithm is used to calculate the posterior probabilities and then they calculate the predictive moments and use the classical optimization formula to compute the portfolio weights. They conclude that the portfolios become well diversified.

Qian (2011) use a Markov Switching Gaussian Mixture Model. They use four regimes and four states to model the future returns. Gibbs sampling (a MCMC algorithm) is used to obtain the predictive returns. The classical formula with predictive moments is used for optimization.

Polson and Tew (2000) use a hierarchical model and informative priors. An analytical derivation of the posterior is presented.

Greyserman et al. (2006) present both a hierarchical model and a model based on the estimators of James and Stein (1961). MCMC is used for solving the models and approximate direct utility optimization approach is taken for weight calculation.

Avramov (2002) uses Bayesian model averaging to select an optimal return forecasting model using economic variables. Useful predictors include difference between lagged returns on long-term and short-term government bonds, earnings yield, and the treasury-bill rate. They show that small high book-to market stocks are more predictable than low-book-to-market stocks. An analytical approach is used.

Non-Bayesian approaches It has to be noted, that the Bayesian portfolio formalism is not the only approach to deal with parameter uncertainty (e.g., see Dangl and Weissensteiner, 2017; Kan and Zhou, 2007, for recent non-Bayesian approaches). Further it is not the only approach to use additional information in the return distribution (e.g. higher moments in the return distribution). By introducing additional assumptions (e.g. normal distribution of returns), frequentist methods can be used with the direct

expected utility optimization method (e.g., see Kan and Zhou, 2007). Also modern non-Bayesian approaches, which assume that asset returns are predictable, exist. To name one example, Rapach et al. (2010) present a technique for combining forecasts to provide evidence of out-of-sample predictive ability of 15 economic variables taken as a whole. They state that this forecast combining method stems from the highly complex data-generation process underlying equity returns which are related to a similar process in the real economy, which is difficult to approximate with a single regression model.

1.3 Outline

In the first two chapters we begin by building the theoretical basis of portfolio selection and Bayesian methods, which will then be used in the following practical part of the thesis, in which we conduct a case study based on historical simulation.

The first chapter focusing on portfolio theory begins by introducing the classical Markowitz portfolio optimization framework and further the Bayesian framework to portfolio optimization. Then the utility theory is presented following a detailed view on direct expected utility optimization, which will also be used in the case study. The chapter is completed by taking a look on some theoretical aspects of historical simulation, in particular some pitfalls of backtesting.

The second chapter focuses on Bayesian methods and MCMC. We begin by introducing the concept of Bayesian inference. We then proceed to methods and algorithms to numerically solve the Bayesian inference problem. First Monte Carlo methods in general are introduced, and then we proceed to the ideas of Markov Chain Monte Carlo and take a detailed look at two specific MCMC algorithms, namely Metropolis-Hastings and Hamilton Monte Carlo. In the last section of the chapter we introduce probabilistic programming languages, which provide an declarative way to define Bayesian models and use MCMC algorithms to do inference on them. In particular the probabilistic programming language Stan is introduced, which will be used in the case study.

In the third chapter, which is the practical part of this thesis, we conduct a historical simulation based on a dataset of the industries of the S&P 500 and a time period between 1989 and 2016. We conduct a backtest analysis comparing the portfolios constructed by the classical model and Bayesian models. We use a non-hierarchical and a hierarchical multivariate normal Bayesian model with two different priors. We optimize the portfolios both with the Mean/Variance optimization approach and a direct expected utility maximization approach. In total we conduct nine different experiments. After conducting the experiments we analyze and compare the results to the classical model using different statistical and economical measures.

2 Portfolio selection

In this chapter we discuss the original theory of Markowitz based portfolio optimization, then we get into the details of Bayesian based methods. Next we present the ideas of utility theory and how the utility of individual investors are modeled in economics, followed by a detailed view on direct expected utility optimization. The chapter is completed by taking a look on some theoretical aspects of historical simulation, in particular some pitfalls of backtesting.

2.1 Markowitz portfolio optimization

Markowitz introduced the Mean-Variance framework (also called Modern Portfolio Theory) in his article Markowitz (1952). He introduced the process of selecting an optimal portfolio for an investor by dividing it into two stages. The first stage is concerned about the future performance of the available financial assets and results in a set of quantified characteristics about this assets. The second stage is concerned about building an optimized portfolio, this is accomplished by taking the quantified characteristics of the financial assets in the first stage and optimizing the portfolio according to some goal function such as the investors expected utility. He concentrates only on two of these quantified characteristics of financial assets, namely the expected value and variance of the portfolio (hence the name Mean/Variance framework) and argues that the investor desires expected return and does not desire variance (i.e. uncertainty). One could argue the best way to build a portfolio is by maximizing only the expected (discounted) return and ignoring the variance, but Markowitz rejects this hypothesis and argues that this would never imply that there is a diversified portfolio which is preferable over a expected value optimized portfolio, but diversification is both observed and sensible, and therefore does not imply the superiority of diversification and thus must be rejected.

2.1.0.1 Problem formulation

This section is mainly based on Markowitz (1952) and the detailed explanations of Chapados (2011).

Asset return We have a risky financial asset, e.g. a stock share, index[1] or a bond, which has a market price p_t at time t, for which the financial asset can be bought or sold. We define the (simple/discrete) return for period $t-1$ to t of the asset as the relative

[1] Indices are a collection financial assets, e.g the top 10 stocks in Europe based on market capitalization of their respective companies, which can often be directly traded using ETFs (exchange traded funds)

change of the price subtracted by 1:

$$r_t = \frac{p_t}{p_{t-1}} - 1$$

The portfolio We have an investable universe of $j = 1..S$ risky financial assets, and an investor chooses a portfolio at time t, i.e. allocates his wealth among the S assets, based on the expectations on the unknown and therefore random returns $\mathbf{R_{t+1}} \in \mathbb{R}^S$ for period t to $t+1$. Usually the investors portfolio is expressed in terms of the portfolio weight vector $w \in \mathbb{R}^S$, where w_j is the fraction of the total capital invested in asset j in period t to $t+1$.

Means and the covariance matrix of returns As said, in the Mean/Variance framework for choosing a portfolio the investor only takes into account the means $\mu \in \mathbb{R}^S$ of the returns and covariances $\Sigma \in \mathbb{R}^S \times \mathbb{R}^S$ between the returns:

$$\mu = \mathbb{E}(\mathbf{R_{t+1}}) \tag{1}$$

$$\Sigma = \mathrm{Cov}(\mathbf{R_{t+1}}) \tag{2}$$

The real μ and Σ are typically unknown. One common method to estimate μ and Σ is to use historical return data up to time t and to calculate the maximum likelihood estimations of μ and Σ based on the historical returns from some period $t - t_{start}$ to t. This period $t - t_{start}$ to t is also called the look-back period.

Means and variance of the portfolio After obtaining μ and Σ, or a proxy thereof, we can calculate the expected return μ_P and variance of the portfolio σ_P^2 in terms of the means and the covariance matrix of returns.

$$\mu_P = w^T \mu \tag{3}$$

$$\sigma_P^2 = w^T \Sigma w \tag{4}$$

Markowitz efficient portfolio A portfolio w is efficient when it has the lowest variance among all obtainable portfolios for a given μ_{target}, where μ_{target} is the pre-chosen target expected portfolio return (Minimum variance formulation). One additional constraint is that the portfolio must be fully invested, i.e. the portfolio weights must sum

to one[2]:

$$w_{eff} = \underset{w}{\operatorname{argmin}} \frac{1}{2}\sigma_P^2 \tag{5}$$

$$= \underset{w}{\operatorname{argmin}} \frac{1}{2} w^T \Sigma w \tag{6}$$

$$\text{s.t. } w^T \mu = \mu_{\text{target}} \tag{7}$$

$$\mathbb{1}^T w = 1 \tag{8}$$

Deriving the analytical solutions for the efficient portfolio problem is straightforward using the method of Lagrange multipliers, which can be found, e.g., in Chapados (2011) or in the R-Code for Example 2.1 in Appendix 21.

Analogue to the efficient portfolio, one can obtain the global minimum variance portfolio w_{gmv}, which is the portfolio with the least variance among all obtainable portfolio. The optimization formulation is equivalent to the efficient portfolio formulation, but without the target return constraint. The solution for w_{gmv} is:

$$w_{gmv} = \frac{\Sigma^{-1}\mathbb{1}}{\mathbb{1}^T \Sigma^{-1} \mathbb{1}} \tag{9}$$

Utility maximization formulation When using the efficient portfolio formulation, one has to choose a target return, which is not always obvious. There is an alternative formulation, in which the expected utility of the investor is directly optimized. It can be shown (Chapados, 2011, e.g., see) that the unconstrained minimum variance formulation (by Markowitz) and the utility maximization formulation are equivalent. For this formulation one has to choose a risk aversion parameter λ, however this parameter has a much more practical interpretation, defining the marginal rate of substitution between expected return and return variance. As pointed out in Section 2.3 a rational investor wants to maximize its expected utility and therefore the problem can be defined as:

$$w^* = \underset{w}{\operatorname{argmax}}\, \mathbb{E}(u(\mathbf{R_P})) \tag{10}$$

$$= \underset{w}{\operatorname{argmax}}\, \mu_P - \frac{\lambda}{2}\sigma_P^2 \tag{11}$$

$$= \underset{w}{\operatorname{argmax}}\, w^T \mu - \frac{\lambda}{2} w^T \Sigma w \tag{12}$$

$$\text{s.t. } \mathbb{1}^T w = 1 \tag{13}$$

[2] $\mathbb{1}$ is the ones vector of size S, i.e. the vector with all components being 1

Example 2.1 (Mean/Variance optimization)**.** We have a hypothetical investable universe of three risky financial assets consisting of two stocks and one bond. We assume the two stocks have yearly expected returns $\mu_1 = 11\%, \mu_2 = 10\%$ and yearly expected volatility of $\sigma_1 = 23\%, \sigma_2 = 22\%$ and a correlation of $\rho_{12} = 0.4$. The bond has a slightly lower yearly return and risk of $\mu_3 = 7\%$ and $\sigma_3 = 12\%$. We assume the bond has a negative correlation to the stocks: $\rho_{13} = -0.2, \rho_{23} = -0.1$.

We can construct the mean vector μ and covariance matrix Σ, using the standard deviations σ and the correlations ρ:[3]

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} = \begin{pmatrix} 0.11 \\ 0.1 \\ 0.07 \end{pmatrix}, \sigma = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{pmatrix} = \begin{pmatrix} 0.23 \\ 0.22 \\ 0.12 \end{pmatrix} \tag{14}$$

$$\rho = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0.4 & -0.2 \\ 0.4 & 1 & -0.1 \\ -0.2 & -0.1 & 1 \end{pmatrix} \tag{15}$$

$$\Sigma = \text{diag}(\sigma)\, \rho\, \text{diag}(\sigma) = \begin{pmatrix} 0.053 & 0.020 & -0.006 \\ 0.020 & 0.048 & -0.003 \\ -0.006 & -0.003 & 0.014 \end{pmatrix} \tag{16}$$

We now want to construct an efficient portfolio w_{eff} according to the Mean/Variance framework with a yearly target return of $\mu_{target} = 0.1$. Additionally we calculate the global minimum variance portfolio w_{gmv} using the solutions provided by the Mean/Variance framework. Additionally we calculate the expected means and volatilities of the portfolios:

$$w_{eff} = \begin{pmatrix} 0.53 \\ 0.29 \\ 0.18 \end{pmatrix}, \ \mu_{peff} = \mu_{target} = 0.1, \ \sigma_{peff} = 0.156 \tag{17}$$

$$w_{gmv} = \begin{pmatrix} 0.18 \\ 0.14 \\ 0.68 \end{pmatrix}, \ \mu_{pgmv} = 0.081, \ \sigma_{pgmv} = 0.092 \tag{18}$$

We plot the assets, the efficient portfolio, the global minimum variance portfolio and the efficient frontier (representing all obtainable efficient portfolios) in the standard deviation vs. expected return space:

[3] Given a vector $x \in \mathbb{R}^S$ the $\text{diag}(x)$ function returns a matrix $\in \mathbb{R}^{S \times S}$ with the components of x on the diagonal

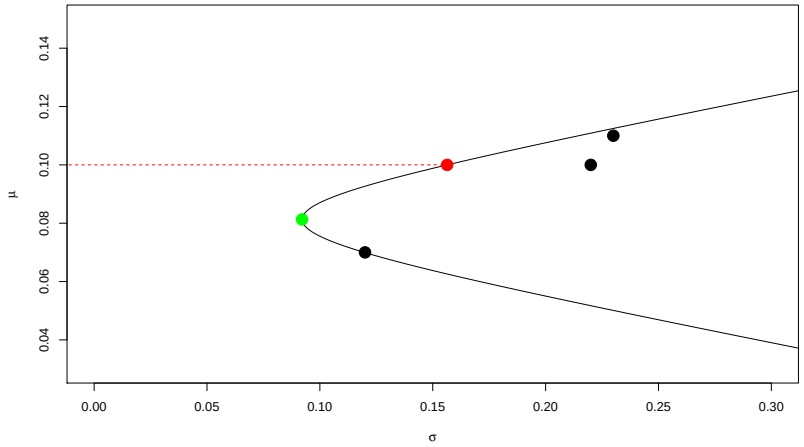

Figure 1: Example 2.1 (Mean/Variance optimization): Black dots: The three assets, Red dot: The efficient portfolio for a target return of 10%, Green dot: The global minimum variance portfolio, Black curve: The efficient frontier

Investing in the efficient portfolio with target return 10%, would mean putting 53% and 29% of our wealth in stock 1 and 2, and the remainder of 18% into the bond. Our constructed portfolio has as nearly as much expected return as the stocks ($\mu_{peff} = 10\%$), while having much lower standard deviation of $\sigma_{peff} = 15.6\%$. We also clearly see, the global minimum variance portfolio has the lowest standard deviation among all obtainable portfolios ($\sigma_{pgmv} = 9.2\%$).

The R-Code for this example can be found in Appendix 21.

Different formulations Different formulations for the efficient portfolio formulation (as defined in the preceding chapter) have been discussed, e.g. by introducing a risk free asset, incorporating additional constraints into the optimization problem, such as transaction cost constraints, maximum holding constraints, or by using a different optimization function, by not minimizing the variance but instead, for example, the value at risk.

One important constraint, which was already discussed by Markowitz in his original paper, and which we will be using in our experiments, is the short sale constraint, which disallows short-selling, i.e. all individual asset weights must be greater than or equal to 0. This is a constraint, with which many investors are confronted in practice, since short-selling may involve regulatory (short selling may be disallowed for certain accounts) or practical hurdles (short selling involves borrowing an asset from a third party which is willing to lend, which may not be available, also this could incorporate additional

short-sale fees). To solve the portfolio optimization problem with the short sale constraint analytically, an iterative Kuhn-Tucker approach can be used (e.g., see Jagannathan and Ma, 2002). Also numerical optimizers, i.e., linear-quadratic solvers, efficiently implement this iterative approach.

2.2 Bayesian based portfolio optimization

In the last section, we introduced the Mean/Variance framework, which provides an formulation for selecting efficient portfolios for risky assets. Avramov and Zhou (2010) in their survey of Bayesian portfolio selection methods, name three arguments of why the Bayesian portfolio selection approach might be useful:

Prior information Pre-known knowledge can be incorporated into the models by using informative prior distributions.

Parameter uncertainty The Bayesian approach is one way (including other, non Bayesian techniques, see above) to deal with parameter uncertainty. The problem of parameter uncertainty is naturally incorporated by the Bayesian approach by treating the model parameters $\boldsymbol{\Theta}$ as probability distributions instead of constants.

Convenience of numerical algorithms Numerical algorithms allow a more declarative approach, where one can focus on the "modeling" part without worrying on the mathematically convenience and/or tractability of the models.

2.2.1 The Bayesian framework to portfolio optimization

One can roughly divide Bayesian based approaches to portfolio selection into two groups (Avramov and Zhou (2010)). The first group treats the asset returns as independent and identically distributed (iid), and concentrates on modeling the return distribution only conditional on the historical returns (e.g. by assuming a Multivariate Gaussian distribution and estimating the μ and Σ parameters). The other group treats the asset returns as time dependent and therefore predictable. One example of such a predictable model is estimating a linear regression model where the return is the dependent variable and the independent variables are based on some economic measure, such as GDP growth and treasury bill rate.

An overview of the state of the art of Bayesian based portfolio selection approach is provided in the introduction in Section 1.2.

In this thesis, we concentrate on models in group one, i.e. on modeling the return distribution only conditional on the historical returns. In the Bayesian framework, the parameter vector $\boldsymbol{\Theta}$, which contains all model parameters (e.g. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in a Multivariate

Gaussian distribution setting), is treated as a random variable. For a detailed discussion of Bayesian inference, see Section 3.1.

We define the model as the likelihood $p(X \mid \Theta)$, defining the probability density of the dataset $\mathbf{X}$, given one particular assignment of the parameter vector $\mathbf{\Theta}$, and $p(\Theta)$ the a-priori distribution of $\mathbf{\Theta}$, which includes all knowledge about the parameters a-priori the inference, i.e. before we observe the data. Having a dataset X available at time t (in our case the historical returns of the financial assets up to time t), we can use the Bayesian inference formulation to get the a-posteriori distribution density $p(\Theta \mid X)$:

$$p(\Theta \mid X) = \frac{p(X \mid \Theta) \cdot p(\Theta)}{\int_\Theta p(X \mid \Theta) \cdot p(\Theta)\ d\Theta} \tag{19}$$

Having the joint posterior densities, we can then integrate out the parameters $\mathbf{\Theta}$ to obtain the predictive return density:

$$p(R_{t+1} \mid X) = \int_\Theta p(R_{t+1} \mid \Theta, X)\ d\Theta \tag{20}$$

Then the Bayesian framework to portfolio optimization can be formulated as a utility maximization of the predictive return distribution (Avramov and Zhou (2010)):

$$w_{Bayes} = \underset{w}{\operatorname{argmax}} \int_{R_{t+1}} u(w)\ p(R_{t+1} \mid X)\ dR_{t+1} \tag{21}$$

In our case study we use different Bayesian models to estimate $\mathbf{\Theta} \mid X$, which are further motivated in Section 4.2.2. We approximate the predictive return density with Markov Chain Monte Carlo by numerically integrating out the parameters. We then calculate the optimal portfolio using direct expected utility optimization (Section 2.3.1).

2.2.2 Using the Bayesian framework for Mean/Variance optimization

The Mean/Variance framework uses the first two moments of the return distribution (the expected value μ and the variance-covariance matrix Σ) to calculate an efficient portfolio. In a Bayesian setting, we can obtain the expected value $\mu = \mathbb{E}(\mathbf{R_{t+1}} \mid \mathbf{X})$ and variance $\Sigma = Var(\mathbf{R_{t+1}} \mid \mathbf{X})$ of the predictive return distribution $\mathbf{R_{t+1}} \mid \mathbf{X}$. As Polson and Tew

(2000) points out, these statistics can be calculated as[4]:

$$\mu = \mathbb{E}(\mathbf{R_{t+1}} \mid \mathbf{X}) = \mathbb{E}(\boldsymbol{\mu} \mid \mathbf{X}) \tag{22}$$

$$\Sigma = Var(\mathbf{R_{t+1}} \mid \mathbf{X}) \tag{23}$$

$$= \mathbb{E}(Var(\mathbf{R_{t+1}} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{X})) + Var(\mathbb{E}(\mathbf{R_{t+1}} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{X})) \tag{24}$$

$$= \mathbb{E}(\boldsymbol{\Sigma} \mid \mathbf{X}) + Var(\boldsymbol{\mu} \mid \mathbf{X}) \tag{25}$$

As we can see, we can calculate the expected value for the predictive return distribution using the expected value of the marginal a-posteriori mean $\boldsymbol{\mu} \mid \mathbf{X}$. For the variance of the predictive return distribution we can use the marginal a-posteriori variance-covariance matrix $\boldsymbol{\Sigma} \mid \mathbf{X}$, as one would expect, but we have to add an additional term to correct for parameter uncertainty, using the variance of the marginal a-posteriori mean $\boldsymbol{\mu} \mid \mathbf{X}$. This correction is consistent with a correction used for parameter uncertainty in a non-Bayesian setting (e.g., see Dangl and Weissensteiner, 2017).

2.3 Utility

In this section, we provide a brief introduction to utility theory and show how and why we chose our utility function used for the direct expected utility optimization technique. This section is based on the texts by Norstad (1999) and Johnson (2007).

A utility function $u(m)$ is a function which maps from a measure of wealth to the perceived value of that wealth for an investor (Johnson (2007)). A utility is twice differentiable with the property of non-satiation, which means an investor always gets some additional utility by getting more wealth ($u'(m) > 0$) and the property of risk aversion ($u''(m) < 0$).

The principle of expected utility maximization states, that a rational investor, when faced with more than one possible outcome, chooses that action, which maximizes the expected utility (Norstad (1999)). The classical Markowitz approach is consistent with this principle, since the Markowitz optimization approach can also be viewed as a expected utility maximization problem (see Section 2.1).

In the portfolio optimization setting (see Section 2.1) we are faced with this type of uncertainty, we have a set of possible outcomes (in our case the predictive return vector $\mathbf{R_{t+1}}$ for a set of assets for the next period) and a set of possible actions (in our case choosing in which assets to invest, i.e. choosing the portfolio weight vector w). To act according to the expected utility maximization we choose our portfolio such that our

[4] $Var(R_{t+1} \mid X)$ is decomposed using the law of total variance (see Weiss et al. (2005))

portfolio weights $w^* = \text{argmax}_w \, \mathbb{E}(u(\mathbf{R_{t+1}}w))$.

There is one important measure of utility functions which measures the relative risk aversion of a utility function based on a measure of wealth m (Johnson (2007)):

$$Rr(m) = -m\frac{u''(m)}{u'(m)}$$

Relative risk aversion measures risk aversion to a loss relative to the investors wealth. Increasing relative risk aversion (IRRA)/Decreasing relative risk aversion (DRRA) state, that when his/her wealth increases he/she will hold more/less fractions of total wealth in risky assets. In the case of constant relative risk aversion (CRRA) the investor will hold the same fractions in risky assets as his/her wealth increases.

We assume the initial wealth in our experiments is an arbitrarily chosen number, e.g. 1 Mio. \$, and we don't want an influence of the absolute value of wealth to our portfolio choices in our experiments, so we choose the isoelastic utility function which has the CRRA property. This also simplifies our calculations, since we can directly work with relative changes of the wealth instead of absolute value of wealth.

The isoelastic utility function (also called power-utility function or CRRA utility function) is given by:

$$u(m) = \begin{cases} \frac{m^{1-\eta}-1}{1-\eta} & \eta \neq 1 \\ \ln(m) & \eta = 1 \end{cases}$$

Where η is the risk aversion and defines the risk preferences of the investor.

There is an ongoing debate, on which value for η is realistic for an average (human) investor. For our experiments we chose a value of $\eta = 3$, based on our own intuition. Using $\eta = 3$, the utility of gaining 20% is $u(0.2) = 0.15$, whereas the utility of loosing 20% of our wealth is $u(-0.2) = -0.28$, so about as twice as bad in terms of utility, which seems reasonable. For higher values of η the utility falloff in the negative regions is much faster, e.g. for $\eta = 8$ the utility of loosing 20% vs. gaining 20% is 5 times as bad, which is in our opinion unrealistic.

Since we don't have the case with $\eta = 1$, and since we work with relative changes of wealth, rather than absolute changes of wealth, we redefine our utility function in terms of returns, substituting the wealth m with the return r, $m = 1 + r$ (allowed because of the CRRA property):

$$u(r) = \frac{(1+r)^{1-\eta} - 1}{1-\eta}$$

In our experiments we work with monthly returns, after inspecting the dataset used in

the case study (for details, see Section 4.1), we found the returns roughly to be in the range of -20% to 20%. In Figure 2 we plotted the utility functions in the relevant range.

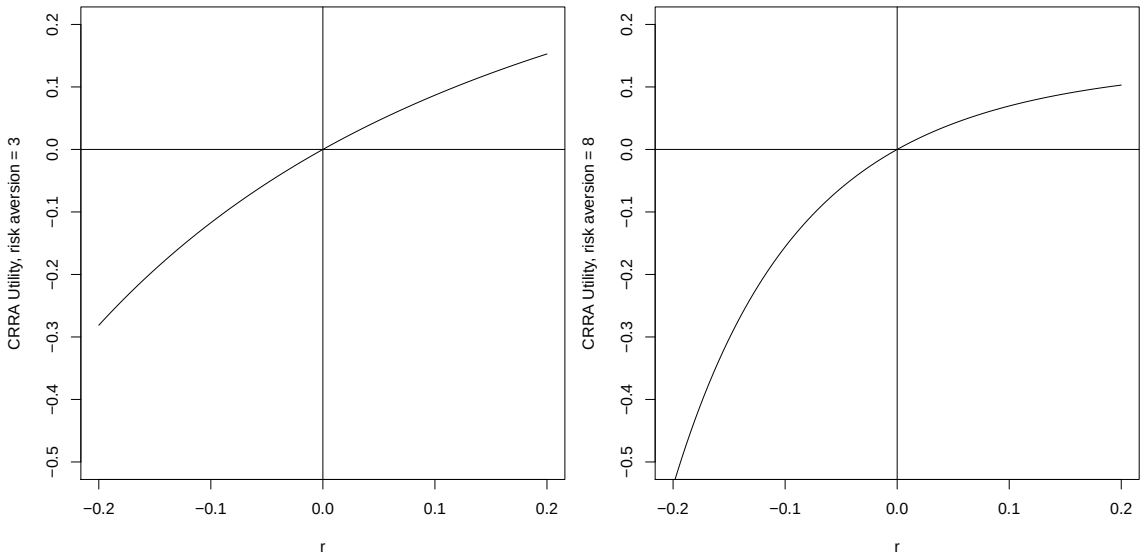

Figure 2: CRRA utility function $u(r)$ for a relative change of wealth r, with risk aversion parameter $\eta = 3$ (left) and $\eta = 8$ (right). For our experiments we chose $\eta = 3$.

2.3.1 Direct expected utility maximization

As said, to act according to the expected utility maximization we choose our portfolio such that our portfolio weights maximize the expected utility dependent the predictive return vector $\mathbf{R_{t+1}}$[5]. In this section we explain how we define the optimization problem and how we used the numerical optimization procedure called Sequential Quadratic Programming (SQP) to find a solution to the the constrained optimization problem.

As discussed in detail in Section 4.3, we use the numerical Markov Chain Monte Carlo procedure to obtain N representative samples from $\mathbf{R_{t+1}}$ and can only approximate the expected utility by calculating the arithmetic average for the utility of each possible outcome. We obtain a sample-matrix $R^{(ij)} \in \mathbb{R}^{N \times S}$, which is a representation of the predictive return vector $\mathbf{R_{t+1}}$, where each $j = 1..S$ column is one asset, and each $i = 1..N$ row is one representative sample (outcome) of $\mathbf{R_{t+1}}$.

As discussed in the preceding section, we chose the isoelastic utility function $u(r) = \frac{(1+r)^{1-\eta}-1}{1-\eta}$, where η denotes the risk aversion and r the portfolio return. For a theoretical

[5]We use the notation $\mathbf{x}$ (bold) to denote a stochastic variable, vector or matrix, as opposed to x which is an ordinary scalar, vector or matrix

discussion of utility theory and the isoelastic utility function see Section 2.3.

The goal of the procedure is, to calculate the portfolio weight vector $w \in \mathbb{R}^S$ for our portfolio which maximizes the expected utility. $w_j = c$ means we invest a fraction of c of our portfolio in asset j. Furthermore we introduce two additional optimization constraints: First, that we are fully invested (the portfolio weights sum to 1), and second, that no short sales are allowed (each weight is ≥ 0).

2.3.1.1 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is one of the most successful (Boggs and Tolle (1995)) method for solving nonlinear constrained optimization methods of the form:

$$\begin{aligned} &\min_x \ f(x) \\ \text{s.t.: } &g(x) = 0, \ h(x) \leq 0 \\ &f : \mathbb{R}^n \to \mathbb{R}, \ h : \mathbb{R}^n \to \mathbb{R}^m, \ g : \mathbb{R}^n \to \mathbb{R}^p \end{aligned} \tag{26}$$

The idea in Sequential Quadratic Programming is to create an approximate solutions by constructing a quadratic programming subproblem, and then iteratively improve this solution by using the last solution as the starting point for the next subproblem. As SQP relies on quadratic subproblems, it benefits from the fact that there are very efficient algorithms for solving quadratic programming problems available.

Fortunately there is an already implemented SQP procedure for R in the library `nloptr` available. The SQP optimization is called with the function `slsqp()`, which takes the target function to be minimized f, the gradient of f, the equality constraint functions g_i, the inequality constraint functions h_i.

As said, we approximate the expected utility by calculating the arithmetic average of the utility of each of the N samples. Our optimization problem is:

$$\max_w \mathbb{E}\big[\, u(\mathbf{R_{t+1}} w)\big] \approx \max_w \frac{1}{N} \sum_{i=1}^{N} u(\sum_{j=1}^{S} R^{(ij)} w_j)$$

$$\text{s.t. } \sum_{j=1}^{S} w_j = 1 \text{ and } w_j \geq 0 \ \forall \ j \in 1..S$$

We can eliminate the $w_j \geq 0$ constraints by substituting $w_j = v_j^2$. To switch from maximization to minimization we multiply the target function by -1.

$$f(v) = -\frac{1}{N}\sum_{i=1}^{N} u(\sum_{j=1}^{S} R^{(ij)} v_j^2) = -\frac{1}{N}\sum_{i=1}^{N} \frac{(1+\sum_{j=1}^{S} R^{(ij)} v_j^2)^{1-\eta} - 1}{1-\eta}$$

$$g_1(v) = \sum_{j=1}^{S} (v_j^2) - 1 = 0$$

Then we derive the gradients as (for a proof, see Appendix B):

$$\begin{aligned} \frac{\partial f}{\partial v_g} &= -2v_g \frac{1}{N}\sum_{i=1}^{N} \left[R^{(ig)} (1 + \sum_{j=1}^{S} R^{(ij)} v_j^2)^{-\eta} \right] \\ \frac{\partial g_1}{\partial v_g} &= 2v_g \end{aligned} \tag{27}$$

The implementation in R can be found in the file `direct-expected-utility.R` function `f.opt.sqp()`.

2.3.1.2 Convergence

The SQP method, like other gradient based methods, guarantees only to find local extrema rather than the global extrema, i.e. the found extrema depend on the starting points. To investigate the global convergence behavior, we conducted a large number of experiments, where we initialized the starting points (in our case the weight vector) randomly using some artificially generated returns. We could not find a case where different starting points led to different extrema, so it seems that SQP always converges to the global extrema for our type of optimization formulation.

2.4 Pitfalls of backtesting

In the case study in Section 4 we will be using a method called backtesting (also called historical simulation) to investigate the performance of our models in a simulated real-world scenario. For details, how we set up the backtesting procedure, take a look at Section 4.3.

There are some pitfalls which one can fall into in the setup of a backtest which lowers the validity of the conducted experiments. In this section we discuss four commonly known pitfalls (DeFusco et al. (2015)) and how we set up our experiments in order to avoid them.

2.4.0.1 Survivorship bias

Survivorship biased data includes only winners, in the sense that companies, which went to bankruptcy got delisted or merged aren't included in the dataset (DeFusco et al. (2015)). This makes the results biased because a company which is bankrupt is likely to have lower returns on it's stock price than the average. Shumway (1997) shows that the survivorship bias is significant and that most datasets don't include delisting data. Also this bias applies when one selects a investment universe based on current data and then conducts a backtest based on that universe. One example is choosing a investment universe from the current constituents of the S&P 500 and conducting a backtest, this biases the returns upwards, because companies which weren't in the S&P 500 at the start of the experiment must have done well to be included, and also companies which were included in the S&P 500 at the start of the experiment but aren't included in the investment universe because they are not part of the S&P 500 anymore must have done badly. To avoid this kind of problem, one must use the constituent data from the start of the experiment.

We avoid this fallacy by using a dataset which includes constituent data, and only use such returns, which are member of the particular index.

2.4.0.2 Look-ahead bias

A backtest is look-ahead biased when it uses data which was not available at a particular time (DeFusco et al. (2015)). One example of this fallacy is a model, which uses accounting information (e.g. earnings report data) of a particular financial year of the company and makes decisions based on this data at the end of this financial year. This kind of setup is look-ahead biased but this is not that obvious, because for example using accounting data for the financial year t at the end of year t seems fine, however this kind of data is usually released much later and this information wasn't available at the end of year t back then, so this use of data is clearly biased. Although the data is for the financial year t, one must use the data only after its original release date (e.g. year $t+1$) to get unbiased results. Another not so obvious example of look-ahead bias is the following: Imagine a mean reversion model for a currency pair (e.g. EUR/USD) where we buy the currency, when it is below the mean value μ, and sell the currency when it is above the mean value μ. If we have a dataset X_t which contains prices from time $t = 0$ to T and calculate the mean as $\mu = 1/T \sum_{t=0}^{T} X_t$ this experiment is clearly biased, although it doesn't use data from future directly, it uses values, which are derived from future data, i.e. the "real" μ wasn't known in earlier periods. The correct approach, to avoid

look-ahead bias, would be to calculate μ for period δ as $\mu_\delta = 1/\delta \sum_{t=0}^{\delta} X_t$, which only uses data which is known up to this time-point.

We avoid this fallacy at the programming level by using a backtest design in which the models can access the data over a specific interface, and which does not allow to query data from the future.

2.4.0.3 Time-period bias

Time-period bias occurs when a specific model or characteristic wants to be shown and the tested time period was chosen, for which this specific characteristic was present (DeFusco et al. (2015)). Usually this bias is reduced by using longer time frames to conduct the experiments.

Although there is no fool-proof way to completely avoid this fallacy, we try to minimize it by using a long time period to conduct our experiments (25 years of data).

2.4.0.4 Data snooping bias

Data-snooping bias (Head et al. (2015), also called p-hacking) is conducting several statistical tests on a single dataset and then reporting the false positives as statistically significant. For example if we statistically test the superiority of 100 models against some reference models with a significance level of 5% we are expected to get 5 false positives by definition. Head et al. (2015) talks about publication pressure and incentives to publish statistically significant results. There is evidence, that journals are more likely to publish results that look significant. However, the paper also concludes that, while false positives can be sticky because positive studies receive more attention and p-hacking is very common in literature, its overall effect on the scientific progress is relatively weak.

We try to avoid this fallacy by reporting the detailed results of all conducted experiments.

2.5 Summary

In the first two sections of this chapter we first introduced the Mean/Variance framework, originally introduced by Markowitz, and discussed how an investor can build an efficient portfolio. We then introduced the Bayesian based portfolio formulation, which is an alternative approach to the portfolio selection problem. We also showed, how the Bayesian framework can be used in the Mean/Variance optimization setting. We then looked into the theory of utility. We introduced the CRRA utility function and then looked into the SQP-method for direct expected utility maximization, which is an alternative to the

Mean/Variance optimization framework, by directly optimizing investors utility. In the last part we looked into some common pitfalls of backtesting.

To summarize, in this chapter we built the basis for the different approaches to portfolio optimization used in the case study in the practical part of this thesis. In the next chapter we build the theoretical foundation of Bayesian methods and Markov chain Monte Carlo.

3 Bayesian statistics and Markov Chain Monte Carlo

We begin this chapter, by showing how the Bayesian theorem (Bayesian formula) is derived, then we take a dive into the concept of Bayesian inference and how it is used to make conclusions using the so called a-posteriori distribution.

We proceed to methods and algorithms to numerically solve the Bayesian inference problem. We will begin by introducing Monte Carlo methods in general and how the use Random Number Generators to answer specific questions. We then proceed to the ideas of Markov Chain Monte Carlo and take a detailed look at two specific MCMC algorithms, namely Metropolis-Hastings and Hamilton Monte Carlo.

In the last section we introduce probabilistic programming languages (PPL), and in particular Stan which is one instance of such a PPL. PPL provide a domain specific language for doing Bayesian inference using MCMC in a declarative and user friendly way.

We use the notation $\mathbf{x}$ (bold) to denote a stochastic variable, vector or matrix, as opposed to x which is an ordinary scalar, vector or matrix.

3.1 The Bayesian theorem and Bayesian inference

The goal of Bayesian inference is to make conclusions on a parameter $\mathbf{\Theta}$ of the model based on observed, fixed data X. In the Bayesian world all parameters are random variables. We are usually interested in the so called a-posteriori density function of the parameters conditioned on the observed data $p_{\mathbf{\Theta} \mid \mathbf{X}}(\Theta \mid x)$, which incorporates all knowledge about the parameters $\mathbf{\Theta}$ after seeing the data, in particular it tells us, how "probable" each of the possible parameter assignments $\mathbf{\Theta}$ is. The a-posteriori distribution can be calculated with the Bayesian theorem after defining the likelihood $p(X \mid \Theta)$, and $p(\Theta)$ the a-priori distribution of $\mathbf{\Theta}$.

The Bayesian theorem (Bayesian formula) directly follows from the definitions of conditional probabilities and the law of total probability (Gelman et al. (2014)) and is defined for two continuous random variables $\mathbf{X}$ and $\mathbf{\Theta}$ as:

$$\begin{aligned} p(\Theta \mid X) &= \frac{p(\Theta, x)}{p(X)} \\ &= \frac{p(X \mid \Theta) \cdot p(\Theta)}{p(X)} \\ &= \frac{p(X \mid \Theta) \cdot p(\Theta)}{\int_{\Theta} p(X \mid \Theta) \cdot p(\Theta)\, d\Theta} \end{aligned} \tag{28}$$

$p(X \mid \Theta)$ is the likelihood, defining the probability density of the dataset $\mathbf{X}$, given one particular assignment of the parameter vector $\mathbf{\Theta}$.

$p(\Theta)$ is the a-priori distribution of $\mathbf{\Theta}$ and this distribution includes everything that we know about the parameter a-priori the inference, i.e. before we observe the data. A-priori distribution can be *non-informative*, which can be interpreted as a distribution, which expresses no particular subjective believe about the distribution of $\mathbf{\Theta}$, although there is no commonly agreed definition of what non-informative prior is and how it should be constructed. The principle of insufficient reason, originally formulated by Bernoulli and Laplace, Sinn (1980), says that, if you know nothing about $\mathbf{\Theta}$ you should use a probability which assigns equal probability for every possible outcome, i.e. a uniform distribution. This seems an obvious choice, but the use of uniform priors is often problematic because it is improper when using an unbounded parameter space in $\mathbf{\Theta}$ (Syversveen (1998)). An improper prior doesn't integrate to one, this might or might not be a problem, in some cases the posterior is proper anyways.

As we can see the choice of non-informative priors is not straightforward, often the prior is chosen based on the principles of information theory. One choice is to use a prior distribution which maximizes the entropy, and therefore minimizes the embedded information in the distribution based on some constraints of the parameter space (Shore and Johnson (1980)). This is called the principle of maximum entropy (Maxent), for example the probability distribution which maximizes the entropy for the parameter space $(0, \infty)$, with the constraint that the expected value $\mathbb{E}(\mathbf{\Theta})$ must exist, is the exponential distribution, in contrast, if we relax the constraint and say the expected value doesn't have to exist, the Maxent distribution would be the unbounded uniform distribution (Conrad (2013)).

Another popular choice is to use Jeffrey's prior (Jeffreys (1946)) which chooses a prior such that $p(\Theta) \propto \sqrt{\det \mathbb{I}(\Theta)}$, where $\mathbb{I}(\cdot)$ is the Fisher information, which also is a measure of the amount of information of an random variable.

For practical reasons, often prior distributions are chosen, which make the Bayesian formula analytically tractable. This is done using so called conjugate families (Bernardo and Smith (1994)). An a-priori distribution is chosen (the conjugate prior), so that the a-posteriori distribution has the same "structure" (distribution family) as the a-priori distribution, and therefore making to solve the Bayesian formula convenient. When using a Hamilton Monte Carlo algorithm, analytical tractability is no issue, so the a-priori distribution can be chosen which best represents the a-priori knowledge, whether it is analytically tractable or not.

3.2 Monte Carlo methods and Random Number Generators

Monte Carlo methods are a broad base of numerical algorithms, which use repeated random sampling using random number generators (RNG's) to answer specific mathematical questions, that would otherwise hard or even impossible to answer using plain analytical derivations. Monte Carlo methods were pioneered by Enrico Fermi, Stanislaw Ulam and John Von Neumann in the 1930's and 1940's and are used extensively in many scientific fields, including Finance, Physics, Chemistry and Medicine, since then.

Since Monte Carlo algorithms rely on random numbers, in this section we talk about different ways to generate random numbers, and in particular how a computer as a deterministic device generates random numbers (Shonkwiler and Mendivil (2009)).

In the beginning we must distinguish between "real" random numbers, which are created by a nondeterministic source, and pseudo random numbers, which are created by deterministic devices and try to "mimic" real random sequences.

Real random numbers are generated by measuring properties of nondeterministic physical processes. Examples of such physical random generators are measuring radioactive decay or measuring noise of a sensor. These processes are all well known to be nondeterministic.

A computer is a deterministic device, which means that for a given fixed input it always produces the same results, and therefore has no stochastic (random) components in itself, it is, by definition, impossible to generate real random numbers within a computer without using external sources of randomness (e.g. user input or physical randomness sources). Real random number generators are usually not available on consumer hardware without using special devices, e.g. a USB device which measures random noise, but since only some properties of real random numbers are wanted, in practice pseudo random number generators are used to simulate the behavior of real random numbers.

For Monte Carlo algorithms, the non-deterministic component is often unwanted, for example when the reproducibility of experiments is required. This is usually accomplished by providing the *seed* value, which is the initialization number of the pseudo random number generator. Given the seed number, the sequence of numbers which is generated by a pseudo random number generator is completely predictable when the RNG algorithm is known. This is usually not an issue when using Monte Carlo for numerical simulation, but it is unwanted when using RNG's for cryptographic applications, for example encryption. In this case the seed value used is usually a number which is hard to predict, for example the current time or the nanoseconds since the computer started up.

Pseudo random number generators usually generate $U(0,1)$ random numbers, i.e. a

sequence which is distributed uniformly in the interval $[0, 1]$. The inversion method Devroye (1986) allows us to generate sequences of any probability distribution if we know the quantile function F^{-1}, which is the inverse of the cumulative distribution function.

Let F be a right-continuous cumulative distribution function on $\mathbb{R}$ and quantile function F^{-1} with $F^{-1}(p) = \inf\left\{x \ : \ F(x) = p, 0 < p < 1\right\}$. If $\mathbf{u}$ is a random variable with $U(0, 1)$ distribution, then $F^{-1}(\mathbf{u})$ has cumulative distribution function F (Inversion method):

$$P(F^{-1}(\mathbf{u}) \leq x) = P(\inf\{y \ : \ F(y) = \mathbf{u}\} \leq x) \tag{29}$$

$$= P(\mathbf{u} \leq F(x)) \tag{30}$$

$$= F(x) \tag{31}$$

Example 3.1 (Generation of exponential distributed random variables)**.** The exponential cumulative distribution function is defined as $F(x) = 1 - e^{\lambda x}$, for $x \geq 0$. We calculate the inverse of the cumulative distribution function $F^{-1}(p) = \frac{-\ln(1-p)}{\lambda}$. In this example we generate $N = 10000$ samples of $U(0, 1)$ distributed numbers using the R function *runif*, and then plug the numbers in the quantile function $X = F^{-1}(U)$, which should result in exponential distributed numbers. To check the results the histogram of the samples U and X and the theoretical density functions (red line), as well as the empirical cumulative density function of X and the theoretical cumulative density function (red dashed line), are plotted. As expected, X is exponential distributed. In this example we set $\lambda = 5$.

```
N = 10000
lambda = 5

U = runif(N, min = 0, max = 1)
X = -log(1 - U) / lambda
```

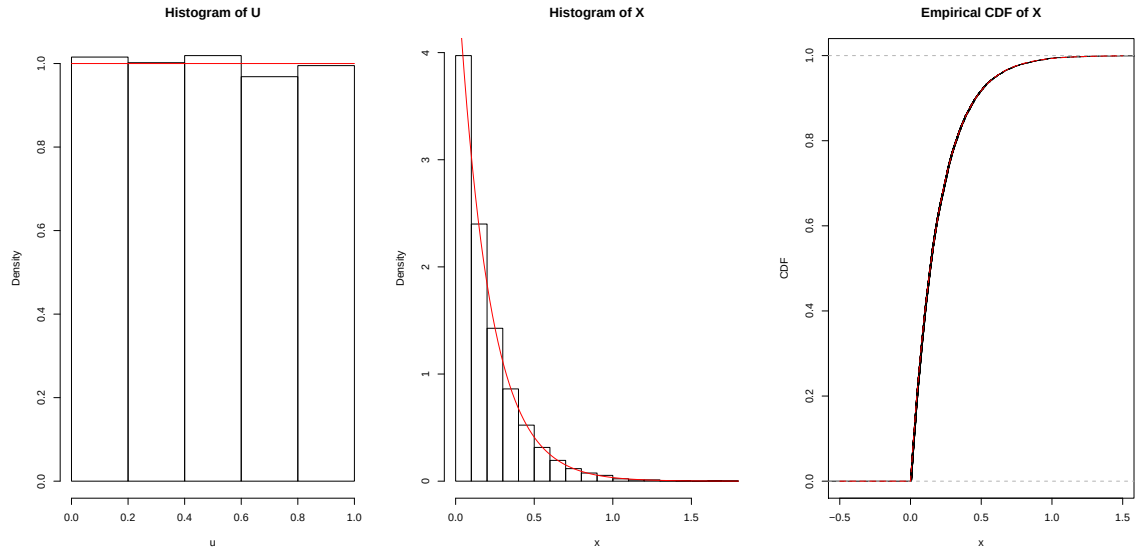

Figure 3: Example 3.1: Histogram of samples U and X, Empirical CDF of X, red lines are the theoretical functions

3.2.1 Monte Carlo integration

One instance of a Monte Carlo based algorithm is Monte Carlo integration.

Suppose we can obtain identically and independently distributed (i.i.d.) samples $x^{(i)}$ from some random variable $\mathbf{x}$ with density $p(x)$, then we can use the arithmetic average of the function of the samples to approximate a definite integral over the density function and some function f using the strong law of large numbers (Andrieu et al., 2003; Jordan, 2010). The arithmetic average converges almost surely (a.s., which means with probability 1) to the definite integral, as the number of samples N goes to infinity:

$$\frac{1}{N}\sum_{i=1}^{N} f(x^{(i)}) \xrightarrow{\text{a.s.}} \int_{x\in X} f(x)\, p(x)dx \qquad \text{when } N \to \infty \tag{32}$$

We can use this form to approximate e.g. the expected value $\mathbb{E}(\mathbf{x})$ using $f(x) = x$, or the variance $Var(\mathbf{x})$ using $f(x) = (x - \mathbb{E}(x))^2$.

The rate of convergence of the arithmetic average is proportional to $\sqrt{N}$ but this proportionality constant increases exponentially with the dimension of the integral (Jordan, 2010).

As said, for this method we have to be able to directly sample from $\mathbf{x}$, however for Bayesian inference, we cannot obtain independent samples from the a-posteriori

distribution in the general case, because this involves solving the Bayesian formula, which could be very hard or even intractable.

There are other Monte Carlo methods, such as Rejection sampling or Importance sampling, which allow to sample from $\mathbf{x}$ using a proposal distribution which is similar to $p(x)$ but easier to sample from. However easy-to-sample from proposal distributions may be impossible to obtain (Andrieu et al. (2003)), or the samples suffer from high variance in higher dimensions (Andrieu et al. (2003)).

In the next section we are going to explore a different idea, by sampling not independent, but slightly dependent samples from $\mathbf{x}$ using Markov Chains.

3.3 Markov Chain Monte Carlo

In the last section we mentioned Monte Carlo algorithms, which can generate independent samples from some random variable $\mathbf{x}$. However we also noted, that when using these methods, the variance of the samples increases greatly in higher dimensions.

Markov Chain Monte Carlo is a different strategy for generating samples from some random variable $\mathbf{x}$. Instead of obtaining independent samples, a MCMC procedure generates samples dependent on the last step.

We introduce the Markov chain theory according to Andrieu et al. (2003), Jordan (2010) and Chang (2007) and using finite state spaces, i.e. $\mathbf{x}$ can only take discrete values.

Markov chain A sequence $\mathbf{x^{(i)}}$ of random variables is called a Markov chain if it holds the Markov property:

$$p(x^{(i)} \mid x^{(i-1)}, ..., x^{(1)}) = p(x^{(i)} \mid x^{(i-1)}) = T(x^{(i)} \mid x^{(i-1)}) \tag{33}$$

To specify a Markov chain (Chang (2007)), a state space $\mathbb{S} \in \{1, 2, ..., M\}, \mathbf{x^{(i)}} \in \mathbb{S}$, an initial probability distribution for $\mathbf{x^{(0)}} \sim p_0(x^{(0)})$ and a probability transition function $T(\mathbf{x^{(i)}} = x \mid \mathbf{x^{(i-1)}} = x')$ is needed. If the state space is finite, one can specify a matrix $P \in [0, 1]^{M \times M}$, where $P_{rc} = T(\mathbf{x^{(i)}} = x_c \mid \mathbf{x^{(i-1)}} = x_r)$, i.e. each row r defines the current state and each column defines the probability going to the the state c conditional on being on the row r. Each row must, of course, sum to one.

Example 3.2 (Example Markov chain specification)**.** We define a Markov chain with state space $\mathbb{S} = \{A, B, C\}$, starting point $p_0 = A$ and transition matrix

$$P = \begin{pmatrix} 0 & 0.2 & 0.8 \\ 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \end{pmatrix}$$

In Figure 4 we can see a graphical representation of the Markov chains, where the circles represent the states and edges represent the transition probabilities. In this example, the initial state is A then the probability of going from state A to B is $P_{12} = 0.2$ and going to state C is $P_{13} = 0.8$.

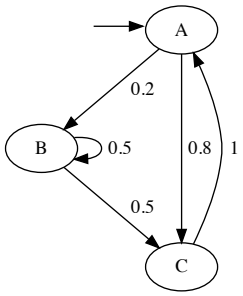

Figure 4: Experiment 3.2: Graphical representation of the Markov chain specified in the example

Markov chains for MCMC For MCMC, one chooses the transitions for the Markov chain, such that the target distribution $p(x)$ is invariant for the Markov chain (also called stationary distribution or equilibrium distribution of the Markov chain):

$$p(x) = \sum_{x'} T(x \mid x')\, p(x') \tag{34}$$

A sufficient, but not necessary condition to show that a particular $p(x)$ is the invariant distribution, is the detailed balance condition (also called reversibility condition):

$$p(x')\, T(x \mid x') = p(x)\, T(x' \mid x) \tag{35}$$

The sufficiency of this condition can easily be seen by summing both sides with all possible values for x[6]:

$$\sum_{x'} p(x')\, T(x \mid x') = \sum_{x'} p(x)\, T(x' \mid x) = p(x) \tag{36}$$

Another important property of the Markov Chain for MCMC is, that one chooses the transitions as such, that they eventually converge to the chosen invariant distribution $p(x)$. This is true if the transition function holds the following properties:

Irreducibility For every possible state of the Markov chain, there is positive probability

[6]Note: $\sum_{x'} T(x' \mid x) = 1$, because all outgoing transition probabilities must, of course, sum to 1

of visiting all other states:

$$\forall x_0, x_1 \in \mathbb{S} : \sum_{i=0}^{\infty} P(X_i = x_1 \mid X_0 = x_0) > 0 \tag{37}$$

Aperiodicity The chain cannot get trapped in cycles: An irreducible Markov chain is defined as aperiodic, if its period[7][8] is 1.

When we construct a Markov chain with these two properties we have an asymptotic guarantee, that the individual distributions of each $\mathbf{x^{(i)}}$, which are dependent on i eventually converge to the invariant distribution.

Let $\mathbf{x^{(0)}}, \mathbf{x^{(1)}}, ...$ be a Markov chain with invariant distribution $p(x)$ which is irreducible and aperiodic, then for all initial distributions p_0 of $\mathbf{x^{(0)}}$ (i.e. starting points) holds (Basic limit theorem of Markov chains, Chang (2007)):

$$\lim_{i \to \infty} p_i(x) = p(x) \tag{38}$$

3.3.0.1 Summary

In this section, we introduced the Markov chain theory as the basis for all MCMC algorithms, and specifically which properties of a Markov chain are important for MCMC.

A MCMC sampling algorithm constructs a Markov chain which has a specific target distribution as the invariant distribution. In case of Bayesian inference the target distribution is the a-posteriori density distribution of a Bayesian model. We can use the detailed balance condition to show, that the target distribution is the invariant distribution.

It is also of interest, to show if an Markov chain converges to this invariant distribution independent of the starting point. We can use irreducibility and aperiodicity conditions to show that the chain eventually converges to the target distribution. When using MCMC in practice, there are measures, for example R-hat the possible scale reduction factor (see Section 3.7.4.5), to check whether a chain has converged to the stationary distribution or not.

In the next sections, we are going to introduce the actual MCMC algorithms, which are methods to construct Markov chains with such properties.

[7] Given a Markov chain $\mathbf{x^{(i)}}$ with transition matrix P, the period d_i of state i is defined as $d_i = gcd\{n : P_{ii}^n > 0\}$, where gcd{x} is the greatest common divisor of set x

[8] All states in an irreducible Markov chain have the same period

3.4 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm, introduced by Metropolis et al. (1953) and further generalized by Hastings (1970) is one of the simplest and easiest to understand MCMC algorithms.

As said, the goal a MCMC algorithm is now to construct a Markov chain, whose invariant distribution is the distribution of interest, i.e. we want to "walk through" (to sample from) the parameter space in proportion to the target distribution $p_{\boldsymbol{\pi}}(x)$ (which is usually the a-posteriori density of a Bayesian model in the case of Bayesian inference $p_{\boldsymbol{\pi}}(x) = p(\Theta \mid X)$).

The algorithm works by first choosing a candidate point x' randomly sampled from a proposal distribution (also called jumping rule) $\mathbf{q} \mid \mathbf{x^{(i)}}$. This distribution is not fixed and is conditional on the current point in the chain. Usually a proposal distribution is chosen whose expectation is the current point $\mathbb{E}(\mathbf{q} \mid \mathbf{x^{(i)}}) = x^{(i)}$, although this is not required. A number c is sampled from a $U(0,1)$ distribution. If $c > \alpha(x^{(i)}, x')$ the candidate point is accepted setting $x^{(i+1)} := x'$ otherwise the chain does not move and stays at its current position $x^{(i+1)} := x^{(i)}$. The function α, which defines the candidate point acceptance is only dependent on the current point in the chain and is given by

$$\alpha(x, y) = \min\left(1, \frac{p_{\boldsymbol{\pi}}(y)\; p_{\mathbf{q} \mid \mathbf{x^{(i)}}}(x \mid y)}{p_{\boldsymbol{\pi}}(x)\; p_{\mathbf{q} \mid \mathbf{x^{(i)}}}(y \mid x)}\right)$$

Listing 1: Metropolis-Hastings Algorithm

```
x^(1) = x^(start)

for i in 1:N
  x' = sample from q | x^(i)
  c = sample from U(0,1)
  a = α(x^(i), x')

  if a > c:
        x^(i+1) = x'
  else
        x^(i+1) = x^(i)
  end
end

output x^(1), ..., x^(i)
```

A discussion of the proof of the Metropolis-Hastings can be found in Appendix A.

3.4.0.1 Choosing a proposal distribution

The choice of the right proposal distribution is not straightforward, since the acceptance probability of the candidate depends on the proposed point. When we consider a proposal distribution with high variance, it will quickly converge to the target distribution since it makes large steps, however when converged the acceptance probability will not be high since it often makes proposals which are in the tail of the target distribution making $p_{\boldsymbol{\pi}}(x')/p_{\boldsymbol{\pi}}(x^{(i)})$, and therefore the acceptance probability small. In contrast when the variance of proposal distribution is small, we have the exact opposite problem, making the convergence slow when starting at a point way out of the high density interval of the target distribution, but having a high acceptance probability.

Example 3.3 (The Metropolis-Hastings algorithm and different proposal distributions)**.** We have a dataset of $I = 20$ data points X_i, $i = 1...I$ and we know the data points to be independently and identically normal distributed with unknown mean and known $\sigma_{real} = 0.2$. We want to infer the probability density of the mean $\boldsymbol{\mu}$ using the Metropolis algorithm. Since we only want to infer the mean we have a one dimensional parameter space. In this example we assume we know that the parameter lies in the interval $[-10, 10]$ so we choose a uniform prior $U(-10, 10)$, for which the probability density is given by $p(\mu) = \frac{1}{20}$ for $-10 \leq \mu \leq 10$ and $p(\mu) = 0$ otherwise. In this example we generate the dataset by sampling 20 values from a normal distribution with $\mu_{real} = 0.4$ and $\sigma_{real} = 0.2$.

The Gaussian likelihood, defining the probability density of the dataset $\mathbf{X}$, given one particular assignment of the parameter vector $\boldsymbol{\Theta}$, is in this case given by the product of the likelihood of the individual data points, since we assume they are independent $p(X \,|\, \mu) = \prod_{i=1}^{N} p_{\text{norm}}(X_i \,|\, \mu, \sigma)$, where p_{norm} is the Gaussian probability density function.

To calculate the a-posteriori distribution of the mean parameter, we can use the Bayesian formula, and since we use a MCMC algorithm we only need to specify a function which is proportional to the posterior, i.e. the a-priori distribution times the likelihood:

$$p(\mu \,|\, X) = \frac{p(X \,|\, \mu)\; p(\mu)}{\int_{\mu \in M} p(X \,|\, \mu)\; p(\mu)\; d\mu} \propto p(X \,|\, \mu)\; p(\mu)$$

We run the experiment three times with 2000 steps, using different Gaussian-distributed proposals with $\sigma = 0.01$, $\sigma = 0.1$ and $\sigma = 0.4$.

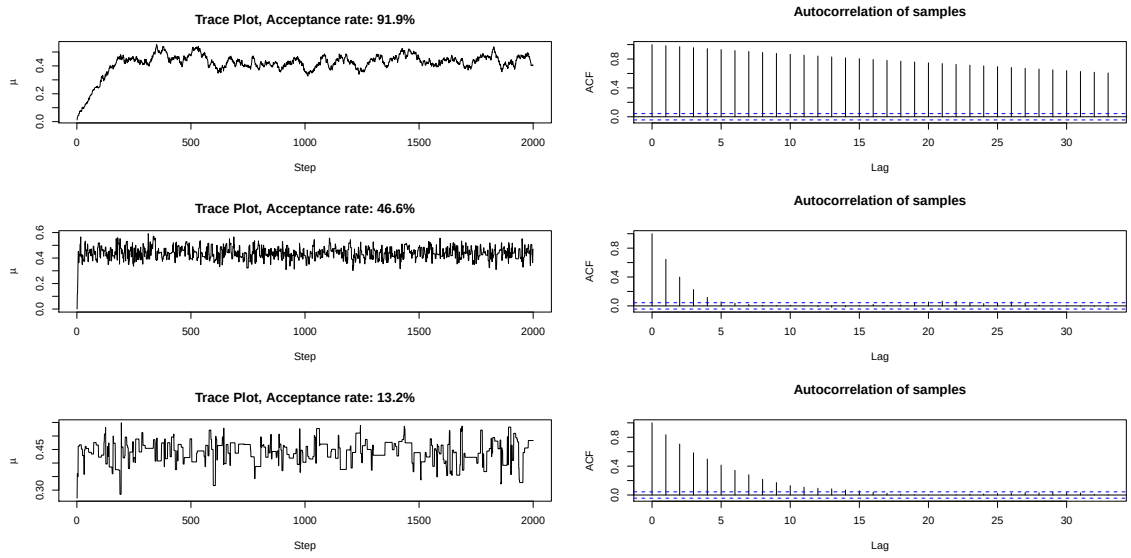

Figure 5: Experiment 3.3: Three different runs of the Metropolis-Hastings algorithm using Gaussian-distributed proposals with $\sigma = 0.01$ (top), $\sigma = 0.1$ (middle) and $\sigma = 0.4$ (bottom). The different proposal distributions lead to different paths in the random walk and different acceptance rates (left, trace plot of the random walk) and different autocorrelations in the random walk (right, ACF plot for the random walk)

As we can see, a low standard deviation in the proposal distribution leads to a high acceptance rate, since the proposed points lie in the area with a high acceptance probability. However the points are highly autocorrelated which leads to a lower effective sample size (see Section 3.7.4.4). A higher standard deviation in the proposal distribution leads to a smaller acceptance probability but also to a lower autocorrelation.

Since the random walk samples represent the a-posteriori distribution, which is in our case the a-posteriori distribution for the μ parameter, we can further analyze the samples, e.g. calculating the mean, or the posterior interval. The $100(1-\alpha)\%$ posterior interval corresponds to the range of values above and below in which lies $100(\frac{\alpha}{2})\%$ of the posterior probability (Gelman et al., 2014). We calculate the a-posteriori mean using the empirical mean of the random walk samples, and the 95% posterior interval (PI) using the 5% quantile and the 95% quantile of the obtained samples (i.e. ordering all values and taking the 5%/95% smallest value). Table 1 shows the results of the experiment.

As we increase the step size or tweak the proposal distribution of the metropolis algorithm, the samples should better represent the "real" a-posteriori distribution (but

	σ	mean	95% PI
1	0.01	0.434	[0.371, 0.497]
2	0.1	0.436	[0.367, 0.514]
3	0.4	0.440	[0.380, 0.510]

Table 1: Example 3.3: Results of the three experiments: σ of the proposal distribution, Empirical average and the 95% posterior interval of the a-posteriori parameter $\boldsymbol{\mu} \mid \mathbf{X}$

not necessary making the posterior interval smaller).

The R-Code for this example can be found in Appendix 16.

3.5 Hamilton Monte Carlo

One major flaw, characterizing all Metropolis-Hastings-based methods, is the inefficiency in models with many parameters (i.e. a high-dimensional target distribution), needing a long time to discover the whole space of interest in their random walk (Gelman et al. (2014)). There are ways around this unwanted inefficiency, such as re-parameterization or more appropriate jumping distributions (exploring the target function with a higher acceptance rate), however this workarounds are often model specific.

In contrast **Hamilton Monte Carlo (HMC, also called Hybrid Monte Carlo)** uses a different idea by introducing an auxiliary momentum variable for each parameter to suppress this unwanted random walk behavior. Hamilton Monte Carlo was introduced by Duane et al. (1987) and this section is mainly based on the explanations provided by Gelman et al. (2014), Stan Development Team (2016) and Hoffman and Gelman (2011).

3.5.0.1 Intuition

First we want to give a little intuition about how the HMC algorithm works, and how it is in many cases more efficient compared to the Metropolis algorithm. Image we have a one-dimensional target function with one mode (like the Gaussian density function), and the most recent step in the sampling process was in the left tail. Since we are using MCMC the next step will be dependent on the step before (Markov property), and ideally the next step would be to the high density region (i.e. somewhere in the middle). So now lets look how the next step proposal differs between the two algorithms.

Using the Metropolis-Hastings-Algorithm, we are using a conditional proposal distribution centered on position of the last step, with some arbitrary, pre-chosen standard

deviation. The proposal is a random relative step to the left or right[9] (see Section 3.4). When a step to the left is chosen, the proposal is likely to be rejected (because the density in the target function at a more left point is lower). Since the proposal distribution doesn't incorporate the shape of the target function, this is called a random walk, and thus leading to a high rejection rate, which is inefficient, because more steps are needed.

Now lets look how the HMC algorithm chooses the next step proposal. HMC uses, as the name suggests, Hamilton dynamics, which is a concept from theoretical physics, to avoid the random walk behavior of the Metropolis-Hastings algorithm. Hamilton dynamics fully describes a physical system in terms position and momentum vectors plus some equations on how these vectors change over time. The important part here is, that the whole dynamics only describe an exchange between kinetic energy and potential energy, where the energies are a function of the position and momentum vectors, but the total energy will always be constant. One could image a (frictionless) swinging pendulum, where at the highest points the kinetic energy is zero and the whole energy is in the potential part, and on the lowest point, in the middle, where the whole energy of the system is in the kinetic part.

In simple terms, in HMC, the last step is taken as the position, then some random momentum is chosen, and the potential energy is set to be proportional to the negative density of the target function at this position. Then the Hamiltonian system is simulated for some amount of time, and the resulting position is taken as the next proposal step. The important part here is the choice of the potential energy function: When we continue the thought experiment from above, having a Gaussian target function, one could image a marble which is placed on the left side of a U-shaped curve (i.e. turning the target function upside-down), where the curve here acts as a analogous to the potential energy function. We push the marble in a random direction, and then take the marble position after one second as the next proposal step. As one can image a marble in a U-shaped curve, if we push the marble in some random direction, after some time, gravity will force the marble back to the middle, and also regions with high potential energies (the sides) require stronger pushes. Back to MCMC, when we imagine the marbles position at some time point as the proposed steps, this behavior is exactly what we want, proposing more positions in low potential energy regions (and thus high density regions), leading to a higher acceptance rate.

[9] unless the proposal distribution function is specifically tuned

3.5.0.2 Description of the Algorithm

Again, our goal is to sample from a target distribution function $p_{\boldsymbol{\pi}}(x)$ (which is usually the a-posteriori density $p(\Theta \mid X)$).

As said, the HMC algorithm uses a special auxiliary momentum variable called ρ, which is of the same dimension as $\boldsymbol{\Theta}$ (i.e. for each parameter in the target function, there is a corresponding momentum value). A zero-mean multivariate Gaussian distribution, which is independent from the parameters $\boldsymbol{\Theta}$, is assigned to $\boldsymbol{\rho}$ such that $\boldsymbol{\rho} \sim MultiNormal(0, \Sigma)$, and a new joint density function is defined as $p(\Theta, \rho) = p(\Theta) \cdot p(\rho)$. [10]. Σ is used to tune the performance of the integrator (see below).

Using a concept from statistical mechanics called canonical distributions (Brooks et al. (2011)) allows us to relate a random variable to a energy function by taking the negative logarithm of the density function. Using this canonical distribution, the joint density function now defines the Hamiltonian, describing the total energy of the system, as the sum of the kinetic energy $T(\rho)$ and potential energy $V(\Theta)$:

$$H(\rho, \Theta) = T(\rho) + V(\Theta) = -\log p(\rho) - \log p(\Theta)$$

Using the Hamiltonian equations, which describe the dynamics of the position and momentum vector, we can now derive the behavior of our system:

$$\frac{d\Theta}{dt} = \frac{\partial H}{\partial \rho} = \frac{\partial T}{\partial \rho} \tag{39}$$

$$\frac{d\rho}{dt} = -\frac{\partial H}{\partial \Theta} = -\frac{\partial V}{\partial \Theta} \tag{40}$$

We are now left with two differential equations which must be discretized in order to be simulatable by a computer. Discretization works by dividing the system into a subset of small discrete time points and iteratively approximating the state for each time step, while ideally keeping the discretization error as small as possible.

The leapfrog algorithm is such an advanced discretization method, which is specifically tuned for the Hamiltonian equation system and it preserves the reversibility and volume preservation properties of the Hamilton system in the discretization step, which is important for the theoretical implications of the resulting Markov Chain.

The leapfrog method is started by choosing a time interval (also called step-size) ϵ, which sets the discretization step-size when advancing the system using the derived

[10] This is the most common setup for $\boldsymbol{\rho}$ and also used by Stan, variants using other distributions for $\boldsymbol{\rho}$ may exist

dynamics of the Hamiltonian system. The leapfrog algorithm works by taking a half step on the momentum, a full step on the position, and finally a half step on the momentum, which results in a total step by ϵ for both variables (for a detailed discussion of the leapfrog algorithm see Gelman et al., 2014):

$$\rho = \rho - \frac{\epsilon}{2}\frac{\partial V}{\partial \Theta} \tag{41}$$

$$\Theta = \Theta + \epsilon \frac{\partial T}{\partial \rho} = \Theta + \epsilon \, \Sigma \, \rho \tag{42}$$

$$\rho = \rho - \frac{\epsilon}{2}\frac{\partial V}{\partial \Theta} \tag{43}$$

After the leapfrog method has done an pre-chosen amount of steps, in the last step of the HMC algorithm a Metropolis-Hastings acceptance step is performed. This is only required because of discretization discrepancy compared to the real dynamics. The proposed step can be rejected, when the total energy of the new state is higher than the total energy of the old state:

$$P(\text{accept proposed step } \Theta') = \min(1, e^{H(\rho^{(i)},\Theta^{(i)}) - H(\rho',\Theta')}) \tag{44}$$

Listing 2: Hamilton Monte Carlo Algorithm

```
input parameters:
  Θ: starting position, vector of size S
  V(Θ): potential energy function
  dV(Θ): gradient of the potential energy function
  ϵ: Leapfrog step-size
  L number: of Leapfrog steps to take
  N number: of samples

define kinetic energy function T(ρ) = 1/2 Σ_{i=1}^{S} ρ_i^2
P = vector of size N
for t in 1:N
  ρ = sample from S-dimensional Standard Multivariate Gaussian
  Θ_old = Θ; ρ_old = ρ

  simulate L Leapfrog steps with ρ, Θ, using step size ϵ and gradient dV(
      ↪ Θ)

  p_accept = min(1, exp( (V(Θ_old) + T(ρ_old)) - (V(Θ) + T(ρ)) ))

  c = sample from U(0,1)
  if p_accept > c:
    P^(t) = Θ
  else
    P^(t) = Θ = Θ_old # rejected, continue on position Θ_old
```

```
  end
end

output P^(1), ..., P^(t)
```

11

A proof that the Hamilton Monte Carlo algorithm produces a proper Markov Chain for MCMC can be found, for example, in Brooks et al. (2011).

3.5.0.3 Discussion

In this section we introduced the Hamilton Monte Carlo algorithms. The HMC algorithm is often more efficient in higher dimensions by avoiding the random walk behavior of the Metropolis-Hastings algorithm.

But the HMC is also not without problems in practice: we have to choose parameters of the algorithm, such as the step size and number of steps for leapfrog and also in practice the covariance matrix Σ of $\boldsymbol{\rho}$ has to be chosen to improve the performance of the algorithm. Also the gradient of the parameter vector in terms of the potential energy function has to be derived manually.

In the next section we introduce the probabilistic programming language Stan which uses the NUTS algorithm, which attempt to solve some of the discussed caveats of the HMC algorithm.

3.6 Probabilistic Programming Languages

Probabilistic programming languages (PPL) are programming languages define a domain specific language (DSL) for describing statistical models and doing inference on these models. For the inference task, they often use Markov chain Monte Carlo algorithms, although this is not required (e.g. Infer.NET a PPL framework for .NET languages uses a message-passing algorithm).

Advantages of using a PPL over a custom implemented algorithm is obviously its simplicity. With a PPL we are concentrating on the "what" task, i.e. we describe a statistical model using a custom language, which this is a more declarative approach. When implementing the algorithm manually, the model is already embedded within the algorithm, this follows a more imperative approach, concentrating on the "how" part. Of course, a custom algorithm provides the most flexibility but often this flexibility is not needed, and other aspects such as ease of use and verifiability are more important, which favors a PPL approach.

[11] Σ is assumed to be the identity matrix, so no correlation between the variables

3.7 Stan

We decided to use Stan as the PPL for this thesis, so in this section we will give a kind introduction and then will further introduce the features of Stan and the structure of a Stan program.

The Stan project was initially started by Gelman and Hill (Stan Development Team (2016)) after discovering shortcomings with BUGS (Lunn et al. (2000)) and JAGS (Hornik et al. (2003)), which are probabilistic programming languages based on Gibbs sampling, a Metropolis-Hasting based approach, when they wanted to fit a multilevel generalized linear model in their paper Gelman and Hill (2007). At this time they got more involved into the Hamiltonian Monte Carlo algorithm which was able to overcome the shortcomings of Metropolis-Hasting based approaches (see Section 3.5). Since then, the Stan project evolved continuously and has now developed a strong community and users from many different fields.

3.7.1 The NUTS algorithm

Stan uses the Hamiltonian Monte Carlo algorithm or in particular a modification of the HMC algorithm called NUTS, the No-U-turn sampler. In Hoffman and Gelman (2011) they propose a method to automatically determine the two parameters of the HMC algorithm, namely the step size and the desired number of steps, which they called the No-U-Turn sampler. To set the number of steps, the method uses a recursive procedure to build a list of possible trajectories along the target distribution while avoiding retracing ("no U-turns"). They show that this procedure of choosing the number of steps performs at least as efficient as manually tuning the parameter. The step size is optimized using a optimization method called "primal-dual averaging", and therefore the NUTS algorithm needs no manual parameter input at all, which makes it very suitable for use in a computing environment like Stan.

3.7.2 Reverse-mode algorithmic differentiation

One disadvantage when choosing the HMC algorithm in favor of the Metropolis-Hastings approach, is that the HMC algorithm needs gradient information (i.e. the first derivatives of all parameters of the target distribution). In general, manually deriving the gradient is no problem, however it can be quite cumbersome, error prone and also time intensive, especially when used in a computing environment like Stan, in which the user focuses on the modeling part rather than on the technical details of the underlying algorithms.

Stan uses a method which is able to automatically derive the required gradients of the target distribution called reverse-mode algorithmic differentiation, avoiding these manual steps. In this paragraph, which give a brief overview of how this algorithm works.

Reverse-mode algorithmic differentiation (short Reverse-mode AD, also called automatic differentiation, Carpenter et al. (2015)) is an algorithm which computes the value of the gradient of a function f automatically. Automatic differentiation, unlike numerical derivation methods such as finite differencing, is exact (up to the machine precision). Also Automatic differentiation may not be confused with Automatic Symbolic differentiation, as used for example in Mathematica, whose output is the entire symbolic derivative function expression whereas algorithmic differentiation only outputs the values for the derivative. Although Automatic Symbolic differentiation has its place for certain applications (e.g. when a symbolic expression for a derivative is needed, for example in a Computer Algebra System), it has also certain difficulties (see Carpenter et al. (2015)). It might seem, that Automatic Symbolic differentiation is more powerful, because it works with symbols, however the opposite is the case, in fact Automatic differentiation is more expressive in the sense that Automatic differentiation can be used to calculate the derivative not only of functions in the mathematical sense but also derivatives of computer programs which include loops, conditionals and function calls.

Algorithmic differentiation uses the meta-programming features of programming languages such as C++ (the Stan Reverse-mode algorithmic differentiation is implemented in C++) to transform the program or function of interest into an expression graph. This expression graph only consists of subexpressions, for which the derivatives are known (e.g. simple expressions like $a + b$ or functions like $sin(x)$) and then the chain rule $\frac{\partial f}{\partial x_i} = \sum_{j=1}^{N} \frac{\partial f}{\partial u_j} \frac{\partial u_j}{\partial x_i}$ is used to calculate the derivative (u_j in this case represent the subexpressions). First the subexpressions and all partial derivatives thereof are determined (called forward pass). In the following reverse pass, each subexpression is assigned a adjoint value, for the root node (the output value), the adjoint value is 1 for all the others it is 0. The expression graph is traversed, multiplying the adjoint values, which then represent the derivatives, and therefore after the pass, the adjoints of the input variables contain the gradients.

3.7.3 The syntax of a Stan program

Stan is an imperative domain specific language (Stan Development Team (2016)). Imperative means, it is possible to use imperative language constructs such as conditionals, loops and functions. Domain specific means, that a Stan program defines a program for a specific domain and is not meant for general purpose programming.

In a nutshell, a Stan program defines a conditional probability function $p(\Theta \mid X)$ using a specialized domain specific syntax, where $\boldsymbol{\Theta}$ stands for the unknown parameters of the model and X for the collection of all observed data.

Since the syntax of Stan is mostly based on ordinary programming languages we won't go into much detail onto how commonly known things like assignments or loops works, instead we only talk about the details, which aren't that common in other programming languages or behave different from what a typical user would expect.

Stan is strongly and statically typed, every variable must have a declared type which cannot change. There are two primitive types *real* for continuous variables and *int* for integer values. Stan also provides data-types for Vectors (*vector* and *row_vector*), matrices (*matrix*), arrays (e.g. *real xs[4]*, defines a array with name *xs* of type *real* and length 4) and specialized types, which are often used in statistical models (e.g. *corr_matrix* which defines a matrix which is positive definite, symmetric, unit diagonal and whose entries are between -1 and 1).

Constraints for data types are also allowed in a Stan program (e.g. *real<lower=0>x*, defines a non negative real). For input parameters, the constraints provide a mechanism for error checking and for parameters the constraints provide implicit priors (see below).

As said, a Stan program defines a conditional probability function and when executed, the model block is executed once for each sample. The model block is so to say, the heart of a Stan program, which returns the conditional probability of the model. In Section 3.4 we demonstrate the structure and syntax of an Stan program based on an example.

3.7.4 Stan in practice

3.7.4.1 Interfacing with R

In a practical application, Stan is used through an interface (Stan Development Team (2016)). A Stan interface is used to control the compilation and the execution (i.e. the sampling) of a Stan program, through the interface the sampler also receives its input parameters and after the sampling process is finished, the output samples are also received through the interface.

In this thesis we concentrate on the R interface RStan, since R is our computing environment of choice, so all parameters will be fed in through R and also the post-run analysis is done with R.

3.7.4.2 Number of chains and startup

Since the starting point of a MCMC simulation is random in the general case, the iterative simulation may take a number of samples to find the high probability space of the target distribution. Therefore early samples may not be representative for the target distribution and are usually discarded for further calculations. These initial samples are usually called warm-up or burn-in samples.

There is no common agreed number, of how many initial samples should be discarded and that number also depends on the algorithm and the parameters. Gelman uses the general conservative practice of discarding the first of all samples.

Another parameter of a MCMC simulation is how many individual chains should be run. Each individual chain is independent and does one execution of the particular MCMC algorithm. At the end all samples obtained by the individual chains, after discarding the warm-up samples, are then collected into a common pool.

There is no common agreed number of how many chains should be used and if its better to run one very long or more but shorter chains. In practice often the number of chains is set to the number of CPU cores, since each chain is independent and can run on one core, and so the computing power is efficiently distributed.

3.7.4.3 Traceplot

The traceplot is a plot of the sampled parameters against the iteration number. The traceplot shows us how the MCMC explored the parameter-space, and can be used to visually assess the convergence and mixing behavior of individual parameters.

3.7.4.4 Effective sample size

By definition of the Markov property each sample obtained by the Markov chain is dependent on the value of the prior sample $x^{(i)} \mid x^{(i-1)}$. This means the correlation between two adjacent samples, i.e. the autocorrelation, is not 0. This is usually taken into account when calculating the effective sample size, since it is smaller when the samples are more correlated, since each sample gives marginally less information of the distribution of interest.

The effective sample size as proposed by Gelman et al. (2014) is:

$$N_{eff} = \frac{N}{1 + 2\sum_{k=1}^{\infty} \rho(k)} \tag{45}$$

where N is the total number of samples, N_{eff} the effective sample size, and $\rho(k)$ the

autocorrelation of the samples at lag k. In practical applications the infinite sum is stopped when $\rho(k) < 0.05$ because usually $\rho(k+1) < \rho(k)$ holds (Kruschke (2014)).

How many effective samples are sufficient, and therefore how long the chain should run, depends on the particular application and the measure of interest.

3.7.4.5 R-hat

R-hat is a measure of the convergence of a chain (Gelman et al. (2014)). When a chain converged, it should have mixed and be stationary. It works by splitting up each chain at the half and computing a measure which uses between- and within-chain variance. In the limit $n \to \infty$ R-hat approaches one. Intuitively, if the possible scale reduction is high, further samples can improve the accuracy of the estimate of the target distribution. For practical applications, Gelman and Hill (2007) suggest a R-hat threshold of 1.1. In the case study we also use this threshold, rerunning the simulation when R-hat is greater 1.1.

3.7.4.6 Example usage of Stan

Example 3.4. We want to show an example use of Stan using the setup from Example 3.3:

We have a dataset of $N = 20$ data points X_i, $i = 1...N$ and we know the data points to be mutually independent and identically normal distributed with known $\sigma = 0.2$. We want to the a-posteriori distribution of $\boldsymbol{\mu}$. We assume we know that the parameter lies in the interval $[-10, 10]$ so we choose a uniform prior $U(-10, 10)$ on $\boldsymbol{\mu}$.

We could incorporate this setup into a Stan model as follows:

Listing 3: Stan example

```
data {
  int N;
  vector[N] x;
}
parameters {
  real mu;
}
model {
  mu ~ uniform(-10, 10);
  x ~ normal(mu, 0.2);
}
```

In the data section we define all input parameters of the model. In the parameter section we define our $\boldsymbol{\mu}$ parameter as a real variable. The model section then defines the uniform prior on $\boldsymbol{\mu}$ and the normal distribution for $\mathbf{X} \mid \boldsymbol{\mu}, \boldsymbol{\sigma} = \mathbf{0.2}$.

Having defined the model, we then can start the sampling process by calling within R:

Listing 4: R

```
res = stan(model_code=stan_model, data=list(N=length(x), x=x), chains=1,
    ↪ warmup=200, iter=2000, init=0)
```

We run the sampler using the same configuration as in Example 3.3, using a warm-up size of 200, total number of steps of 2000, and starting point at 0.

We can use the *summary*(*res*) function to get a overview of the inferred parameters. The approximated expected value of $\boldsymbol{\mu} \mid \mathbf{X}$ is 0.438, and the 95% posterior interval[12] of $\boldsymbol{\mu} \mid \mathbf{X}$ is $[0.345, 0.522]$.

The effective sample size obtained by the sampling process was $N_{eff} = 663$, and the R-hat value was 1.0004098, which is a indicator that the chain has successfully converged to the target distribution.

When we compare the trace plot and autocorrelation function (Figure 3.4), we see that the resulting Markov chain has less autocorrelation than those of Example 3.3.

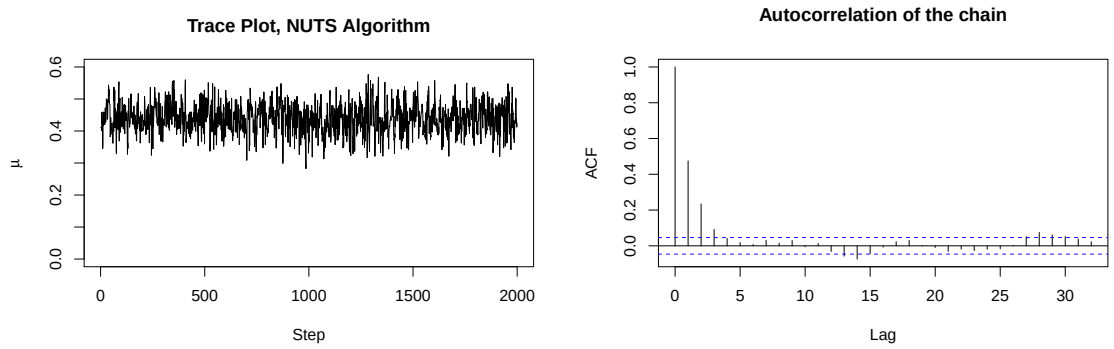

Figure 6: Experiment 3.4: Stan example using an HMC-based algorithm, Left: Trace plot of $\boldsymbol{\mu} \mid \mathbf{X}$, Right: Autocorrelation of the samples of $\boldsymbol{\mu} \mid \mathbf{X}$

As we could see in this example, using Stan is very convenient in practice, because it only requires us to specify the model in a declarative way. Because we are using an Hamiltonian Monte Carlo based algorithm, we would normally be required to specify the step size and number of steps for leapfrog and also the covariance matrix Σ of $\boldsymbol{\rho}$. The NUTS algorithm is used by Stan to find good values for these parameters in the warm-up period. Also the gradient of the parameter vector is needed, however Stan automatically calculates these by using the automatic algorithmic differentiation method.

[12]The $100(1-\alpha)\%$ posterior interval corresponds to the range of values above and below in which lies $100(\alpha/2)\%$ of the posterior probability (Gelman et al., 2014)

The R-Code for this example can be found in Appendix 17.

3.8 Summary

In this chapter we introduced the Bayesian theorem as the basis for Bayesian inference. We further explored the difficulties in the practical applications of Bayesian inference when using complicated models, which are often hard to solve or even mathematically intractable.

We then proceed to Monte Carlo algorithms to numerically solve the Bayesian inference problem, and thus avoiding this mathematical inconvenience. We saw, although these methods can be used to do Bayesian analysis numerically, they suffer from high variance in higher dimensions.

We introduced MCMC as an alternative to sample from arbitrary probability distributions using a purpose-built Markov chain. We explored the Metropolis-Hastings algorithm, however we saw, that these algorithm can be inefficient in high dimensions and when using complicated models. An advanced MCMC algorithm, namely Hamilton Monte Carlo is then introduced which aims to solve this efficiency issue by avoiding a random walk.

In the last section we introduced probabilistic programming languages which provide a domain specific language for doing Bayesian inference using MCMC in a declarative and user friendly way. We further focused on the probabilistic programming language called Stan which will be used in the case study of this thesis. We introduced the techniques, which Stan uses, in particular an HMC based algorithm called NUTS, and algorithmic differentiation to algorithmically obtain the gradients of the target distribution.

In the next chapter, we now use the built theoretical basis from the last two chapter to conduct a case study and investigate whether the Bayesian models are superior to the classical Mean/Variance MLE approach.

4 Empirical evidence

In this chapter we conduct a historical backtest analysis based on a dataset of the industries of the S&P 500 and a time period between 1989 and 2016, comparing the portfolios constructed by the classical model and the Bayesian models. We use a non-hierarchical and a hierarchical multivariate normal Bayesian model with two different priors. We are considering two different portfolio optimization methods: The classical Mean/Variance optimization and an expected utility maximization optimization approach using the isoelastic utility function. The dataset used consists of monthly percentage returns of the 10 industry sectors of the S&P 500 index between October 1989 and August 2016. For all historical estimations we use a look-back period of $12 \cdot 5$ months.

We report the out of sample mean squared prediction error of the models, the Certainty Equivalent return and do an analysis using the Fama-French-Carhart model using the realized returns of the constructed portfolios for each of the models and for each of the two optimization methods.

The purpose of this case study is to investigate whether the Bayesian models are superior to the classical Mean/Variance MLE approach based on above criteria.

This chapter is organized as follows: The dataset and the preprocessing thereof is introduced, then the models and the optimization procedure is presented in detail. The backtest experiment setup is then discussed. At the end of the chapter we present the results and discuss them.

4.1 The dataset

4.1.1 Preprocessing the dataset

In order to be used in our backtesting procedure the dataset must be preprocessed. Our goal is to construct indices representing the S&P 500 industry sectors. The dataset comes from the data provider Thomson Reuters Datastream.

The dataset comes in `.Rraw` files which can be directly loaded by R using the `load()` function, and are then available in the environment. Table 4.1.1 explains the contents of the raw files.

Filename	**Contents**
Rf_SimpR_W_USD	Time series representing the percentage returns of the risk free asset.
FirmInfo	Contains meta data of the firms, most importantly the industry in which the firm is operating (Column "ICBIN").
IsinIX_W	Contains a matrix with binary values, where the row names represent the date and the column names the firm. The value 1 in row r and column c means, that firm c at date c was constituent in the S&P 500. The value 0 means the firm was not a constituent in the S&P 500 at that particular date.
Benchmark_RIAbsSimpR_W_USD	Time series representing the percentage returns of the S&P 500 Total Return index.
RIAbsSimpR_W_USD	A matrix representing the total returns (i.e. price return plus dividends) of the individual firms.
MV_W_USD	A matrix representing the total market capitalization (= number of share outstanding * share price) of the individual firms.

Table 2: Individual files of the dataset

The indices are constructed as following: First the firms are grouped according to their industry. Then for each day, those firms are taken, which were constituent in the S&P 500 at this day. Each remaining firm in the industry is weighted as a fraction of its market capitalization over the total market capitalization.

An important thing is to lag the market capitalizations one day, i.e. for day t the market capitalization of day $t-1$ must be taken, because otherwise this would heavily inflate the returns, because the market capitalization of today was not known at the beginning at the day. Assume a firm which has market capitalization C at day t and experiences a positive return r on its stock price at that day. At the end of the day, ceteris paribus, the market capitalization would be higher $C' = C \cdot (1 + r)$ and therefore also would have resulted in a greater weight in the index and therefore inflating the result (the same argument applies for a negative return, resulting in a lower weight in the index). This would be an instance of look ahead bias (see Section 2.4.0.2).

We reconstructed the indices, but at this point we only have the returns, and the backtesting procedure expects price data, so we calculate the price (= index value) of each index for time t as $p_t = p_0 \prod_{i=1}^{t} 1 + r_i$, the start price p_0 is not important for our

purposes so we arbitrarily chose $p_0 = 100$.

To check the results, we then weighted all indices by their market capitalization which results in one time series, which should represent the S&P 500 Total Index. Since we have a file in the dataset with a precomputed S&P 500 index, we took the two return time-series and subtracted them, which theoretically should result in a zero-only time-series, since both time series represent the same index. In reality we got some small discrepancies, but we decided that these small differences are not significant since they are only about 0.05% per month (see Figure 7).

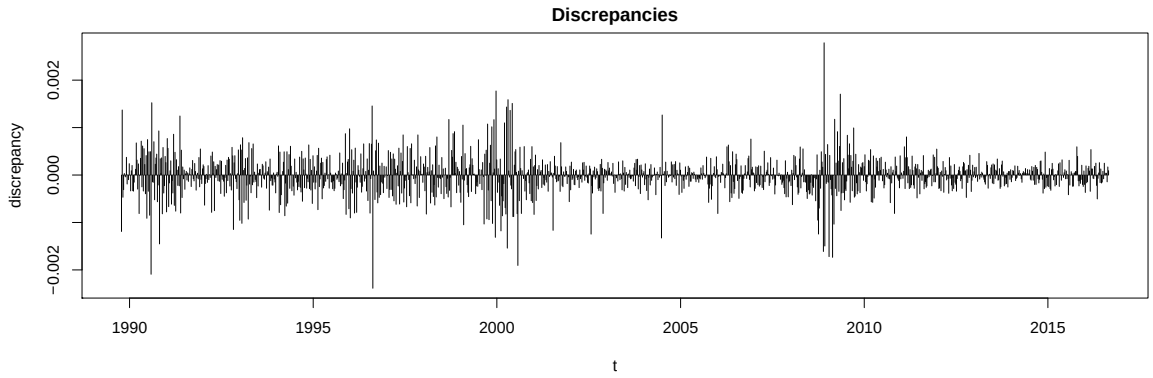

Figure 7: Difference of the constructed S&P 500 index using the industry indices and the actual S&P 500 index. For our purposes, the difference is not significant (about 0.05% per month).

At the end we saved the constructed sector indices, the risk free rate, and the S&P 500 index (as a proxy for the market) in the file `marketdata.Rdata`. The pre-processing script can be found in the Appendix Listing 9.

Summary statistics of the dataset can be found in Table 3, a time-series plot in Figure 4.1.1.

Industry	**mean/%**	**vola./%**	**total ret./%**
Industrials	11.0	17.4	1155.9
Consumer Services	11.0	16.5	1210.6
Basic Materials	9.3	21.4	550.6
Technology	14.0	25.2	1709.5
Oil & Gas	11.3	19.2	1161.8
Financials	10.4	21.5	765.0
Utilities	9.5	15.3	836.7
Consumer Goods	11.4	13.2	1566.1
Health Care	12.5	15.5	1952.9
Telecommunications	8.1	19.2	439.0

Table 3: Summary statistics of the dataset consisting of the 10 industries of the S&P 500 from Oct. 1989 to Aug. 2016: mean: annualized yearly mean return, vola.: annualized yearly standard deviation, total ret.: total return from Oct. 1989 to Aug. 2016

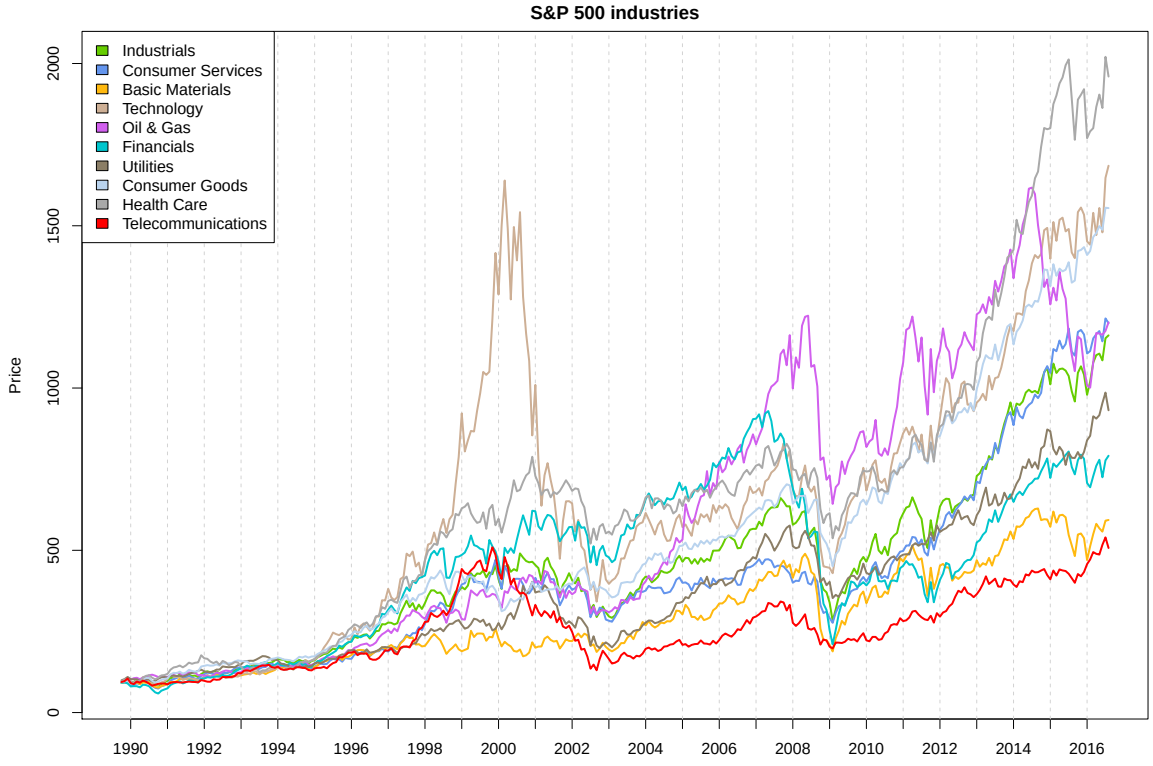

Figure 8: Price dataset of the 10 industries of the S&P 500 index from Oct. 1989 to Aug. 2016

4.2 Estimating the means and covariance matrix

In this section we present the used estimators for the means and covariance matrix of the returns ("the models"). We estimate μ and Σ in the case of Mean/Variance optimization or $\mathbf{R_{t+1}}$ in the case of direct expected utility optimization using the most recent 60 monthly observed historical returns of the financial assets $X_{ij} \in \mathbb{R}^{N \times S}$, where each i represents one month and each j one asset. First we describe the classical frequentist MLE approach and then the Bayesian based approach.

4.2.1 MLE estimation of the means and covariance matrix

As a baseline ("the classical model"), we use the standard Maximum Likelihood Estimations (MLE) approach for calibrating μ and Σ, which shall serve as a benchmark for the Bayesian models.

The MLE estimators for μ and Σ are:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{46}$$

$$\widehat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (X_i - \hat{\mu})(X_i - \hat{\mu})^T \tag{47}$$

4.2.2 Bayesian estimation of the means and covariance matrix

In this section we describe the Bayesian models which describe the marginal a-posteriori probability distribution of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. We then either calculate point estimates μ_P and Σ_P or numerically integrate out the parameters to obtain the predictive return distribution $\mathbf{R_{t+1}}$, for details see Section 4.3.

Multivariate Gaussian model

The multivariate Gaussian model is the Bayesian counterpart of the classical (frequentist) MLE model, since it uses only very weak priors. E.g. when we look at one $\boldsymbol{\mu_j}$ which uses a normal distribution with standard deviation 1, which is magnitudes higher than the expected range when using monthly return data (which lies in the range $\sim \pm 10\%/12 = \pm 0.008$).

4.2.2.1 Multivariate Gaussian distribution with an Inverse Wishart Prior for $\boldsymbol{\Sigma}$

$$\begin{aligned} X_i|\mu,\Sigma &\sim MultiNormal(\mu,\Sigma), \text{iid for each period } i \\ \mu_j &\sim Normal(0,1), \text{ for each asset } j \\ \Sigma &\sim InverseWishart(S+1, diag(1)) \end{aligned} \tag{48}$$

The Inverse Wishart distribution is often chosen as a prior distribution for the covariance matrix of a Multivariate Gaussian distribution of dimension S, since it is conjugate (see Section 3.1). The density function of the Inverse Wishart prior, as defined in Stan can be found in Stan Development Team (2016).

The Inverse Wishart prior $InverseWishart(\nu, \Lambda)$ has two parameters, the degrees of freedom $\boldsymbol{\nu}$ and $\boldsymbol{\Lambda}$ a positive definite matrix, with the same dimensions as the covariance matrix. We set $\boldsymbol{\nu} = S+1$ and $\boldsymbol{\Lambda}$ to the identity matrix, which implies a marginal uniform distribution for all correlations. Using the Inverse Wishart prior induces a inverse scaled chi-square distribution for each variance (Alvarez et al. (2014)).

The Stan file for the Multivariate Gaussian distribution with an Inverse Wishart Prior for $\boldsymbol{\Sigma}$ can be found in Appendix 5.

4.2.2.2 Multivariate Gaussian distribution using the separation strategy and a LKJ[13] prior for $\boldsymbol{\rho}$

$$\begin{aligned} X_i|\mu,\Sigma &\sim MultiNormal(\mu,\Sigma), \text{iid for each period } i \\ \mu_j &\sim Normal(0,1), \text{ for each asset } j \\ \Sigma &= diag(\sigma)\ \Omega\ diag(\sigma) \\ \sigma_j &\sim Cauchy(0,1), \text{ for each asset } j \\ \Omega &\sim LKJ(1) \end{aligned} \tag{49}$$

The Inverse Wishart prior is often used for $\boldsymbol{\Sigma}$ in a Multivariate Gaussian model because the maximum likelihood estimator of the covariance of a Multivariate Gaussian distribution is Inverse Wishart distributed. The Inverse Wishart distribution is not constructed to serve as an uninformative prior. Alvarez et al. (2014) discusses a property of the Inverse Wishart prior which is the dependency between the correlation and

[13]LKJ stands for the authors Lewandowski, Kurowicka, and Joe, who introduced this prior (Lewandowski et al., 2009)

variances, a necessary feature of the small sample distribution of covariance estimators, for which the Inverse Wishart distribution was originally described. Alvarez et al. (2014) in his paper, provides a good visualization for this behavior.

Here we use an alternative strategy for constructing a prior for the covariance matrix named separation strategy, proposed by Barnard et al. (2000) and described in Stan Development Team (2016) and also in Alvarez et al. (2014). The separation strategy works by decomposing the covariance matrix $\mathbf{\Sigma} = \text{diag}(\boldsymbol{\sigma})\, \boldsymbol{\rho}\, \text{diag}(\boldsymbol{\sigma})$[14] into a correlation matrix $\rho \in \mathbb{R}^{S \times S}$ and a standard deviation vector $\boldsymbol{\sigma} \in \mathbb{R}^S$. Then the prior densities for $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$ can be chosen independently and then they are also independent (in a statistical meaning) by design.

For the standard deviations, we chose a weak prior using the Cauchy distribution.

For the correlation matrix, we chose the LKJ prior, which is recommended by Stan Development Team (2016). The LKJ prior $LKJ(s)$ was introduced by Lewandowski et al. (2009) and has one shape parameter $\mathbf{s}$ and density $p(\rho \mid s) \propto det(\rho)^{s-1}$, where $det(\rho)$ is the determinant of the matrix ρ. We set $\mathbf{s} = 1$, which induces a uniform density on the correlation matrix: $p(\rho \mid s = 1) \propto 1$ (Stan Development Team (2016)). The LKJ prior is not conjugate, but since we use a HMC based algorithm this is no further problem.

Alvarez et al. (2014) points out a problem with the LKJ prior: in high dimensions ($d > 10$) the mass concentrates near zero and therefore favoring lower correlations. In our experiments we have a dataset of 10 assets and therefore $d = 10$.

The Stan file for the Multivariate Gaussian distribution using the separation strategy and a LKJ prior for $\boldsymbol{\rho}$ can be found in Appendix 6. In the implementation we use a numerical optimization which increases the efficiency and numerical stability of the sampling process by using the Cholesky factorization for $\boldsymbol{\rho}$, as recommended in Stan Development Team (2016).

Hierarchical Multivariate Gaussian model

As an alternative to the ordinary Multivariate Gaussian model described in the previous section, here we use a Hierarchical Multivariate Gaussian model, inspired by Greyserman et al. (2006). The advantage of using a Hierarchical model is that the hyper-parameter of the means should capture the common nuisance of all assets in the posterior and therefore leading to better predictions (Greyserman et al. (2006)). Greyserman et al. (2006) also notes, that the predictive distribution of hierarchical models are usually not analytically tractable, therefore numerical methods have to be used. The hierarchical

[14] Given a vector $x \in \mathbb{R}^N$ the $\text{diag}(x)$ function returns a matrix $\in \mathbb{R}^{N \times N}$ with the components of x on the diagonal

model introduces a common hyper-parameter $\boldsymbol{\mu}^{(\mathbf{hyp})}$ for the prior of all individual means $\boldsymbol{\mu}_\mathbf{j}$ of the Multivariate Gaussian distribution. Each $\boldsymbol{\mu}_\mathbf{j}$ prior is then set to be normal distributed with mean $\boldsymbol{\mu}^{(\mathbf{hyp})}$ and standard deviation $\boldsymbol{\sigma}_\mathbf{j}/S$. The prior for the hyper-parameter is set to be a weak prior (standard normal distribution). Again here we use both prior strategies for the covariance matrix, discussed in the preceding section.

The Hierarchical model induces shrinkage behavior which can be shown based on an example:

Example 4.1 (Shrinkage behavior of the hierarchical Gaussian model)**.** We generate 104 random return samples of a S-variate Gaussian distribution with "real" $\mu = 0$ and $\sigma = 1$ for all $j = 1..S$ and no correlations. We use the non-hierarchical Gaussian Model and the hierarchical Gaussian model to estimate the real μ_j using the generated samples. After we extracted the a-posteriori samples of $\boldsymbol{\mu}_\mathbf{j} \mid \mathbf{X}$ of the models, we calculate the mean thereof and compare it to the real value.

As we can see in Figure 9, the common hyper-prior for each of the $\boldsymbol{\mu}_\mathbf{j}$ in the hierarchical model uses all available information to form a common mean value. This hyper-prior is then used as the mean value for the $\boldsymbol{\mu}_\mathbf{j}$ prior, which leads to a shrinkage of the individual $\boldsymbol{\mu}_\mathbf{j}$ towards the hyper-prior. As we can see, this leads to a much smaller error for the individual $\boldsymbol{\mu}_\mathbf{j}$, since they lie closer to the real μ.

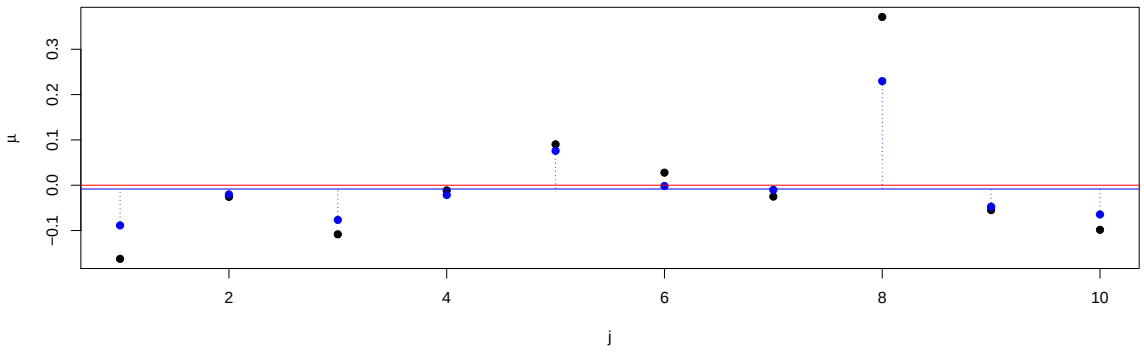

Figure 9: Example 4.1: Shrinkage behavior of the hierarchical model for a 104 sample estimation of standard multivariate normal distributed samples Red line: real $\mu = 0$ Black points: means of a-posteriori distributions $\boldsymbol{\mu}_\mathbf{j} \mid \mathbf{X}$ of the non-hierarchical Gaussian model Blue points: means of a-posteriori distributions $\boldsymbol{\mu}_\mathbf{j} \mid \mathbf{X}$ of the hierarchical Gaussian model Blue line: mean of a-posteriori distribution for the hyper-prior $\boldsymbol{\mu}^{(\mathbf{hyp})}$ of the hierarchical Gaussian model

4.2.2.3 Hierarchical Multivariate Gaussian distribution with an Inverse Wishart Prior

$$
\begin{aligned}
X_i|\mu, \Sigma &\sim Normal(\mu, \Sigma), \text{ iid for each period } i \\
\mu_j &\sim Normal(\mu^{(hyp)}, \frac{\sigma_j}{0.1 \cdot N}), \text{ for each asset } j \\
\mu^{(hyp)} &\sim Normal(0, 1) \\
\sigma^{(hyp)} &\sim InverseGamma(0.0001, 0.0001) \\
\Sigma &\sim InverseWishart(S + 1, diag(1) \cdot 1/\sigma^{(hyp)})
\end{aligned}
\tag{50}
$$

Here we also use an hyper prior for the scale of the covariance matrix ($\boldsymbol{\sigma}^{(\mathbf{hyp})}$) with an inverse gamma prior, as also used in Greyserman et al. (2006). The Stan file for the Hierarchical Multivariate Gaussian distribution with an Inverse Wishart Prior for $\boldsymbol{\Sigma}$ can be found in Appendix 7.

4.2.2.4 Hierarchical Multivariate Gaussian distribution using the separation strategy

$$
\begin{aligned}
X_i|\mu, \Sigma &\sim Normal(\mu, \Sigma), \text{ iid for each period } i \\
\mu_j &\sim Normal(\mu^{(hyp)}, \frac{\sigma_j}{0.1 \cdot N}), \text{ for each asset } j \\
\mu^{(hyp)} &\sim Normal(0, 1) \\
\Sigma &= diag(\sigma)\ \Omega\ diag(\sigma) \\
\sigma_j &\sim Cauchy(0, 1), \text{ for each asset } j \\
\Omega &\sim LKJ(1)
\end{aligned}
\tag{51}
$$

The Stan file for the Hierarchical Multivariate Gaussian distribution using the separation strategy and a LKJ prior for $\boldsymbol{\rho}$ can be found in Appendix 8. In the implementation we again use the numerical optimization based on the Cholesky factorization for $\boldsymbol{\rho}$.

4.3 The backtest setup

For the MCMC sampler we implemented some automatic convergence checks based on R-hat and also systematically inspected the created log files to detect possible divergences.

We follow the experiment setup used by Greyserman et al. (2006). Our data set consists of data from the 10 industries of the S&P 500 from October 1989 to August 2016, in

total 323 monthly periods. We use a lookback period of $12 * 5 = 60$ months, so we have in total $323 - 60 = 273$ out of sample periods to be used in the experiment.

4.3.1 Mean/Variance optimization

The procedure using Mean/Variance optimization of the portfolio works as follows:

1. Use the 60 trailing monthly returns to obtain a point-estimate of μ and Σ.

   a) The classical approach uses the Maximum likelihood estimators for estimating μ and Σ. This shall only serve as a "basis benchmark", as one of the simplest models, of course, more sophisticated frequentist models have been developed (e.g., see Kan and Zhou, 2007).

   b) The Bayesian approach uses the MCMC sampler to obtain the approximate conditional a-posteriori distributions of the random quantities $\boldsymbol{\mu} \mid \mathbf{X}$ and $\boldsymbol{\Sigma} \mid \mathbf{X}$ for each model. We then calculate the expected conditional predictive values as $\mu_p = \mathbb{E}(\mathbf{R_{t+1}}) = \mathbb{E}(\boldsymbol{\mu} \mid \mathbf{X})$ and $\Sigma_p = Var(\mathbf{R_{t+1}}) = \mathbb{E}(\boldsymbol{\Sigma} \mid \mathbf{X}) + Var(\boldsymbol{\mu} \mid \mathbf{X})$, for details see Section 2.2.2.

2. The point estimates of μ and Σ are used to calculate the Mean/Variance efficient portfolio w with risk aversion parameter $\lambda = 3$. We use the fully invested constraint (the portfolio weights must sum to one) and the short-sale constraint (each weight in the portfolio must be ≥ 1). There is no closed-form solution, thus we use a numerical quadratic programming solver (see Section 2.1).

3. Use w and calculate the out-of-sample return of the portfolio by multiplying the asset weight with the actual return of the next period for each asset.

4. Advance by one month and repeat steps 1-4. For the first period the return observations for month 1-60 are used to obtain the point-estimates for μ and Σ then the actual return of the portfolio is calculated with return for period 61, and so on.

4.3.2 Direct expected utility optimization based on the isoelastic utility function

Although in this case study we do not use the DEU approach with the "classical model" using MLE estimations of the means and covariance matrix this method is not Bayesian exclusive. By introducing additional assumptions (normal distribution of returns), also frequentist methods can be used with the direct expected utility optimization method (e.g., see Kan and Zhou, 2007).

The procedure with direct expected utility optimization of the portfolio works as follows:

1. Use the 60 trailing monthly returns to obtain the predictive return distribution $\mathbf{R_{t+1}}$. The MCMC sampler is used to obtain the approximate conditional a-posteriori distributions of the random quantities $\boldsymbol{\mu} \mid \mathbf{X}$ and $\boldsymbol{\Sigma} \mid \mathbf{X}$. $\boldsymbol{\mu} \mid \mathbf{X}$ and $\boldsymbol{\Sigma} \mid \mathbf{X}$ are numerically integrated out to obtain the approximate predictive return distribution for $\mathbf{R_{t+1}}$.

2. The approximate predictive return distribution for $\mathbf{R_{t+1}}$ is used to calculate the portfolio w using the direct expected utility maximization method using the CRRA utility function with risk aversion parameter $\eta = 3$. We use the fully invested constraint (the portfolio weights must sum to one) and the short-sale constraint (each weight in the portfolio must be ≥ 1). For details see Section 2.3.1.

3. Use w and calculate the out-of-sample return of the portfolio by multiplying the asset weight with the actual return of the next period for each asset.

4. Advance by one month and repeat steps 1-4. For the first period the return observations for month 1-60 are used, then the actual return of the portfolio is calculated with return for period 61, and so on.

In total we run 9 different experiments with the following configurations:

	Optimization approach	**Model**	**Ref**
1	Mean/Variance optimization using point estimates of μ and Σ	Frequentist Maximum Likelihood Estimators (MLE)	Sec. 4.2.1
2		Bayesian Multivariate Gaussian model with the Inverse Wishart prior for $\mathbf{\Sigma}$	Sec. 4.2.2.1
3		Bayesian Multivariate Gaussian model using the separation strategy and the LKJ prior for $\boldsymbol{\rho}$	Sec. 4.2.2.2
4		Bayesian Hierarchical Multivariate Gaussian model with the Inverse Wishart prior for $\mathbf{\Sigma}$	Sec. 4.2.2.3
5		Bayesian Hierarchical Multivariate Gaussian model using the separation strategy and the LKJ prior for $\boldsymbol{\rho}$	Sec. 4.2.2.4
6	Direct expected CRRA utility optimization based on the predictive return distribution $\mathbf{R_{t+1}}$	Bayesian Multivariate Gaussian model with the Inverse Wishart prior for $\mathbf{\Sigma}$	Sec. 4.2.2.1
7		Bayesian Multivariate Gaussian model using the separation strategy and the LKJ prior for $\boldsymbol{\rho}$	Sec. 4.2.2.2
8		Bayesian Hierarchical Multivariate Gaussian model with the Inverse Wishart prior for $\mathbf{\Sigma}$	Sec. 4.2.2.3
9		Bayesian Hierarchical Multivariate Gaussian model using the separation strategy and the LKJ prior for $\boldsymbol{\rho}$	Sec. 4.2.2.4

Table 4: Configurations of the conducted experiments

4.4 Analysis of results

In this section we report statistical and economical measures of the constructed portfolios of the out-of-sample experiments. In particular we report basic statistical measures, the R-squared point prediction accuracy measure, the certainty equivalent return and we do a regression analysis using the Carhart four factor regression model. In the last section we use these reported measures to compare and discuss the results. We use the notation R_t for the realized portfolio return at time t of one experiment.

4.4.1 Overview

Figure 10 plots the cumulative out-of-sample returns of the constructed portfolios of the conducted experiments.

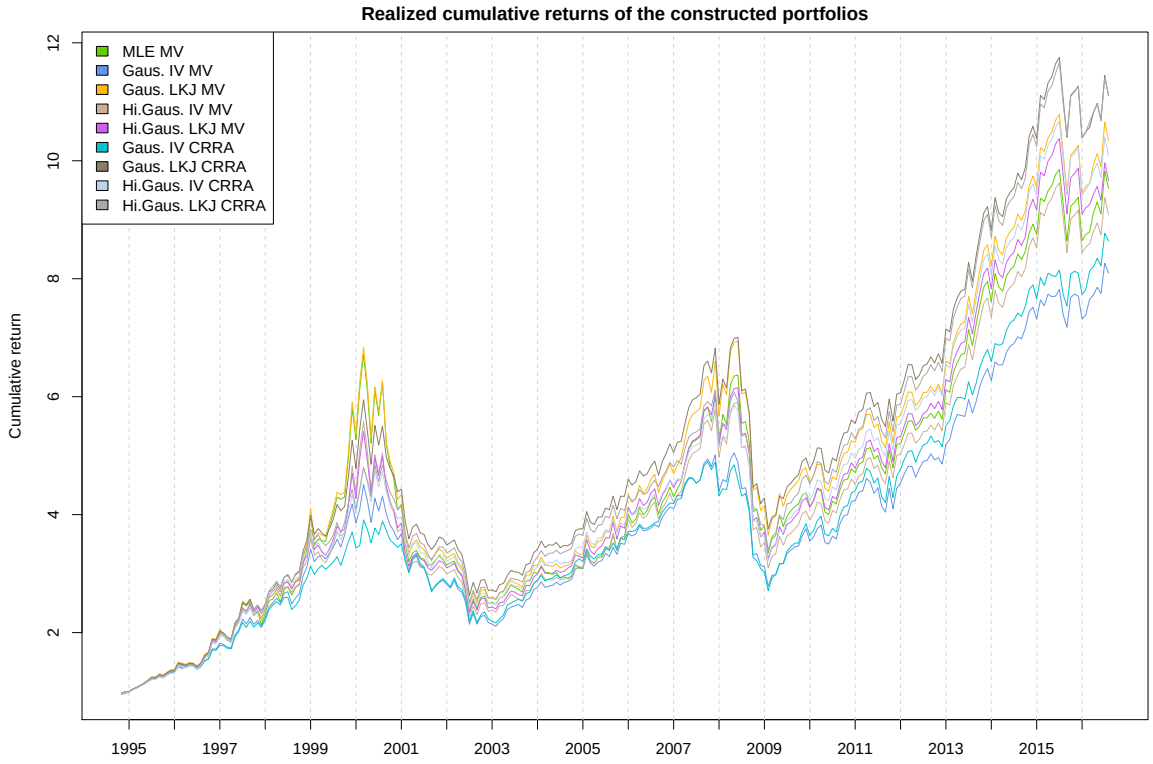

Figure 10: Cumulative out-of-sample returns of the constructed portfolios of the conducted experiments

In Table 5 we report the mean returns and standard deviations of the realized out-of-sample returns of the constructed portfolio for each experiment. Additionally we report the mean excess return which is defined as the average of the realized out-of-sample returns minus the risk free return of the period. As a risk adjusted return measure, we report the Sharpe ratio (Sharpe (1966)), which is defined as the quotient of the average excess return over the standard deviation of the returns. We annualized the reported measures by multiplying the reported returns by 12, the standard deviations by $\sqrt{12}$ and the Sharpe ratio also by $\sqrt{12}$.[15]

In the overview we see, all the portfolios of the Bayesian models have a lower yearly return volatility (ranging form 15% to 20.2%) than the classical model (20.5%). The realized yearly mean excess return is about equal or slightly lower (ranging from 10.3% to 8.4%) compared to the classical model portfolio (10%). When looking at the quotient of the excess returns divided by the standard deviation (i.e. the Sharpe ratio), all

[15]The Sharpe ratio is annualized by multiplying by $\sqrt{12}$ because: $\frac{\mu_{Pex} \cdot 12}{\sigma_P \cdot \sqrt{12}} = \frac{\mu_{Pex}}{\sigma_P} \cdot \sqrt{12}$

Bayesian portfolios dominate the classical portfolio (0.49), and also all portfolios which were optimized using DEU (ranging from 0.57 to 0.62) dominate their Mean/Variance counterparts (ranging from 0.51 to 0.54).

Optimization	Model	Prior	$\mu_P/\%$	$\sigma_P/\%$	$\mu_{Pex}/\%$	$\frac{\mu_{Pex}}{\sigma_P}$
Mean/Var	MLE		12.48	20.47	10.01	0.49
	Gaussian	Inv.Wis.	10.90	15.93	8.44	0.53
		LKJ	12.79	20.17	10.33	0.51
	Hier.Gauss.	Inv.Wis.	11.86	18.38	9.39	0.51
		LKJ	12.05	17.87	9.58	0.54
DEU (CRRA)	Gaussian	Inv.Wis.	11.05	14.98	8.59	0.57
		LKJ	12.69	17.91	10.23	0.57
	Hier.Gauss.	Inv.Wis.	11.95	16.13	9.48	0.59
		LKJ	12.35	15.89	9.89	0.62

Table 5: Annualized measures of the constructed out-of-sample portfolios for each experiment. μ_P: mean of the realized returns, σ_P: standard deviation of the realized returns, μ_{Pex}: mean of the realized excess returns, $\frac{\mu_{Pex}}{\sigma_P}$: realized Sharpe ratio

4.4.2 R-squared

In Table 6 we compare the models using the MSE, $RMSE$ and R-squared measure.

For each model we computed point-forecasts of the returns for each asset for the next period using the calibrated models using the historical data window.

For the classical model, which use the maximum likelihood estimators, we took the estimated mean μ as the point forecast value for the next period (see Section 4.2.1).

For the Bayesian models we took the mean of the predictive return distribution, which is simply the mean of the marginal a-posteriori distribution for $\boldsymbol{\mu} \mid \mathbf{X}$ (see Section 2.2.2).

After we obtained the point-forecast for the returns of the next period, we calculated the mean squared prediction error MSE as the squared difference from point forecasts for the next period and the actual value of the next period and then took the mean value over all assets and all time points. We also report the root mean squared prediction error $RMSE$, which is simply the square root of the MSE, $RMSE = \sqrt{MSE}$.

As the last step we calculated the R-squared measure (R^2_{OOS}) as one minus the quotient of the MSE of the model and the MSE of a benchmark model. In our case we chose the benchmark model to be the classical model.

$$R^2_{OOS} = 1 - \frac{MSE}{MSE_{\text{benchmark}}} \tag{52}$$

On the statistical measure, when looking the point forecast accuracy using the R-squared measure, the non-hierarchical Bayesian models perform worse than the MLE model, having a negative R-Squared value of -0.05% for the Gaussian model with Inverse Wishart prior for the covariance matrix, and -0.02% for the Gaussian model with the separation strategy and the LKJ prior for the correlation matrix. The hierarchical Bayesian models perform slightly better than the MLE model, having a R-Squared value of 0.71% for the hierarchical Gaussian model with Inverse Wishart prior, and 0.68% for the Gaussian model with the LKJ prior.

Model	Prior	MSE	RMSE	$R^2/\%$
MLE		0.003235	0.056881	0
Gaussian	Inv.Wis.	0.003237	0.056894	-0.045
	LKJ	0.003236	0.056887	-0.019
Hier.Gauss.	Inv.Wis.	0.003212	0.056677	0.714
	LKJ	0.003213	0.056687	0.679

Table 6: Statistical point prediction accuracy measures of the model for each experiment. MSE: mean squared prediction error of the return point forecast, $RSME$: root mean squared prediction error of the return point forecast, R^2: R-squared measure comparing the prediction accuracy of the model to the benchmark model

4.4.3 Certainty Equivalent Return

In Table 7 we compare the portfolios using the average realized utility, the average annualized certainty equivalent return (CER) and the average annualized certainty equivalent return gain. To compare the realized utilities, we use the differences in certainty equivalent return (McCulloch and Rossi (1990)). The CER gain is defined as the difference of the average CER of the experiment minus the average CER of the benchmark portfolio which is the portfolio of experiment 1.

The certainty equivalent return r_C is defined as the certain return, which would provide the same utility as the expected utility of the return on the risky asset:

$$u(r_C) = \mathbb{E}(u(\mathbf{r})) \tag{53}$$

In our case we calculate the expected utility of the return on the risky portfolio as the

average utility of the realized returns of the optimized portfolio:

$$u(r_C) = \mathbb{E}(u(\mathbf{r})) \approx \frac{1}{N}\sum_{i=1}^{N} u(R_i) \tag{54}$$

To calculate the CER for the CRRA utility function we have to invert it:

$$u_{CRRA}(r) = \frac{(1+r)^{1-\eta} - 1}{1-\eta} \tag{55}$$

$$r(u_{CRRA}) = ((1-\eta)(u_{CRRA} + \frac{1}{1-\eta}))^{\frac{1}{1-\eta}} - 1 \tag{56}$$

$$\tag{57}$$

The CER for the squared utility $r(u_{SQ})$ is simply the utility value, because the certainty return is a constant, having zero variance and expected value $\mathbb{E}(r) = r$:

$$u_{SQ}(\mathbf{r}) = \mathbb{E}(\mathbf{r}) - \frac{\lambda}{2} Var(\mathbf{r}) \tag{58}$$

$$r(u_{SQ}) = u_{SQ} \tag{59}$$

Then we can estimate the CER r_C for both utility functions with:

$$r_C = r(\mathbb{E}(u(\mathbf{r}))) \approx r(\frac{1}{N}\sum_{i=1}^{N} u(R_i)) \tag{60}$$

Table 7 shows the realized utilities, both for the squared utility function and CRRA utility function, of the portfolios of the conducted experiments. The target utility function for Mean/Variance optimization was the squared utility function with risk aversion parameter $\lambda = 3$. The target utility function for the direct expected utility optimization was the CRRA utility function with risk aversion parameter $\eta = 3$. To provide a complete picture, we report realized utilities for both utility functions.

From a utility perspective, when looking at the average yearly certainty equivalent return, all Bayesian portfolios dominate the classical portfolio, for both utility functions. Also all DEU optimized portfolios dominate the Mean/Variance portfolios in terms of average yearly CER. In total we see a yearly CER gain (compared to the classical model) of about 0.5% to 1% for Mean/Variance optimization and about 1.5% to 2.45% for DEU (for both utility functions). We can see, that both utility functions don't differ to much in terms of CER, so the choice of the particular utility function (Squared utility vs. CRRA utility) is not so important relative to the optimization method (Mean/Variance

optimization vs. Direct expected utility optimization), which has a rather big impact in the realized CER.

Optimization	Model	Prior	Squared Utility ($\lambda = 3$)			CRRA Utility ($\eta = 3$)		
			u	$r_C/\%$	Gain/%	u	$r_C/\%$	Gain/%
Mean/Var	MLE		0.0052	6.19	0	0.0050	6.00	0
	Gaussian	Inv.Wis.	0.0059	7.09	0.89	0.0057	6.94	0.94
		LKJ	0.0056	6.69	0.49	0.0054	6.50	0.49
	Hier.Gauss.	Inv.Wis.	0.0057	6.78	0.59	0.0055	6.60	0.60
		LKJ	0.0060	7.25	1.06	0.0058	7.07	1.06
DEU (CRRA)	Gaussian	Inv.Wis.	0.0064	7.68	1.49	0.0062	7.56	1.55
		LKJ	0.0066	7.87	1.68	0.0064	7.71	1.71
	Hier.Gauss.	Inv.Wis.	0.0067	8.04	1.85	0.0065	7.92	1.92
		LKJ	0.0071	8.56	2.37	0.0070	8.45	2.45

Table 7: Utility measures of the returns of the portfolios of each experiment: u: Average realized utility, r_C: average annualized Certainty Equivalent return, Gain: average annualized Certainty Equivalent return gain, The three measures are reported for using both the the squared utility function ($\lambda = 3$) and the CRRA utility function ($\eta = 3$)

4.4.4 Fama-French-Carhart four factor model

In Table 8 we report the results of the regression analysis using the four-factor model of Fama and French (2015), which tries to explain the realized returns of the portfolios using four different risk factors.

In their paper Fama and French (1993) Fama and French develop a regression model to explain returns of portfolios in the stock markets. The model is based on the idea that particular portfolios seem to have higher returns compared to the CAPM predicted return, where the return is only based on the sensitivity to the market. In particular, they use a model which consists of three explanatory variables (factor returns):

- the equity premium ($RMRF$), which is defined as the return on a value weighted portfolio minus the risk free rate (one month T-bill rate)
- small minus big (SMB), the return of small-cap portfolios minus large-cap portfolios
- high minus slow (HML), the return of high book-to-market portfolios minus low book-to-market portfolios

Carhart (1997) extended this model this model by introducing a fourth factor, which incorporates the well known momentum anomaly:

- the momentum premium (MOM), the return on a portfolio of past years winners (with the highest returns) minus past year losers (with the lowest returns)

The Fama-French-Carhart four factor model can then be defined as:

$$R_t - R_{RFt} = \alpha + \beta_{RMRF} RMRF_t + \beta_{HML} HML_t + \beta_{SMB} SMB_t + \beta_{MOM} MOM_t + \epsilon_t \tag{61}$$

R_t is the portfolio return at time t, R_{RFt} the risk free rate at time t. This quantity is now described as the sum of the excess return unexplained by the other factors α and the factors of the factor loading β_i times the factor returns at time t plus some error term ϵ_t, which captures the residual return not explained by the model.

This model can now be used to attribute the sources of the ex-post returns of an portfolio to the risk factors. We downloaded the factor returns for the four variables of the relevant time period, which is provided on French's website (French, 2018). We then did a ordinary regression using the downloaded factor returns and the returns of the portfolio of each conducted experiment. The results of the regression analysis is presented in Table 8.

Analyzing the realized returns of the portfolios using the Fama-French-Carhart four factor regression model we can explain where the additional return contributions came from. Unsurprisingly all portfolios have significant positive equity premium factor loadings. Additionally all portfolios had significant negative HML factor loadings, which suggests that the portfolios have a bias to low book-to-market (value) industries . Also all portfolios have also significant positive momentum factor loadings, which suggests that the portfolios have a bias for industries, which were successful in the recent past. This is unsurprising, because the portfolio selection method uses estimated returns based on recent historical return data. The non-hierarchical Gaussian model with Inverse Wishart prior and the portfolios using the hierarchical Gaussian models (but only with the DEU optimization method) have a significant negative SMB factor loading, suggesting a bias for industries with large-cap companies . Although all alpha factors are positive, which suggests additional return factors not explained by the four factor model, none alpha factor is statistically significant. To summarize the results of the regression analysis of the four factor model, all realized returns of the portfolios can be explained using the four risk factors of the model.

Optimization	Model	Prior	α	β_{RMRF}	β_{SMB}	β_{HML}	β_{MOM}
Mean/Var	MLE		2.9 (3.0)	** 11.1 (0.7)	-0.6 (0.9)	** -5.6 (1.0)	** 2.5 (0.6)
	Gaussian	Inv.Wish.	1.8 (1.9)	** 10.4 (0.5)	* -1.4 (0.6)	** -2.6 (0.7)	** 1.3 (0.4)
		LKJ	3.1 (2.9)	** 11.1 (0.7)	-0.6 (0.9)	** -5.2 (1.0)	** 2.6 (0.6)
	Hier. Gauss.	Inv.Wish.	2.5 (2.7)	** 10.3 (0.6)	-0.5 (0.8)	** -4.1 (0.9)	** 2.4 (0.5)
		LKJ	2.9 (2.6)	** 10.2 (0.6)	-0.6 (0.8)	** -3.9 (0.9)	** 2.2 (0.5)
DEU (CRRA)	Gaussian	Inv.Wish.	2.1 (1.7)	** 10.3 (0.4)	** -1.8 (0.5)	* -1.2 (0.6)	* 0.7 (0.3)
		LKJ	3.5 (2.5)	** 10.5 (0.6)	-1.1 (0.8)	** -4.1 (0.9)	** 2.2 (0.5)
	Hier. Gauss.	Inv.Wish.	3.2 (2.2)	** 9.8 (0.5)	* -1.4 (0.7)	** -2.9 (0.8)	** 1.6 (0.5)
		LKJ	3.6 (2.2)	** 9.8 (0.5)	* -1.5 (0.7)	** -2.7 (0.7)	** 1.5 (0.4)

Table 8: Coefficients of the Fama-French-Carhart four factor model analysis using the realized portfolio returns: $*\ \widehat{=}$ coefficient significant on the 5% level, $**\ \widehat{=}$ coefficient significant on the 1% level, Reported are the coefficients (coefficient standard deviation), all reported coefficients and coefficient standard deviations are multiplied by $12 \cdot 100$

4.4.5 Summary

To summarize the analysis, we found the realized returns of portfolios using the Bayesian technique have slightly lower volatility than the classical portfolio using the Mean/Variance MLE approach. They all have a higher risk-adjusted return (Sharpe ratio). From a forecast perspective, the Bayesian models do not or do only slightly provide better point prediction forecast ability. Also we found out, that all additional returns can be explained using economically meaningful factors using the Fama-French-Carhart four factor model. From a utility perspective, all portfolios have a higher certainty equivalent return than the classical portfolio. The gains range up to 1-2% additional certainty equivalent return per year, which could be economically significant. When directly comparing the models using the two different optimization methods, we saw that the direct expected utility optimization performed better than the Mean/Variance optimization method in every case. We also saw, that the hierarchical models tend to perform better than their non-hierarchical counterparts. Also the separation strategy and the LKJ prior tends to

perform better than the Inverse Wishart prior, although not in every case.

To conclude the analysis, we answer the research questions: Although the used Bayesian models do not provide better point prediction forecast ability, the use of Bayesian methods was justified by providing an utility gain which ranges up to 1-2% additional certainty equivalent return per year. Also the use of the direct expected utility optimization method was justified by performing better than the Mean/Variance optimization method. All additional returns of the Bayesian portfolios can be explained by the Fama-French-Carhart four factor model.

4.4.5.1 Limitations

Although, we tried to obtain a high validity of this case-study by trying to avoid commonly known pitfalls of backtesting, this case study also has some limitations. We did not incorporate trading costs, which in reality could lower the realized returns. In real-life scenario there might also be liquidity constraints. The limited time-period and investment universe of course only provides a limited picture of the validity of the conducted experiments and results. To get a more complete picture, the experiments can be repeated using other investment universes and/or time-periods.

5 Conclusion

In this thesis we investigated the Bayesian approach to portfolio selection. We used a non-hierarchical and a hierarchical multivariate Gaussian Bayesian model with two different priors. Due to the intractability of such complicated models, we used a probabilistic programming language called Stan, which uses the numerical Markov Chain Monte Carlo (MCMC) approach to do inference on these models. We considered two different portfolio optimization methods: The classical Mean/Variance optimization and an expected utility maximization approach (MEU). We wanted to answer, whether such advanced Bayesian models and optimization methods are superior to the classical Mean/Variance approach with ML estimators, and therefore the usage of such advanced models and optimization methods is justified.

In the first two chapters we began with the theory of portfolio selection, Bayesian methods and MCMC. To provide evidence to the asked research question, in the practical part of this thesis we conducted a historical backtesting analysis based on a dataset consisting of monthly percentage returns of the 10 industry sectors of the S&P 500 index between 1989 and 2016. We compared the constructed portfolios to the portfolio constructed by the classical approach. We found that the realized returns of the constructed Bayesian-based portfolios have slightly lower volatility than the classical portfolio. The Bayesian-based portfolios all have a higher risk-adjusted return (Sharpe ratio). From a forecast perspective, the Bayesian models do not or do only slightly provide better point prediction forecast ability. Also we found out, that all additional returns can be explained using the risk factors of the Fama-French-Carhart model. From a utility perspective, all Bayesian-based portfolios have a higher certainty equivalent return than the classical portfolio. The gains range up to 1-2% additional certainty equivalent return per year, which could be economically significant. When directly comparing the models using the two different optimization methods, we saw that the direct expected utility optimization performed better than the Mean/Variance optimization method in every case. We also saw, that the hierarchical models tend to perform better than their non-hierarchical counterparts. Also the separation strategy with the LKJ-prior tend to perform better than the Inverse Wishart prior, although not in every case. To summarize the analysis and answer the research question, although they do not provide better point prediction forecast ability, the use of Bayesian methods was justified, by providing an utility gain which ranges up to 1-2% additional certainty equivalent return per year. Also the use of the direct expected utility optimization method was justified by performing better than the Mean/Variance optimization method.

Appendices

A Outline of the proof of the Metropolis-Hastings algorithm

A discussion of the Metropolis-Hasting algorithm can be found in Section 3.4. Here we provide an outline of the proof that the Metropolis-Hastings algorithm does actually converge to the target distribution (inspired by Gelman et al. (2014)).

To proof the convergence we first have to show that the sequence of the samples $x^{(i)}$ is a Markov chain with a unique stationary distribution, this is the case when the Markov chain is irreducible, aperiodic and not transient[16]. Irreducibility is present, when every state can eventually be reached with positive probability. This is the case as long as no misbehaving proposal distribution is chosen. The other two properties, except for trivial exceptions, hold for a random walk over any proper distribution (Gelman et al. (2014)).

The second point is to show that the invariant distribution is the target distribution. To ease up the notation, we use the substitution $q(\cdot \mid \cdot) = p_{\mathbf{q} \mid \mathbf{x^{(i)}}}(\cdot \mid \cdot)$. Suppose, without loss of generality, we are at time $i-1$ and we just sampled $x^{(i-1)}$ from the target distribution $p_{\boldsymbol{\pi}}(x)$, and we have any two samples x^A, x^B with $p_{\boldsymbol{\pi}}(x^B)q(x^A \mid x^B) \geq p_{\boldsymbol{\pi}}(x^A)q(x^B \mid x^A)$. The probability to be at x^A and to transit to x^B is exactly the probability to be at point x^A, getting x^B proposed and getting x^B accepted by the acceptance rule. Since we defined x^A, x^B such that the expression $p_{\boldsymbol{\pi}}(x^B)q(x^A \mid x^B) \geq p_{\boldsymbol{\pi}}(x^A)q(x^B \mid x^A)$ the fraction within the min function is always greater 1 the whole expression reduces to 1.

$$
\begin{aligned}
p(\mathbf{x^{(i-1)}} = x^A, \mathbf{x^{(i)}} = x^B) &= p_{\boldsymbol{\pi}}(x^A) \cdot q(x^B \mid x^A) \cdot \alpha(x^A, x^B) && (62)\\
&= p_{\boldsymbol{\pi}}(x^A) \cdot q(x^B \mid x^A) \cdot \min\left(1, \frac{p_{\boldsymbol{\pi}}(x^B)q(x^A \mid x^B)}{p_{\boldsymbol{\pi}}(x^A)q(x^B \mid x^A)}\right) && (63)\\
&= p_{\boldsymbol{\pi}}(x^A) \cdot q(x^B \mid x^A) && (64)
\end{aligned}
$$

In the other case, the probability density to be at x^B and to transits to x^A is exactly as above but in this case the fraction within the min function is always less than 1 and the expression reduces to the fraction.

[16] According to Andrieu et al. (2003), the "not transient" property is not mandatory for convergence

$$p(\mathbf{x^{(i-1)}} = x^B, \mathbf{x^{(i)}} = x^A) = p_{\boldsymbol{\pi}}(x^B) \cdot q(x^A \mid x^B) \cdot \alpha(x^B, x^A) \tag{65}$$

$$= p_{\boldsymbol{\pi}}(x^B) \cdot q(x^A \mid x^B) \cdot \min\left(1, \frac{p_{\boldsymbol{\pi}}(x^A) q(x^B \mid x^A)}{p_{\boldsymbol{\pi}}(x^B) q(x^A \mid x^B)}\right) \tag{66}$$

$$= p_{\boldsymbol{\pi}}(x^B) \cdot q(x^A \mid x^B) \cdot \frac{p_{\boldsymbol{\pi}}(x^A) q(x^B \mid x^A)}{p_{\boldsymbol{\pi}}(x^B) q(x^A \mid x^B)} \tag{67}$$

$$= p_{\boldsymbol{\pi}}(x^A) \cdot q(x^B \mid x^A) \tag{68}$$

So we have $p(\mathbf{x^{(i-1)}} = x^B, \mathbf{x^{(i)}} = x^A) = p(\mathbf{x^{(i-1)}} = x^A, \mathbf{x^{(i)}} = x^B)$, so the joint distribution of $\mathbf{x^{(i-1)}}$ and $\mathbf{x^{(i)}}$ is symmetric, which means $\mathbf{x^{(i-1)}}$ and $\mathbf{x^{(i)}}$ must have the same marginal distribution and hence $p_{\boldsymbol{\pi}}(x)$ is the invariant distribution.

A detailed discussion of the algorithm can be found in Gelman et al. (2014) and in the original papers Metropolis et al. (1953) and Hastings (1970).

B Gradients of the target and constraint functions for Direct Expected Utility maximization of the CRRA-Utility function using Sequential Quadratic Programming

We want to maximize the expected utility of the CRRA-Utility function, keep the portfolio fully invested (weights must sum to one), disallow short sales and therefore set up the optimization problem as (for details, see Section 2.3.1):

$$\max_w \mathbb{E}\big[\, u(\mathbf{R}w)\big] \approx \max_w \frac{1}{N} \sum_{i=1}^{N} u(\sum_{j=1}^{S} R^{(ij)} w_j)$$

$$\text{s.t. } \sum_{j=1}^{S} w_j = 1 \text{ and } w_j \geq 0 \; \forall \; j \in 1..S$$

We can eliminate the $w_j \geq 0$ constraints by substituting $w_j = v_j^2$ (see 2.3.1). To switch from maximization to minimization we multiply the target function by -1.

$$f(v) = -\frac{1}{N} \sum_{i=1}^{N} u(\sum_{j=1}^{S} R^{(ij)} v_j^2) = -\frac{1}{N} \sum_{i=1}^{N} \frac{(1 + \sum_{j=1}^{S} R^{(ij)} v_j^2)^{1-\eta} - 1}{1 - \eta}$$

$$g_1(v) = \sum_{j=1}^{S} (v_j^2) - 1 = 0$$

For the SQP solver we need to derive the gradients of the target function and the constraint-function in terms of the weight vector. We derived the gradients as:

$$
\begin{aligned}
\frac{\partial f}{\partial v_g} &= \frac{\partial}{\partial v_g}\Big(-\frac{1}{N}\sum_{i=1}^{N}\frac{(1+\sum_{j=1}^{S} R^{(ij)}v_j^2)^{1-\eta}-1}{1-\eta}\Big) \\
&= \frac{\partial}{\partial v_g}\Big(-\frac{1}{N}\frac{1}{(1-\eta)}\sum_{i=1}^{N}(1+\sum_{j=1}^{S} R^{(ij)}v_j^2)^{1-\eta}-1\Big) \\
&= -\frac{1}{N}\frac{1}{(1-\eta)}\sum_{i=1}^{N}\frac{\partial}{\partial v_g}\Big((1+\sum_{j=1}^{S} R^{(ij)}v_j^2)^{1-\eta}-1\Big) \\
&= -\frac{1}{N}\frac{1}{(1-\eta)}\sum_{i=1}^{N}\frac{\partial}{\partial v_g}\Big((1+\sum_{j=1,j\neq g}^{S} R^{(ij)}v_j^2 + R^{(ig)}v_g^2)^{1-\eta}-1\Big) \\
&= -\frac{1}{N}\frac{1}{(1-\eta)}\sum_{i=1}^{N} -2R^{(ig)}v_g(\eta-1)(1+\sum_{j=1,j\neq g}^{S} R^{(ij)}v_j^2 + R^{(ig)}v_g^2)^{-\eta} \\
&= 2v_g\frac{1}{N}\frac{(\eta-1)}{(1-\eta)}\sum_{i=1}^{N}\Big[R^{(ig)}(1+\sum_{j=1}^{S} R^{(ij)}v_j^2)^{-\eta}\Big] \\
&= -2v_g\frac{1}{N}\sum_{i=1}^{N}\Big[R^{(ig)}(1+\sum_{j=1}^{S} R^{(ij)}v_j^2)^{-\eta}\Big]
\end{aligned}
\tag{69}
$$

$$
\frac{\partial g_1}{\partial v_g} = \frac{\partial}{\partial v_g}\Big(\sum_{j=1}^{S}(v_j^2)-1\Big) = 2v_g
$$

C Stan files

Listing 5: Stan file for the Bayesian Multivariate Gaussian model with the Inverse Wishart prior for Σ, normal-inverse-wishart.stan

```
data {
  int S; int N;
  vector[S] y[N];
}
parameters {
  cov_matrix[S] Sigma;
  vector[S] mus;
}
model {
  y ~ multi_normal(mus, Sigma);
  mus ~ normal(0, 1);
  Sigma ~ inv_wishart(S+1, diag_matrix(rep_vector(1.0, S)));
}
```

Listing 6: Stan file for the Bayesian Multivariate Gaussian model using the separation strategy and the LKJ prior for ρ, normal-lkj-cholesky.stan

```
data {
  int S; int N;
  vector[S] y[N];
  real shape;
}
parameters {
  vector<lower=0>[S] sigmas;
  corr_matrix[S] Omega;
  vector[S] mus;
}
transformed parameters {
  cov_matrix[S] Sigma;
  Sigma = quad_form_diag(Omega, sigmas); # = diag(sigmas) * Omega * diag(
      ↪ sigmas)
}
model {
  y ~ multi_normal(mus, Sigma);
  sigmas ~ cauchy(0, 1);
  mus ~ normal(0, 1);
  Omega ~ lkj_corr(shape);
}
```

Listing 7: Stan file for the Bayesian Hierarchical Multivariate Gaussian model with the Inverse Wishart prior for Σ, normal-inverse-wishart.stan

```
data {
  int S; int N;
  vector[S] y[N];
}
parameters {
  real mu_hyper;
```

```
  real sigma_sq_hyper;
  vector[S] mus;
  cov_matrix[S] Sigma;
}
transformed parameters {
  vector[S] vars;
  vars = diagonal(Sigma);
}
model {
  Sigma ~ inv_wishart(S+1, diag_matrix(rep_vector(1.0, S))/sigma_sq_hyper
      ↪ );
  y ~ multi_normal(mus, Sigma);
  mu_hyper ~ normal(0, 1);
  sigma_sq_hyper ~ inv_gamma(0.0001, 0.0001);

  for(j in 1:S)
    mus[j] ~ normal(mu_hyper, sqrt(vars[j]) / (N * 0.1));
}
```

Listing 8: Stan file for the Bayesian Hierarchical Multivariate Gaussian model using the separation strategy and the LKJ prior for ρ, normal-lkj.stan

```
data {
  int S; int N;
  vector[S] y[N];
  real shape;
}
parameters {
  real mu_hyper;
  vector[S] mus;
  cholesky_factor_corr[S] L_Omega;
  vector<lower=0>[S] L_sigma;
}
model {
  matrix[S, S] L_Sigma;
  L_Sigma = diag_pre_multiply(L_sigma, L_Omega);

  L_Omega ~ lkj_corr_cholesky(shape);
  L_sigma ~ cauchy(0, 1);
  y ~ multi_normal_cholesky(mus, L_Sigma);
  mu_hyper ~ normal(0, 1);

  for(j in 1:S)
    mus[j] ~ normal(mu_hyper, L_sigma[j] / (N * 0.1));
}
generated quantities {
  cov_matrix[S] Sigma;
  Sigma = quad_form_diag(multiply_lower_tri_self_transpose(L_Omega), L_
      ↪ sigma); // = diag(sigmas) * Omega * diag(sigmas)
}
```

D Listings

Listing 9: Dataset preprocessing, datastream_extract.R

```
library(data.table)
library(magrittr)
library(xts)
library(quantmod)

files = c('dcSPCOMP_Benchmark_RIAbsSimpR_W_USD.Rraw', 'dcSPCOMP_FirmInfo.Rraw', 'dcSPCOMP_IsinIX_W.Rraw', '
    ↪ dcSPCOMP_MV_W_USD.Rraw', 'dcSPCOMP_RIAbsSimpR_W_USD.Rraw', 'dcSPCOMP_Rf_SimpR_W_USD.Rraw')
for(i in files) load(paste0("data/", i))

rfrate = xts(dcSPCOMP_Rf_SimpR_W_USD$data, as.Date(dcSPCOMP_Rf_SimpR_W_USD$date)) / 100
rfrate = cumprod(1 + na.omit(rfrate))

summary(dcSPCOMP_FirmInfo)

firminfo = data.table(t(dcSPCOMP_FirmInfo$data))
setkey(firminfo, DSCD)
firminfo["130062",]

dates = dcSPCOMP_MV_W_USD$date
marketvalue = dcSPCOMP_MV_W_USD$data[,firminfo$DSCD]
insp500 = dcSPCOMP_IsinIX_W$data[,firminfo$DSCD]
tr = dcSPCOMP_RIAbsSimpR_W_USD$data[,firminfo$DSCD]/100
tr.sp500 = dcSPCOMP_Benchmark_RIAbsSimpR_W_USD$data/100

insp500.na = insp500; insp500[insp500 == 0] <- NA
marketvalue.sp500 = marketvalue * insp500.na[,colnames(marketvalue)]

weights = marketvalue.sp500[c(1, 1:(nrow(marketvalue.sp500)-1)),] # lag market values
weights[1,] = NA
weights[is.na(weights)] = 0
rownames(weights) = rownames(marketvalue.sp500)
all.na = which(apply(weights, 1, function(x) all(x == 0)))
weights = weights[-all.na[-length(all.na)],] # remove rows with all entries == 0
weights[length(all.na)] = 0

industries = as.character(na.omit(unique(firminfo$ICBIN)))
industries

tr.industries = matrix(data = NA, nrow=nrow(weights), ncol=length(industries) + 1)
colnames(tr.industries) <- c("Market", industries)
rownames(tr.industries) <- rownames(weights)

calc.p = function(filter) sapply(rownames(weights), function(r) sum(tr[r,filter] * weights[r,filter]/sum(
    ↪ weights[r,filter]), na.rm = T)) %>% xts(as.Date(rownames(weights)))
tr.industries[,"Market"] = calc.p(firminfo$DSCD)
for(industry in industries) {
  tr.industries[,industry] = calc.p(firminfo[ICBIN==industry]$DSCD)
}
tr.industries.xts = xts(tr.industries, order.by = as.Date(rownames(tr.industries)))
sp500.industries = cumprod(1 + tr.industries.xts) * 100 # reconstruct index

library(dygraphs)

dygraph(sp500.industries[,"Market"])
dygraph(sp500.industries)

marketdata.weekly = merge(sp500.industries, rfrate, all=F)
names(marketdata.weekly) <- c(toupper(make.names(names(sp500.industries))), "RF")

marketdata = apply.monthly(marketdata.weekly, function(x) tail(x, 1))
dygraph(cbind(marketdata[,"MARKET"], marketdata.weekly[,"MARKET"])) # compare
dygraph(cbind(marketdata[,"INDUSTRIALS"], marketdata.weekly[,"INDUSTRIALS"])) # compare

save(marketdata, file = "data/marketdata.Rdata")

###

# compare assembled index with actual index
comp = na.omit(merge(xts(tr.sp500, dates), ROC(sp500.industries[,"Market"], type='discrete'), all = F));
    ↪ names(comp) <- c("Benchmark", "Constructed")
dygraph(cumprod(1 + comp))
c(tail(cumprod(1 + comp[,1]), 1), tail(cumprod(1 + comp[,2]), 1))


xlab = format(as.Date(rownames(.md)), "%Y")
xlab.thin.b = c(F, xlab[-1] > xlab[-length(xlab)])
xlab.thin.x = which(xlab.thin.b)
xlab.thin.labs = xlab[xlab.thin.b]

saveplot(name = "indicies-discrepancy", f = function() {
  par(mfrow=c(1, 1),
```

```
      oma = c(0.2,0,0,0.5),
      mar = c(4,4,2,0))
  plot.default(x=index(comp), comp[,1] - comp[,2], ylab='discrepancy', main="Discrepancies", type='h', xlab='
      ↪ t')
  axis(1, at=xlab.thin.x, labels=xlab.thin.labs)
}, h = 4, w=12)

saveplot(name = "indicies", f = function() {
  par(mfrow=c(1, 1),
      oma = c(0.2,0,0,0.5),
      mar = c(2,4,2,0))
  #plot.xts(x=marketdata[,setdiff(symbols, c('RF', 'MARKET'))]), col=palette, ylab='Price', main="S&P 500
      ↪ industries", minor.ticks=NULL, major.ticks = NULL, bg="white", grid.ticks.on='years')

  .md = as.data.frame(marketdata[,setdiff(symbols, c('RF', 'MARKET'))])

  plot.default(x=index(.md), y=.md[,1],  type='n', ylim=c(min(.md), max(.md)), col=palette, xlab=NULL, ylab='
      ↪ Price', main="S&P␣500␣industries", xaxt='n')
  axis(1, at=xlab.thin.x, labels=xlab.thin.labs)
  abline(v=xlab.thin.x,  lty=2, col="lightgrey")

  for (i in 1:ncol(.md)){
    lines(x=index(.md), y=.md[,i], col=palette[i], type='l', lwd=2)
  }

  legend("topleft",
           industries,
           cex=1, fill=palette, bg = 'white')

}, h = 8, w=12) # todo: plot does not work

paste(min(index(marketdata)), "-", max(index(marketdata)))
nrow(marketdata)

industries.names = setdiff(symbols, c('RF', 'MARKET'))
marketdata.returns = ROC(marketdata[,industries.names], type = 'discrete')[-1,]
summ = data.frame("Yearly␣mean␣return" = apply(marketdata.returns, 2, mean)*12*100,
          "Yearly␣volatility" = apply(marketdata.returns, 2, sd)*sqrt(12)*100,
          "Total␣return/%" = (as.vector(marketdata[nrow(marketdata),industries.names])/as.vector(marketdata
              ↪ [1,industries.names]) - 1) * 100,
          row.names = industries)
print(xtable(summ, digits = 1), include.rownames=T)
```

Listing 10: Backtest core, backtest.R

```
library(data.table)
library(pryr)
library(quantmod)

diffyear <- function(t1, t2) {
  as.numeric(difftime(t1, t2, units="secs")) / (60 * 60 * 24 * 365)
}

contract.id <- function(order) toupper(order$symbol)
value <- function(c, spot, date) c$amount * spot

drawdowns <- function(xs) {
  dd = rep(0, length(xs))
  max = xs[0]
  for(i in 1:length(xs)) {
    max = max(xs[i], max)
    dd[i] = min(0, xs[i]/max-1)
  }
  dd
}

first.nonzero.idx = function(xs) Reduce(function(i, x) if(i>0) i else ifelse(x == 0, i-1, -i), xs, -1)

backtest <- function(xs, f.backtest, wealth, log.enabled=F, bankruptcy.stop=F, plot.every=F, data=list()) {
  name = nameof(f.backtest)
  n = nrow(xs)
  net.liq = wealth
  pl = rep(0, nrow(xs))
  positions = list()
  allocations = data.table()
  pl.total = 0
  bankruptcy = F
  ret = ROC(xs, na.pad = F)
  rs = NULL

  print(paste0("Starting␣backtest␣n=", n, ":␣", name))

  for(i in 1:n) {
    for (pn in names(positions)) {
      position = positions[[pn]]

      sym = position$symbol
```

```
  spot = as.numeric(xs[i, sym])
  pl.current = value(position, as.numeric(xs[i, sym])) - value(position, as.numeric(xs[i-1, sym]))

  pl[i] <- pl[i] + pl.current
}

pl.total = pl.total + pl[i]
net.liq = net.liq + pl[i]

if(net.liq <= 0) {
  bankruptcy = T
  if(bankruptcy.stop) {
    stop('Bankruptcy')
  }
}

spx = as.vector(xs[i]); names(spx) <- names(xs)
date = index(xs[i])
ctx = list(
  date = date,
  spot = function(s) spx[s],
  history.df = xs[1:i,],
  history = function(s) xs[1:i,s],
  wealth.initial = wealth,
  idx = i,
  ret = function(s) ret[index(ret) <= date, s],
  ret.df = ret[index(ret) <= date,],
  positions = positions,
  pl.total = pl.total,
  net.liq = net.liq,
  data = data,
  record = function(df) {
    if(is.null(rs)) {
      rs <<- as.data.frame(matrix(rep(NA, n*ncol(df)), nrow=n))
      colnames(rs) <<- colnames(df)
      rownames(rs) <<- index(xs)
    }
    rs[i,] <<- df
  },
  position = function(s) positions[[s]],
  position.amount = function(s) {
    p = positions[[s]]
    if(is.null(p)) {
      0
    } else {
      p$amount
    }
  }
)
ctx$order.amount = function(s, target=NA, target_weight=NA) {
  if(!is.na(target)) {
    target - ctx$position.amount(s)
  } else if(!is.na(target_weight)) {
    floor(ctx$net.liq / ctx$spot(s) * target_weight) - ctx$position.amount(s)
  }
}

orders = f.backtest(ctx)

for(on in index(orders)) {
  order = orders[[on]]
  if(order$amount == 0) {
    next
  }

  id = contract.id(order)
  o = positions[[id]]

  if(!is.null(o)) {
    o$amount = o$amount + order$amount
    positions[[id]] <- o
  } else {
    order$symbol = toupper(order$symbol)
    positions[[id]] <- order
  }

  if(log.enabled) {
    print(sprintf("%s: Order %f %s (Total: %f)", index(xs[i]), order$amount, id, positions[[id]]$amount))
  }
}

allocation <- list(.d=1)
for (pn in names(positions)) {
  position = positions[[pn]]
  if(position$type == "s") {
    allocation[[position$sym]] = position$amount * as.numeric(xs[i, position$sym]) / net.liq
  }
}
allocations <- rbindlist(list(allocations, do.call(data.frame, allocation)), fill = T, use.names = T)
```

```
    if(is.numeric(plot.every) && i%%plot.every == 0) {
      plot( ((wealth + cumsum(pl[1:i]))/wealth - 1)*100, type='l')
    }
  }

  net.liqs = wealth + cumsum(pl)
  acct.ret = c(0, net.liqs[-1] / net.liqs[-n] - 1)

  eff.start = max(first.nonzero.idx(acct.ret) - 1, 1)
  eff.idx = eff.start:n

  effective.ret = acct.ret[eff.idx]

  years = diffyear(index(xs[n]), index(xs[eff.start]))
  cagr = (cumprod(1 + effective.ret) %>% tail(1))^(1/years) - 1

  allocations[,".d"] = NULL
  allocations[is.na(allocations)] = 0

  to.x = partial(xts, order.by=index(xs))
  list(name=name, xs = xs, effective.date=c(index(xs[eff.start]), index(xs[n])), effective.idxs = eff.idx,
       ret.total=net.liq/wealth-1, ret=acct.ret %>% to.x, ret.sd = sd(effective.ret), ret.mean = mean(
            ↪ effective.ret),
       ret.cagr=cagr,
       positions = positions, pl=pl %>% to.x, bankruptcy=bankruptcy,
       years=years, net.liq=net.liqs %>% to.x, drawdowns = drawdowns(net.liqs) %>% to.x,
       records=rs %>% to.x, allocations=allocations %>% to.x)
}
```

Listing 11: Backtest functions, backtest_functions.R

```
f.opt.mean_var = function(target.sd) function(mus, cov) {
  w.opt = (solve(cov) %*% mus) %>% as.vector
  sd.p = sqrt(t(w.opt) %*% cov %*% w.opt)
  factor = (target.sd/sd.p) %>% as.numeric
  w.opt = w.opt * factor
}

f.opt.quadprod = function(risk.aver = 3) {
  named(paste0("quadprog␣riskaver=", risk.aver), function(mus, cov) {
    w = solve.QP(Dmat = cov * risk.aver/2, dvec = mus, Amat = cbind(1, diag(nrow(cov))), bvec = c(1, rep(0,
         ↪ nrow(cov))), meq = 1)$solution # sum(w) = 1, no short sells
    w[abs(w) <= 1e-7] = 0
    w
  })
}

model.mcov.mle = function(lb = 12*2)
  named(paste0("meanvar␣MLE,␣lb=", lb), function(ctx, symbols) {
    if(nrow(ctx$ret.df) < lb) return(NA)
    R = tail(ctx$ret.df[,symbols], lb)
    list(
      mus=apply(R, 2, mean),
      cov=cov(R),
      rhat=c(1), # fake rhat
      yp=c(0) # fake yp
    )
  })


# f.pred = function(ctx, symbols) : returns list(mus: Expected returns, cov: Expected covariances)
# f.opt = function(mus, cov) : returns w.opt
f.bt.generic = function(symbols, f.pred, f.optimize, rhat.threshold = 0.1)
named(paste0("(model=", nameof(f.pred), ",␣optimization=", nameof(f.optimize), ")"),
  function(ctx) {
    while(T) {
      p = NULL
      try({p = f.pred(ctx, symbols)})

      if(!is.null(p) && is.na(p)) { # no error - skip prediction
        return(lapply(symbols, function(s) list(type="s", symbol=s, amount=ctx$order.amount(s, target_weight
             ↪ =0))))
      }
      if(is.null(p) || any(is.na(p$yp))) {
        print(paste("BT-WARN:␣Error␣in␣prediction,␣skipping", ctx$date))
        return(lapply(symbols, function(s) list(type="s", symbol=s, amount=ctx$order.amount(s, target_weight
             ↪ =0))))
      }

      rhat = na.omit(p$rhat)
      # convergence check
      if(!is.na(min(rhat)) && length(min(rhat)) == 1 && is.numeric(min(rhat)) && min(rhat) > (1 - rhat.
           ↪ threshold) && max(rhat) < (1 + rhat.threshold))
        break;
      print(sprintf("BT-WARN:␣DIVERGENCE␣DETECTED:␣%s,␣retrying:␣rhat.min=%f,␣rhat.max=%f,␣rhat=%s", ctx$date
           ↪ , min(p$rhat), max(p$rhat), p$rhat))
```

```
    }

    cov = p$cov
    mus = p$mus
    cor = cov2cor(cov)
    sds = diag(cov) %>% sqrt

    w.opt = NULL
    try({w.opt = f.optimize(mus, cov)})
    if(is.null(w.opt) || any(is.na(w.opt))) {
      print(paste("BT-WARN: Error in optimization, skipping", ctx$date))
      return(lapply(symbols, function(s) list(type="s", symbol=s, amount=ctx$order.amount(s, target_weight=0)
          ↪ )))
    }
    names(w.opt) = symbols

    mu.p = (t(w.opt) %*% mus)
    sd.p = sqrt(t(w.opt) %*% cov %*% w.opt)
    sharpe.p = mu.p/sd.p

    point.forecasts = mus

    ctx$record(
      cbind(
        data.frame(
          sharpe = sharpe.p * sqrt(250),
          mu = mu.p * 250,
          sd = sd.p * sqrt(250)
          #cor = cor[2,1]
        ),
        (sds * sqrt(250)) %>% t %>% `colnames<-`(sapply(symbols, function(s) paste0(s, ".sd"))) %>% as.data.
            ↪ frame,
        (mus * 250) %>% t %>% `colnames<-`(sapply(symbols, function(s) paste0(s, ".mu"))) %>% as.data.frame,
        w.opt %>% t %>% `colnames<-`(sapply(symbols, function(s) paste0(s, ".w"))) %>% as.data.frame,
        point.forecasts %>% t %>% `colnames<-`(sapply(symbols, function(s) paste0("point.forecast.", s))) %>%
            ↪  as.data.frame
      )
    )

    lapply(symbols, function(w) list(type="s", symbol=w, amount=ctx$order.amount(w, target_weight=w.opt[w])))
  })


# f.pred = function(ctx, symbols) : returns list()... TODO
# f.optimize = function(mus, cov) : returns w.opt
f.bt.generic.yp = function(symbols, f.pred, f.optimize, rhat.threshold = 0.1) {
  named(paste0("(model=", nameof(f.pred), ", optimization=", nameof(f.optimize), ")"), function(ctx) {
    while(T) {
      p = NULL
      try({p = f.pred(ctx, symbols)})

      if(!is.null(p) && is.na(p)) { # no error - skip prediction
        return(lapply(symbols, function(s) list(type="s", symbol=s, amount=ctx$order.amount(s, target_weight
            ↪ =0))))
      }
      if(is.null(p) || any(is.na(p$yp))) {
        print(paste("BT-WARN: Error in prediction, skipping", ctx$date))
        return(lapply(symbols, function(s) list(type="s", symbol=s, amount=ctx$order.amount(s, target_weight
            ↪ =0))))
      }

      rhat = na.omit(p$rhat)
      # convergence check
      if(!is.na(min(rhat)) && length(min(rhat)) == 1 && is.numeric(min(rhat)) && min(rhat) > (1 - rhat.
          ↪ threshold) && max(rhat) < (1 + rhat.threshold))
        break;
      print(sprintf("BT-WARN: DIVERGENCE DETECTED: %s, retrying: rhat.min=%f, rhat.max=%f, rhat=%s", ctx$date
          ↪ , min(p$rhat), max(p$rhat), p$rhat))
    }

    yp = p$yp

    opt = NULL
    try({opt = f.optimize(yp)})
    if(is.null(opt) || any(is.na(opt$w)) || sum(opt$w) > 1e3 || sum(opt$w) < 0 ) {
      print(paste("BT-WARN: Error in optimization, skipping", ctx$date))
      return(lapply(symbols, function(s) list(type="s", symbol=s, amount=ctx$order.amount(s, target_weight=0)
          ↪ )))
    }

    w.opt = opt$w
    names(w.opt) = symbols

    mus = apply(yp, 2, mean)
    cov = cov(yp)
    mu.p = (t(w.opt) %*% mus)
    sd.p = sqrt(t(w.opt) %*% cov %*% w.opt)

    ctx$record(
```

```
      cbind(
        data.frame(
          mu = mu.p,
          sd = sd.p,

          rhat.min=min(p$rhat),
          rhat.max=max(p$rhat),
          opt.util=opt$util
        ),
        w.opt %>% t %>% `colnames<-`(sapply(symbols, function(s) paste0(s, ".w"))) %>% as.data.frame,
        p$point.forecasts %>% t %>% `colnames<-`(sapply(symbols, function(s) paste0("point.forecast.", s)))
            ↪ %>% as.data.frame
      )
    )

    lapply(symbols, function(w) list(type="s", symbol=w, amount=ctx$order.amount(w, target_weight=w.opt[w])))
  })
}
```

Listing 12: Backtest main, main.R

```
# install required packages: install.packages(c('data.table', 'pryr', 'quantmod', 'magrittr', 'xts', 'mvtnorm
    ↪ ', 'nloptr', 'xtable', 'rstan', 'quadprog', 'sn'))

cores = 8
chains = 2 # number of MCMC chains
iter = 2000 # number of samples per chain
myseed = 123 # seed for RNG

source('~/Dropbox/thesis/r/init.R')
source('backtest.R')
source('backtest_functions.R')
source('direct_expected_utility.R')
source('models.R')

f.sample = function(model, data, ...) sampling(model, data=data, chains=chains, warmup=1000, iter=iter,
    ↪ verbose=F, cores=cores, open_progress=F, show_messages=F, refresh=-1, seed=myseed, save_warmup=F,
    ↪ ...)
f.extract = function(fit, name) rstan::extract(fit, name)[[name]]
f.extract_rhat = function(fit) rstan::stan_rhat(fit)$data$stat
bt.filename = function(data, m) paste0("backtests/", index(data)[1], "␣-␣", index(data)[length(index(data))],
    ↪  "␣-␣", nameof(m), ".backtest.Rdata")

get_models <- function(symbols,
                       lookback = 12 * 5,
                       mv.riskaversion = 3,
                       crra.riskaversion = 3) {
  list(
    f.bt.generic(symbols, model.mcov.mle(lb = lookback), f.opt.quadprod(risk.aver=mv.riskaversion)),

    # Inverse Wishart prior
    f.bt.generic.yp(symbols, model.normal(lb=lookback, prior="inverse␣wishart", hierarchial=F), f.opt.sqp(
        ↪ riskaver = crra.riskaversion)),
    f.bt.generic(symbols, model.normal(lb=lookback, prior="inverse␣wishart", hierarchial=F), f.opt.quadprod(
        ↪ risk.aver=mv.riskaversion)),

    f.bt.generic.yp(symbols, model.normal(lb=lookback, prior="inverse␣wishart", hierarchial=T), f.opt.sqp(
        ↪ riskaver = crra.riskaversion)),
    f.bt.generic(symbols, model.normal(lb=lookback, prior="inverse␣wishart", hierarchial=T), f.opt.quadprod(
        ↪ risk.aver=mv.riskaversion)),

    # LKJ correlation prior
    f.bt.generic.yp(symbols, model.normal(lb=lookback, prior="lkj", shape=1, hierarchial=F), f.opt.sqp(
        ↪ riskaver = crra.riskaversion)),
    f.bt.generic(symbols, model.normal(lb=lookback, prior="lkj", shape=1, hierarchial=F), f.opt.quadprod(risk
        ↪ .aver=mv.riskaversion)),

    f.bt.generic.yp(symbols, model.normal(lb=lookback, prior="lkj", shape=1, hierarchial=T), f.opt.sqp(
        ↪ riskaver = crra.riskaversion)),
    f.bt.generic(symbols, model.normal(lb=lookback, prior="lkj", shape=1, hierarchial=T), f.opt.quadprod(risk
        ↪ .aver=mv.riskaversion))
  )
}

run_all <- function() {
  load('data/marketdata.Rdata'); symbols = names(marketdata); symbols = setdiff(symbols, c('RF', 'MARKET'));
      ↪ symbols
  models = get_models(symbols); data = marketdata
  sapply(1:length(models), function(i) run_model(models[[i]], data))
}

run_model <- function(model, data) {
  tryCatch({
    file = bt.filename(data, model)

    if(file.exists(file)) {
      stop(paste0("BTPAR:␣SKIP:␣skipping␣because␣exists:␣", file))
```

```
    }

    sink(file = paste0(file, '.log'), append = FALSE, type = c("output", "message"), split = T)

    print(paste0('BTPAR: START ', file))
    r = backtest(xs = data, f.backtest = model, wealth=10000000, plot.every=10, log.enabled=T)
    save("r", file=paste0("backtests/", index(r$xs)[1], " - ", index(r$xs)[length(index(r$xs))], " - ", r$
        ↪ name, ".backtest.Rdata"))

    print(paste0('BTPAR: DONE ', file))
  }, error=function(e) print(paste0('BTPAR: ERROR ', e)), finally = sink())
}

crra.util.r = function(r, ra) ((1+r)^(1-ra) - 1)/(1-ra)
crra.util.ce = function(u, ra) (((1 - ra)* (u + 1/(1 - ra)))^(1/(1 - ra)) - 1)

squared.util.r = function(r, ra) mean(r) - var(r)*ra/2
squared.util.ce = function(u, ra) u

contains = function(ss, part) sapply(ss, function(s) length(grep(part, s)) > 0)
get.model_name = function(rr) {
  s = rr$name
  if(contains(s, "model=meanvar MLE")) r = ("MLE")
  else if(contains(s, "model=Normal Inverse Wishart")) if(contains(s, "hierarchical=TRUE")) r = ("Hi.Gaus. IV
      ↪ ") else r = ("Gaus. IV") # todo: remove
  else if(contains(s, "model=Normal LKJ")) if(contains(s, "hierarchical=TRUE")) r = ("Hi.Gaus. LKJ") else r =
      ↪  ("Gaus. LKJ")
  else {
    warning("Unknown model")
    return("Unknown")
  }
  return(r)
}
get.optimization_name = function(rr) {
  s = rr$name
  if(contains(s, "optimization=quadprog")) r = "MV"
  else if(contains(s, "optimization=CRRA")) r = paste0("CRRA")
  else {
    warning("Unknown optimization type")
    return("Unknown")
  }
  return(r)
}
get.full_name = function(r) paste(get.model_name(r), get.optimization_name(r))

get.squared_error = function(r) {
  sqerr = c()
  for(s in symbols) {
    x = merge(
      ROC(marketdata[,s], type='discrete'),
      r$records[,paste0('point.forecast.', s)]
    )
    x[,2] = lag(x[,2])
    x = na.omit(x)
    sqerr = c(sqerr, as.vector((x[,1] - x[,2])^2))
  }
  return(sqerr)
}
map.bts = function(bts, f) {
  names = c()
  df = as.data.frame(t(sapply(bts, function(b) {
    load(b)
    r$ret.eff = r$ret[r$effective.idxs][-1]
    names <<- c(names, get.full_name(r))
    f(r)
  })))
  rownames(df) = names
  df
}
map.bts.xts = function(bts, f) {
  names = c()
  df = as.data.frame(t(sapply(bts, function(b) {
    load(b)
    r$ret.eff = r$ret[r$effective.idxs][-1]
    names <<- c(names, get.full_name(r))
    f(r)
  })))

  load(bts[1])
  r$ret.eff = r$ret[r$effective.idxs][-1]
  index = index(f(r))

  rownames(df) = names
  xts(t(df), order.by=index)
}


.main.interactive = function() {
  load('data/marketdata.Rdata'); symbols = names(marketdata); symbols = setdiff(symbols, c('RF', 'MARKET'))
```

```
rf.returns = ROC(marketdata[,"RF"], type='discrete')

bts = paste0('backtests/', dir('backtests', pattern='*.Rdata$'))
bts = bts[c(1,3,7, 5,9, 2,6, 4,8)] # reorder
load(bts[8])

benchmark.mse = map.bts(bts, function(r) c(mse = mean(get.squared_error(r)), x=1))[1,1]

# ret must be taken with effective idxs

saveplot(name = "portfolio-returns", f = function() {
  par(mfrow=c(1, 1),
      oma = c(0.2,0,0,0.5),
      mar = c(2,4,2,0))

  df = map.bts.xts(bts, function(r) r$ret.eff)
  df.prices = as.data.frame(cumprod(1 + df))

  xlab = format(as.Date(rownames(df.prices)), "%Y")
  xlab.thin.b = c(F, xlab[-1] > xlab[-length(xlab)])
  xlab.thin.x = which(xlab.thin.b)
  xlab.thin.labs = xlab[xlab.thin.b]

  plot.default(x=index(df.prices), y=df.prices[,1],  type='n', ylim=c(min(df.prices), max(df.prices)), col=
      ↪ palette, xlab=NULL, ylab='Cumulative␣return', main="Realized␣cumulative␣returns␣of␣the␣
      ↪ constructed␣portfolios", xaxt='n')
  axis(1, at=xlab.thin.x, labels=xlab.thin.labs)
  abline(v=xlab.thin.x,  lty=2, col="lightgrey")

  for (i in 1:ncol(df.prices)){
    lines(x=index(df.prices), y=df.prices[,i], col=palette[i], type='l', lwd=1)
  }

  legend("topleft",
         colnames(df.prices),
         cex=1, fill=palette, bg = 'white')

}, h = 8, w=12)

## summary
df = map.bts(bts, function(r) {
  ret.excess = na.omit(r$ret.eff - rf.returns)
  c(
    "Mean/%"=mean(r$ret.eff)*12*100,
    "SD/%"=sd(r$ret.eff)*sqrt(12)*100,
    "Mean␣Excess/%"=mean(ret.excess)*12*100,
    "Sharpe␣ratio"=mean(ret.excess)/sd(r$ret.eff)*sqrt(12)
  )
})
df
print(xtable(df, digits = 2), include.rownames=F)

## statistical analysis
df = map.bts(bts, function(r) {
  sqerr = get.squared_error(r)
  rsq = 1-mean(sqerr)/benchmark.mse
  c(
    MSE=mean(sqerr),
    RMSE=sqrt(mean(sqerr)),
    "R^2/%"=rsq*100
  )
})
df = df[-which(contains(rownames(df), "CRRA")),]; rownames(df) = sub("␣MV","", rownames(df))
df
print(xtable(df, digits = 6), include.rownames=F)

## CE
df = map.bts(bts, function(r) {
  u.sq = squared.util.r(r$ret.eff, 3)
  ce.sq = sapply(mean(u.sq), function(u) squared.util.ce(u, 3))

  u.crra = crra.util.r(r$ret.eff, 3)
  ce.crra = sapply(mean(u.crra), function(u) crra.util.ce(u, 3))
  c(
    "Mean␣Sq.Ut"=mean(u.sq),
    "CE␣Sq/%"=ce.sq*12*100,
    "CE␣Sq␣Gain/%"=0,

    "Mean␣CRRA␣Ut"=mean(u.crra),
    "CE␣CRRA/%"=ce.crra*12*100,
    "CE␣CRRA␣Gain/%"=0
  )
})
#df = df[-which(contains(rownames(df), "MV")),]; rownames(df) = sub(" CRRA","", rownames(df))
df[,"CE␣CRRA␣Gain/%"] = (df[,"CE␣CRRA/%"] - df[1,"CE␣CRRA/%"])
df[,"CE␣Sq␣Gain/%"] = (df[,"CE␣Sq/%"] - df[1,"CE␣Sq/%"])
df
print(xtable(df, digits = 4), include.rownames=F)
```

```
  # 4 factor model
  ffactors = read.csv('data/F-F_Research_Data_Factors.CSV')

  ffactors.mom = read.csv('data/F-F_Momentum_Factor.CSV')
  ffactors = xts(ffactors, order.by=as.Date(paste0(as.character(ffactors$Date), "01"), format='%Y%m%d'))
  ffactors.mom = xts(ffactors.mom, order.by=as.Date(paste0(as.character(ffactors.mom$Date), "01"), format='%Y
      ↪ %m%d'))
  ffactors = merge(ffactors, ffactors.mom)

  df = map.bts(bts, function(r) {
    merged = xts(r$ret, order.by = as.Date(paste0(gsub('.{2}$', '', as.character(index(marketdata))), "01")))
        ↪ [r$effective.idxs][-1]
    merged = merge(merged, ffactors, all=F)
    m = lm(x - RF/100 ~ Mkt.RF + SMB + HML + Mom, data=merged)
    m.summ = summary(m)

    x = paste(sig.level(m.summ$coefficients[,"Pr(>|t|)"]), paste0(sprintf("%.1f", m.summ$coefficients[,"
        ↪ Estimate"]*12*100), "␣(", sprintf("%.1f", m.summ$coefficients[,"Std.␣Error"]*12*100), ")"))
    names(x) = rownames(m.summ$coefficients)
    x
  })
  df
  print(xtable(df, digits = 4), include.rownames=F)
  #

  df = map.bts(bts, function(r) {
    merged = xts(r$ret, order.by = as.Date(paste0(gsub('.{2}$', '', as.character(index(marketdata))), "01")))
        ↪ [r$effective.idxs][-1]
    merged = merge(merged, ffactors, all=F)
    m = lm(x ~ Mkt.RF + SMB + HML + RMW + CMA, data=merged)
    m.summ = summary(m)

    m.summ$coefficients
  })
  df

}

####


.main <- function() {
    load('data/marketdata.Rdata')
    symbols = names(marketdata)
    symbols = setdiff(symbols, c('RF', 'MARKET'))
    benchmark = ROC(marketdata[,'MARKET'], type = 'discrete', na.pad = F)

    args <- commandArgs(trailingOnly = TRUE)
    if(length(args) > 0) {
      models = get_models(symbols); data = marketdata
      is = as.numeric(args[1]):as.numeric(args[2])
      print(paste('Running␣models␣', as.character(is)))
      sapply(as.numeric(args[1]):as.numeric(args[2]), function(i) run_model(models[[i]], data))
    }
}

.main()
```

Listing 13: Backtest models, models.R

```
library(mvtnorm)

normal_lkj.stan = stan_model(file='models/normal-lkj-cholesky.stan')
normal_lkj.hierarchial.stan = stan_model(file='models/normal-lkj-hierarchial.stan')

normal_inv_wishart.stan = stan_model(file='models/normal-inv-wishart.stan')
normal_inv_wishart.hierarchial.stan = stan_model(file='models/normal-inv-wishart-hierarchical.stan')

model.normal = function(lb = 12*2, prior, hierarchial, shape=1) {
  if(prior == "lkj") {
    name = paste0("Normal␣LKJ(lb=", lb, ",␣hierarchical=", hierarchial, ",␣shape=",  shape,")")
    model = if(hierarchial) normal_lkj.hierarchial.stan else normal_lkj.stan
    run = function(R) f.sample(model, data=list(y=R, N=nrow(R), S=ncol(R), shape=shape))
  }
  else if(prior == "inverse␣wishart") {
    name = paste0("Normal␣Inverse␣Wishart(lb=", lb, ",␣hierarchical=", hierarchial, ")")
    model = if(hierarchial) normal_inv_wishart.hierarchial.stan else normal_inv_wishart.stan
    run = function(R) f.sample(model, data=list(y=R, N=nrow(R), S=ncol(R)))
  }
  else stop("Unknown␣prior")

  named(name, function(ctx, symbols) {
    if(nrow(ctx$ret.df) < lb) return(NA)
    R = tail(ctx$ret.df[,symbols], lb)

    S = length(symbols)
    N = nrow(R)
```

```
    s = run(R)
    mus = f.extract(s, "mus")
    Sigma = f.extract(s, "Sigma")

    set.seed(myseed)
    N.sample = nrow(mus)*4
    yp = matrix(1, nrow=N.sample, ncol=S)
    for(i in 1:N.sample) {
      ii = ((i-1) %% nrow(mus)) + 1
      yp[i,] = rmvnorm(1, mean = mus[ii,], sigma=Sigma[ii,,])
    }
    point.forecasts = apply(mus, 2, mean)
    #browser()

    mus.pred = apply(mus, 2, mean)
    cov.pred = apply(Sigma, c(2,3), mean) + cov(mus)

    list(yp=yp, rhat=f.extract_rhat(s), point.forecasts=point.forecasts, mus=mus.pred, cov=cov.pred)
  })
}
```

Listing 14: Utility optimization, direct_expected_utility.R

```
f.opt.crra <- function(riskaver = 0.5, tau=1.1, e = 2e-3, maxstep=100)
  named(paste0("CRRA Utility, AugLang (riskaversion=", riskaver, ", tau=", tau, ", e=", e, ", maxstep=",
       ↪ maxstep, ")"), function(R) {
    f.util = function(w, R) ((1 + R %*% w^2) ^ (1 - riskaver) - 1)/(1 - riskaver)

    f.g = function(w) sum(w^2) - 1

    f.l = function(lambda, r) function(w) {
      u = f.util(w, R)
      g = f.g(w)
      -sum(u)/length(u) + lambda * g + 0.5 * r * g^2
    }

    f.l.grad = function(lambda, r) function(w) {
      grad = rep(NA, length(w))
      for(i in 1:length(w)) {
        s = R[,i] * (1 + R %*% (w^2))^(-riskaver)
        grad[i] =  2 * w[i] * (-sum(s)/length(s) + lambda + r * f.g(w))
      }
      grad
    }

    .search.auglang(R, f=f.l, f.grad = f.l.grad, f.g = f.g, tau=tau, e=e, maxstep = maxstep)
  })

f.opt.sqp <- function(riskaver = 0.5)
  named(paste0("CRRA Utility, SQP (riskaversion=", riskaver, ")"), function(R) {
    library(nloptr)
    N = nrow(R)

    f.util = function(w, R) ((1 + R %*% w^2) ^ (1 - riskaver) - 1)/(1 - riskaver)

    f.target = function(w) 1/N * -sum(f.util(w, R))
    f.target.grad = function(w) {
      grad = rep(NA, length(w))
      for(i in 1:length(w)) {
        s = R[,i] * (1 + R %*% (w^2))^(-riskaver)
        grad[i] = 2 * w[i] * -sum(s)/length(s)
      }
      grad
    }

    f.g = function(w) sum(w^2) - 1
    f.g.grad = function(w) w*2

    w0 = rep(1/ncol(R), ncol(R))

    f.opt = function(ftol_rel, maxeval) slsqp(w0, fn = f.target, gr = f.target.grad,
                                              #hin = f.ine, hinjac = f.ine.grad,
                                              heq = f.g, heqjac = f.g.grad,
                                              nl.info=T, control=list(check_derivatives=F, maxeval=maxeval,
                                                   ↪ ftol_rel=ftol_rel))

    wopt = find.solution(f.opt, ftol_rel=1e-7, maxeval=1500)

    w=wopt$par^2
    w[w<1e-4] = 0
    list(w=w, util=sum(f.util(wopt$par, R)))
  })

# helper function for aug. lagrange
.search.auglang <- function(R, f, f.grad, f.g, tau=1.1, e = 2e-3, maxstep=100) {
```

```
  minimize <- function(R, lambda, r) {
    w.init = rep(1/ncol(R), ncol(R))
    gr = f.grad(lambda, r)
    o = optim(par = w.init, fn = f(lambda, r), gr = gr, control=list(trace=3, maxit=1000, fnscale=0.00001),
        ↪ method = 'L-BFGS-B')
    w.opt = o$par^2
    list(g = f.g(o$par), w=w.opt, grad=gr(o$par))
  }

  r = 10
  lambda = 1

  counter = 0
  for(i in 1:maxstep) {
    counter = counter + 1
    pars = minimize(R, lambda, r)
    lambda = lambda + r * pars$g
    r = tau * r
    if( sqrt(sum(pars$grad^2)) + abs(pars$g) < e ) break
  }

  if(counter == maxstep) warning(paste0("opt search didn't converge after steps=", counter))

  w = pars$w
  list(w=w, pars=pars, i=counter)
}

# helper function for SQP solution search
find.solution = function(f.opt, ftol_rel, maxeval) {
  while(T) {
    sol = f.opt(ftol_rel, maxeval)

    if(sol$value < -1000) sol$par = sol$par*0
    if(sol$convergence >= 1 && sol$convergence <= 4) return(sol)

    ftol_rel = ftol_rel * 2
    maxeval = maxeval * 1.5
    print(sprintf('BTPAR: Optimization did not converge: %s, retrying with ftol_rel=%f, maxeval=%f', sol$
        ↪ message, ftol_rel, maxeval))
  }
}


# test functions
function() {
  library(MASS)
  S = 10
  R = mvrnorm(10000, mu = rep(0.06/52, S), Sigma=diag((0.2/sqrt(52))^2, S))
  w = rep(1/S, S)

  sum(f.opt.sqp(riskaver = 1.5)(R)$w)
  sum(f.opt.sqp.nw(riskaver = 20)(R)$w)

  f.grad.num <- function(f, e = 0.00001) function(w) {
    grad = rep(NA, length(w))
    for(i in 1:length(w)) {
      x = rep(0, length(w)); x[i] = e
      grad[i] = (f(w + x) - f(w - x))/(2*e)
    }
    grad
  }
}
```

Listing 15: Initialization file, init.R

```
# source('~/Dropbox/thesis/r/init.R')
library(magrittr)
library(data.table)
library(xts)
library(xtable)
library(quantmod)
library(rstan)
library(quadprog)

par(cex=0.6)
set.seed(myseed)
rstan_options(auto_write = TRUE)

setwd('~/Dropbox/thesis/r')

palette <- c('chartreuse3', 'cornflowerblue', 'darkgoldenrod1', 'peachpuff3',
             'mediumorchid2', 'turquoise3', 'wheat4', 'slategray2', "darkgrey", "red")

dens <- function(x, adj=1, ...) {
  plot(density(x, adjust = adj), ...)
}
```

```
named <- function(name, o) {
  attr(o, "name") <- name
  o
}
nameof <- function(o) {
  x <- attr(o, "name")
  if(is.null(x)) warning(paste0(summary(o), "␣has␣no␣name"))
  x
}

multilineplot <- function(xs) {
  plot(xs[,1], ylim = c(min(apply(xs, 2, min)), max(apply(xs, 2, max))),type = "n")

  cl <- rainbow(ncol(xs))

  for (i in 1:ncol(xs)) {
    lines(xs[,i], col=cl[i])
  }
  legend("topleft", legend=names(xs), col=cl)
}

extract_pars <- function(pars, name) pars[grep(paste0(name, "\\["), names(pars))]

saveplot <- function(name, f, w=6, h=3, font.size = 0.8) {
  f()
  pdf(file=paste0('~/Dropbox/thesis/tex/img/', name, '.pdf'), width = w, height = h)
  par(cex=font.size)
  f()
  dev.off()

  cat(sprintf(paste0("\\begin{figure}[H]\n",
                     "\\includegraphics{img/%s.pdf}\n",
                     "\\caption{...}\n",
                     "\\label{fig:%s}\n",
                     "\\end{figure}\n"), name, name))
}

saveplot.tikz <- function(name, f, w=6, h=3) {
  f()

  require(tikzDevice)
  fntsize <- 0.8
  options(tikzDocumentDeclaration = c(
    "\\documentclass[a4paper]{article}",
    "\\usepackage{amssymb,amsmath,graphicx}",
    "\\usepackage{tikz}",
    "\\usepackage{cite}",
    "\\usepackage{algorithm}",
    "\\usepackage{algorithmic}",
    "\\usepackage{siunitx}",
    "\\sisetup{per=fraction,fraction=nice,alsoload=binary}",
    "\\usepackage{fixme}",
    "\\fxsetup{draft}",
    "\\fxuselayouts{index}"))

  tikz(file=paste0('~/Dropbox/thesis/tex/img/', name, '.tex'), width = w, height = h)
  par(cex=fntsize)
  f()
  dev.off()

  cat(sprintf(paste0("\\begin{figure}[H]\n",
        "\\input{img/%s.tex}\n",
        "\\caption{...}\n",
        "\\label{fig:%s}\n",
        "\\end{figure}\n"), name, name))
}
```

Listing 16: Metropolis Hastings algorithm and experiment, examples/metropolis.R

```
source('~/Dropbox/thesis/r/init.R')

metropolis = function(
  f.prior, # prior f(theta)
  f.likelihood, # likelihood f(x | theta)
  theta.init = 0,
  f.proposal = function(theta) rnorm(1, theta, 0.5),
  steps = 5000
) {
  accept.count = 0
  chain = rep(NA, steps)
  t.curr = theta.init
  f.p = function(t) f.prior(t) * f.likelihood(t)

  for(i in 1:steps) {
    t.cand = f.proposal(t.curr)
    p.curr = f.p(t.curr)
    p.cand = f.p(t.cand)
```

```
    p = min(1, p.cand/p.curr)
    if(is.nan(p) || p > runif(1)) { # p is NaN, if p is very small
      accept.count = accept.count + 1
      t.curr = t.cand
    }
    chain[i] = t.curr
  }

  list(chain = chain, acceptance.rate = accept.count/steps)
}


function() {
  # p(x) = dunif(x, -10, 10)
  # p(x|theta) = dnorm(x, theta, 0.2)
  set.seed(1)
  x = rnorm(20, 0.4, 0.2)
  f.prior = function(theta) dunif(theta, -10, 10)
  f.likelihood = function(theta) prod(dnorm(x, theta, 0.2))

  run <- function(sd, steps=2000, startup=200, start=0) {
    r = metropolis(
      f.prior = f.prior, f.likelihood=f.likelihood,
      steps = steps,
      f.proposal = function(theta) rnorm(1, theta, sd),
      theta.init = start
    )

    chain = r$chain[startup:steps]
    #plot(density(chain, adjust=2), main=paste0('Density estimate, Proposal standard deviation: ', sd))
    plot(1:length(r$chain), r$chain, ylab=expression(mu), xlab='Step', type='l', main=sprintf('Trace␣Plot,␣
        ↪ Acceptance␣rate:␣%.1f%%', r$acceptance.rate * 100))
    acf(r$chain, main=paste('Autocorrelation␣of␣samples'))

    print(paste0('Proposal␣standard␣deviation:␣', sd, ",␣mean=␣", mean(r$chain[startup:steps])))

    chain
  }

  saveplot("proposal_distributions", h = 6, w=12, f=function() {
    par(mfrow=c(2, 1),
        oma = c(0,0,0.5,0) + 0.1,
        mar = c(4,4,2.5,0.5) + 1)
    par(mfrow=c(3, 2))
    c1 = run(0.01)
    c2 = run(0.1)
    c3 = run(0.4)
    par(mfrow=c(1, 1))

    mk.int = function(cs) sprintf("[%.3f,␣%.3f]", quantile(cs, 0.05), quantile(cs, 0.95))
    print(xtable(data.table("\\sigma"=c("0.01", "0.1", "0.4"), mean=c(mean(c1), mean(c2), mean(c3)), "95\\%␣
        ↪ credible␣interval"=c(mk.int(c1), mk.int(c2), mk.int(c3))), digits = 3))
  })

}
```

Listing 17: Stan example, examples/stan.R

```
source('~/Dropbox/thesis/r/init.R')
library(rstan)


stan_model = "
data␣{
␣␣int␣N;
␣␣vector[N]␣x;
}
parameters␣{
␣␣real␣mu;
}
model␣{
␣␣mu␣~␣uniform(-10,␣10);
␣␣x␣~␣normal(mu,␣0.2);
}"

# generate data
set.seed(1)
x = rnorm(20, 0.4, 0.2)

# run sampler
res = stan(model_code=stan_model, data=list(N=length(x), x=x), chains=1, warmup=200, iter=2000, init=0)

summary(res)
rstan::plot(res)
```

```
saveplot("stan_example", h = 4, w=12, f=function() {
  par(mfrow=c(1, 2),
      oma = c(0,0,0.5,0) + 0.1,
      mar = c(4,4,2.5,0.5) + 1)
  plot(rstan::stan_trace(res, inc_warmup = T)$data$value, ylim=c(0, 0.6), ylab=expression(mu), xlab='Step',
       ↪ type='l', main='Trace␣Plot␣␣NUTS␣Algorithm')
  acf(rstan::stan_trace(res)$data$value, main='Autocorrelation␣of␣the␣chain', xlab="Lag")
})
```

Listing 18: Mean/Variance optimization example, examples/efficient-frontier.R

```
mus = c(0.11, 0.10, 0.07)
sds = c(0.23, 0.22, 0.12)

corr = matrix(data = c(1,0.4,-0.2, 0.4,1,-0.1, -0.2,-0.1,1), nrow = 3, ncol = 3)
corr

cov = diag(sds) %*% corr %*% diag(sds)
cov

solve.min_var = function(target.mu, mus, cov) {
  ones = rep(1, length(mus))
  cov.inv = solve(cov)
  a = t(ones) %*% cov.inv %*% ones
  b = t(ones) %*% cov.inv %*% mus
  c = t(mus) %*% cov.inv %*% mus

  delta = a * c - b^2
  l1 = (c - b * target.mu) / delta
  l2 = (a*target.mu - b) / delta

  cov.inv %*% (as.numeric(l1) * ones + as.numeric(l2) * mus)
}
solve.gmv = function(mus, cov) {
  ones = rep(1, length(mus))
  (solve(cov) %*% ones) / as.numeric(t(ones) %*% solve(cov) %*% ones)
}

mu.p = function(w, mus) mus %*% w
sd.p = function(w, cov) sqrt(t(w) %*% cov %*% w)

w.opt = solve.min_var(0.1, mus, cov)
w.gmv = solve.gmv(mus, cov)


target.mus = seq(0.01, 0.15, by = 0.001)
ws.efficient = sapply(target.mus, function(target.mu) solve.min_var(target.mu, mus, cov))
mus.efficient = apply(ws.efficient, 2, function(w) mu.p(w, mus))
sds.efficient = apply(ws.efficient, 2, function(w) sd.p(w, cov))

saveplot(name = "mean-var-ex-plot", f = function() {
  par(mfrow=c(1, 1),
      oma = c(0,0,0.5,0) + 0.1,
      mar = c(4,8,0.5,4) + 0.1)

  plot(sds, mus, xlim=c(0, 0.3), ylim=c(0.03, 0.15),xlab=expression(sigma), ylab=expression(mu), pch=19, cex
       ↪ =2)
  lines(sds.efficient, mus.efficient)
  lines(y = c(0.1, 0.1), x=c(-1, sd.p(w.opt, cov)), lty=2, col='red')
  points(sd.p(w.opt, cov), mu.p(w.opt, mus), col='red', pch=19, cex=2)

  points(sd.p(w.gmv, cov), mu.p(w.gmv, mus), col='green', pch=19, cex=2)

  par(mfrow=c(1, 1))
}, h = 6, w=12)



##

mat.to.latex = function(a, digits = 2)
  print(xtable(a,align=rep("",ncol(a)+1), digits = digits), floating=FALSE, tabular.environment="pmatrix",
        hline.after=NULL, include.rownames=FALSE, include.colnames=FALSE)
mat.to.latex(cov, 3)

mat.to.latex(w.opt, 2)
mu.p(w.opt, mus)
sd.p(w.opt, cov)

mat.to.latex(w.gmv, 2)
mu.p(w.gmv, mus)
sd.p(w.gmv, cov)
```

Listing 19: Utility functions plot, examples/utility-functions.R

```
crra.util = function(r, riskaver) ((1 + r) ^ (1 - riskaver) - 1)/(1 - riskaver)

saveplot(name = "crra-utility-plot", f = function() {
  par(mfrow=c(1, 2),
      oma = c(0,0,0.5,0) + 0.1,
      mar = c(4,4,0.5,0.5) + 0.1)
  curve(crra.util(r, 3), -0.2, 0.2, ylim=c(-0.5, 0.2), xname='r', xlab='r', ylab=paste('CRRA␣Utility,␣risk␣
      ↪ aversion␣=␣3')); abline(v=0); abline(h=crra.util(0, 3))
  curve(crra.util(r, 8), -0.2, 0.2, ylim=c(-0.5, 0.2), xname='r', xlab='r', ylab=paste('CRRA␣Utility,␣risk␣
      ↪ aversion␣=␣8')); abline(v=0); abline(h=crra.util(0, 8))
  par(mfrow=c(1, 1))
}, h = 6, w=12)
```

Listing 20: Hierarchical shrinkage example, examples/hierarchial-shrinkage-example.R

```
normallkj_norm.stan = stan_model(file='models/normal-lkj-cholesky.stan')
normallkj_hier.stan = stan_model(file='models/normal-lkj-hierarchial.stan')

library(MASS)

mu = 0; sd = 1
S = 10
R = mvrnorm(12*5, mu = rep(mu, S), Sigma=diag(sd^2, S))

s.norm = sampling(normallkj_norm.stan, data=list(S=S, N=nrow(R), shape=1, y=R), cores=1, chains=1, iter=4000)
s.hier = sampling(normallkj_hier.stan, data=list(S=S, N=nrow(R), shape=1, y=R), cores=1, chains=1, iter=4000)

mus.norm = f.extract(s.norm, 'mus')
mus.hier = f.extract(s.hier, 'mus')
mu_hyper.hier = f.extract(s.hier, 'mu_hyper')

saveplot(name = "hierarchial-shrinkage-example", f = function() {
  par(mfrow=c(1, 1),
      oma = c(0,0,0.5,0) + 0.1,
      mar = c(5,4,0.5,0.5) + 0.1)

  plot(apply(mus.norm, 2, mean), xlab='j', ylab=expression(mu), pch=19)
  points(apply(mus.hier, 2, mean), col='blue', pch=19)
  abline(h=mu, col='red')
  abline(h=mean(mu_hyper.hier), col='blue')
  segments(1:S, apply(mus.hier, 2, mean), 1:S, mean(mu_hyper.hier), col='blue', lty=3, pch=24)

  par(mfrow=c(1, 1))
},h=4, w=12)
```

Listing 21: Efficient frontier example, examples/efficient-frontier.R

```
mus = c(0.11, 0.10, 0.07)
sds = c(0.23, 0.22, 0.12)

corr = matrix(data = c(1,0.4,-0.2, 0.4,1,-0.1, -0.2,-0.1,1), nrow = 3, ncol = 3)
corr

cov = diag(sds) %*% corr %*% diag(sds)
cov

solve.min_var = function(target.mu, mus, cov) {
  ones = rep(1, length(mus))
  cov.inv = solve(cov)
  a = t(ones) %*% cov.inv %*% ones
  b = t(ones) %*% cov.inv %*% mus
  c = t(mus) %*% cov.inv %*% mus

  delta = a * c - b^2
  l1 = (c - b * target.mu) / delta
  l2 = (a*target.mu - b) / delta

  cov.inv %*% (as.numeric(l1) * ones + as.numeric(l2) * mus)
}
solve.gmv = function(mus, cov) {
  ones = rep(1, length(mus))
  (solve(cov) %*% ones) / as.numeric(t(ones) %*% solve(cov) %*% ones)
}

mu.p = function(w, mus) mus %*% w
sd.p = function(w, cov) sqrt(t(w) %*% cov %*% w)

w.opt = solve.min_var(0.1, mus, cov)
w.gmv = solve.gmv(mus, cov)
```

```
target.mus = seq(0.01, 0.15, by = 0.001)
ws.efficient = sapply(target.mus, function(target.mu) solve.min_var(target.mu, mus, cov))
mus.efficient = apply(ws.efficient, 2, function(w) mu.p(w, mus))
sds.efficient = apply(ws.efficient, 2, function(w) sd.p(w, cov))

saveplot(name = "mean-var-ex-plot", f = function() {
  par(mfrow=c(1, 1),
      oma = c(0,0,0.5,0) + 0.1,
      mar = c(4,8,0.5,4) + 0.1)

  plot(sds, mus, xlim=c(0, 0.3), ylim=c(0.03, 0.15),xlab=expression(sigma), ylab=expression(mu), pch=19, cex
       ↪ =2)
  lines(sds.efficient, mus.efficient)
  lines(y = c(0.1, 0.1), x=c(-1, sd.p(w.opt, cov)), lty=2, col='red')
  points(sd.p(w.opt, cov), mu.p(w.opt, mus), col='red', pch=19, cex=2)

  points(sd.p(w.gmv, cov), mu.p(w.gmv, mus), col='green', pch=19, cex=2)

  par(mfrow=c(1, 1))
}, h = 6, w=12)



##

mat.to.latex = function(a, digits = 2)
  print(xtable(a,align=rep("",ncol(a)+1), digits = digits), floating=FALSE, tabular.environment="pmatrix",
        hline.after=NULL, include.rownames=FALSE, include.colnames=FALSE)
mat.to.latex(cov, 3)

mat.to.latex(w.opt, 2)
mu.p(w.opt, mus)
sd.p(w.opt, cov)

mat.to.latex(w.gmv, 2)
mu.p(w.gmv, mus)
sd.p(w.gmv, cov)
```

References

Alvarez, I., Niemi, J., and Simpson, M. (2014). Bayesian inference for a covariance matrix. *arXiv preprint arXiv:1408.4050*.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43.

Avramov, D. (2002). Stock return predictability and model uncertainty. *Journal of Financial Economics*, 64(3):423–458.

Avramov, D. and Zhou, G. (2010). Bayesian portfolio analysis. *Annu. Rev. Financ. Econ.*, 2(1):25–47.

Bade, A., Frahm, G., and Jaekel, U. (2009). A general approach to bayesian portfolio optimization. *Mathematical Methods of Operations Research*, 70(2):337–356.

Barnard, J., McCulloch, R., and Meng, X.-L. (2000). Modeling covariance matrices in terms of standard deviations and correlations, with application to shrinkage. *Statistica Sinica*, pages 1281–1311.

Bernardo, J. M. and Smith, A. F. (1994). Bayesian theory.

Boggs, P. T. and Tolle, J. W. (1995). Sequential quadratic programming. *Acta numerica*, 4:1–51.

Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (2011). *Handbook of markov chain monte carlo*. CRC press.

Carhart, M. M. (1997). On persistence in mutual fund performance. *The Journal of finance*, 52(1):57–82.

Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M. A., Guo, J., Li, P., and Riddell, A. (2016). Stan: A probabilistic programming language. *J Stat Softw*.

Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., and Betancourt, M. (2015). The stan math library: Reverse-mode automatic differentiation in c++. *arXiv preprint arXiv:1509.07164*.

Chang, J. (2007). Stochastic processes.

Chapados, N. (2011). *Portfolio choice problems: An introductory survey of single and multiperiod models*. Springer Science & Business Media.

Conrad, K. (2013). Probability distributions and maximum entropy. *retrieved November*, 14:2013.

Dangl, T. and Halling, M. (2012). Predictive regressions with time-varying coefficients. *Journal of Financial Economics*, 106(1):157–181.

Dangl, T. and Weissensteiner, A. (2017). Long-term asset allocation under time-varying investment opportunities: Optimal portfolios with parameter and model uncertainty. *SSRN Scholarly Paper ID*, 2883768.

DeFusco, R. A., McLeavey, D. W., Anson, M. J., Pinto, J. E., and Runkle, D. E. (2015). *Quantitative investment analysis*. John Wiley & Sons.

Devroye, L. (1986). General principles in random variate generation. In *Non-Uniform Random Variate Generation*. Springer.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, 195(2):216–222.

Fama, E. F. and French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, 33(1):3–56.

Fama, E. F. and French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, 116(1):1 – 22.

French, K. (2018). Kenneth r. french - data library, `http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html`. Accessed: 2018-03-17.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2014). *Bayesian data analysis*, volume 2. Chapman & Hall/CRC Boca Raton, FL, USA.

Gelman, A. and Hill, J. (2007). *Data analysis using regression and multilevelhierarchical models*, volume 1. Cambridge University Press New York, NY, USA.

Greyserman, A., Jones, D. H., and Strawderman, W. E. (2006). Portfolio selection using hierarchical bayesian analysis and mcmc methods. *Journal of Banking & Finance*, 30(2):669–678.

Harvey, C. R., Liechty, J. C., Liechty, M. W., and Müller, P. (2010). Portfolio selection with higher moments. *Quantitative Finance*, 10(5):469–485.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.

Head, M. L., Holman, L., Lanfear, R., Kahn, A. T., and Jennions, M. D. (2015). The extent and consequences of p-hacking in science. *PLoS Biol*, 13(3):e1002106.

Hoffman, M. D. and Gelman, A. (2011). The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *ArXiv e-prints*.

Hornik, K., Leisch, F., and Zeileis, A. (2003). Jags: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of DSC*, volume 2, pages 1–1.

Jacquier, E. and Polson, N. (2010). Simulation-based-estimation in portfolio selection. *Frontiers of Statistical Decision Making and Bayesian Analysis: In Honor of James O. Berger. Springer*, pages 396–410.

Jagannathan, R. and Ma, T. (2002). Risk reduction in large portfolios: a role for portfolio weight constraints. *Unpublished Working Paper*.

James, W. and Stein, C. (1961). Estimation with quadratic loss. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 361–379.

Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 186, pages 453–461. The Royal Society.

Jegadeesh, N. and Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of finance*, 48(1):65–91.

Johnson, T. (2007). Utility functions. *C2922 economics, Heriot Watt University, Edinburgh*.

Jordan, M. I. (2010). Monte carlo sampling.

Jorion, P. (1986). Bayes-stein estimation for portfolio analysis. *Journal of Financial and Quantitative Analysis*, 21(03):279–292.

Kalymon, B. A. (1971). Estimation risk in the portfolio selection model. *Journal of Financial and Quantitative Analysis*, 6(01):559–582.

Kan, R. and Zhou, G. (2007). Optimal portfolio choice with parameter uncertainty. *Journal of Financial and Quantitative Analysis*, 42(3):621–656.

Kruschke, J. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

Lewandowski, D., Kurowicka, D., and Joe, H. (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of multivariate analysis*, 100(9):1989–2001.

Lunn, D. J., Thomas, A., Best, N., and Spiegelhalter, D. (2000). Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337.

Markowitz, H. (1952). Portfolio selection. *The journal of finance*, 7(1):77–91.

McCulloch, R. and Rossi, P. E. (1990). Posterior, predictive, and utility-based approaches to testing the arbitrage pricing theory. *Journal of Financial Economics*, 28(1-2):7–38.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092.

Norstad, J. (1999). An introduction to utility theory.

Plummer, M. et al. (2003). Jags: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, volume 124, page 125. Vienna.

Polson, N. G. and Tew, B. V. (2000). Bayesian portfolio selection: an empirical analysis of the s&p 500 index 1970–1996. *Journal of Business & Economic Statistics*, 18(2):164–173.

Qian, H. (2009). Bayesian portfolio selection with gaussian mixture returns.

Qian, H. (2011). Bayesian portfolio selection in a markov switching gaussian mixture model.

Rapach, D. E., Strauss, J. K., and Zhou, G. (2010). Out-of-sample equity premium prediction: Combination forecasts and links to the real economy. *The Review of Financial Studies*, 23(2):821–862.

Robert, C. and Casella, G. (2011). A short history of markov chain monte carlo: subjective recollections from incomplete data. *Statistical Science*, pages 102–115.

Sahu, S. K., Dey, D. K., and Branco, M. D. (2003). A new class of multivariate skew distributions with applications to bayesian regression models. *Canadian Journal of Statistics*, 31(2):129–150.

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of business*, 39(1):119–138.

Shonkwiler, R. W. and Mendivil, F. (2009). *Explorations in Monte Carlo Methods*. Springer Science & Business Media.

Shore, J. and Johnson, R. (1980). Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on information theory*, 26(1):26–37.

Shumway, T. (1997). The delisting bias in crsp data. *The Journal of Finance*, 52(1):327–340.

Sinn, H.-W. (1980). A rehabilitation of the principle of insufficient reason. *The Quarterly Journal of Economics*, pages 493–506.

Stan Development Team (2016). Stan modeling language users guide and reference manual, version 2.14.0, `http://mc-stan.org`.

Syversveen, A. R. (1998). Noninformative bayesian priors. interpretation and problems with construction and applications. *Preprint Statistics*, 3.

Weiss, N., Holmes, P., and Hardy, M. (2005). *A Course in Probability*. Pearson Addison Wesley.