LUISS

Dipartimento
di Economia e Finanza

Cattedra di Mathematical methods for Economics and Finance

Montecarlo and Value at Risk: empirical evidence from the Italian stock market

Emerito Prof. G. Olivieri
RELATORE

Ch.ma Prof.ssa P. Fersini
CORRELATORE

Martina Aquila 694261
CANDIDATO

Anno Accademico 2018 / 2019

Declaration of Authorship

I, Martina AQUILA, declare that this thesis titled, "Montecarlo and Value at Risk: empirical evidence from the Italian Stock market" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 30/09/2019

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin"

John von Neumann

LUISS GUIDO CARLI

Abstract

Department or Economics and Finance

Master Degree in Economics and Finance

Montecarlo and Value at Risk: empirical evidence from the Italian Stock market

by Martina AQUILA

This thesis describes Montecarlo simulation together with random number generation techniques, with a focus on their principal application to market risk management: Value at Risk (VaR).

After three chapters devoted to description and efficiency considerations on random number generation, VaR computations are explained in steps.

Chapter five presents the empirical results of this work.

VaR was computed for the Italian and US stock markets according to the three most widely used VaR approaches (Variance-Covariance, Historical, and Montecarlo simulations). It is found that Montecarlo outperforms the other two methods in both markets concerning the test statistics proposed.

Finally, the accuracy of Montecarlo estimates on the Italian stock market was evaluated with respect to distributional assumptions.

Among Normal, Log-Normal and Geometric Brownian motion the latter outperforms the other two.

Keywords: VaR,Montecarlo simulation, market risk, capital requirements for market risk, random number generation, multiple recursion

Acknowledgements

Arrivata alla conclusione del mio percorso di studi, ritengo necessario ringraziare alcune figure che hanno, pur in modo diverso, segnato la mia crescita in questi anni.

In primo luogo, ringrazio il mio Relatore, Emerito Prof. G. Olivieri, per aver accettato di accompagnarmi per la seconda volta nell'atto conclusivo del percorso accademico. I suoi insegnamenti resteranno impressi nella mia memoria.

Grazie alle due persone più importanti della mia vita per avermi insegnato a riconoscere i miei limiti e a imparare dagli stessi. A loro dedico tutto quello che ho scritto e la fatica che ne è derivata, perchè solo quando si fatica si può provare soddisfazione. Questo me lo avete insegnato voi. Spero che siate fieri di me. La fortuna inizia con la nascita.

Grazie a mia Sorella Giorgia per avermi sempre capita senza bisogno di parlare. Grazie a tutte le mie amiche per la spensieratezza che mi hanno regalato in questi anni.

Grazie ai Nonni lontani e vicini.

Grazie a Pietro.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction to Montecarlo simulation 3
- 1.1 Origins . . . 3
- 1.2 History . . . 7
- 1.3 Applications . . . 10
- 1.4 The need for simulation techniques . . . 13

2 Generation of uniform random numbers 15
- 2.1 Common considerations . . . 16
  - 2.1.1 Generic recursive generators . . . 17
  - 2.1.2 Modular arithmetics . . . 17
- 2.2 Linear congruential generators . . . 18
  - 2.2.1 Definition . . . 18
  - 2.2.2 Properties and parameter choice . . . 18
  - 2.2.3 MATLAB implementation . . . 20
  - 2.2.4 Add with carry and Multiply with Carry . . . 24
- 2.3 Multiple recursive generators . . . 24
- 2.4 Matrix congruential generators . . . 25
- 2.5 Inversive congruential generators . . . 27
- 2.6 Mixed generators . . . 27
  - 2.6.1 Wichmann Hill generator . . . 28
  - 2.6.2 L'Ecuyer mixed generator . . . 29
  - 2.6.3 Other mixed generators . . . 29
- 2.7 Statistical tests . . . 31
  - 2.7.1 Static tests . . . 32
  - 2.7.2 Dynamic tests . . . 34

3 Generation of random numbers from other distributions 35
- 3.1 Exact methods . . . 35
  - 3.1.1 Inverse-transform method . . . 35
  - 3.1.2 Alias method . . . 39

3.1.3 Acceptance-rejection method . . . 41
3.1.4 Ratio of Uniforms method . . . 43
3.2 Specific distributions . . . 43
3.2.1 Bernoulli Distribution . . . 44
3.2.2 Binomial Distribution . . . 44
3.2.3 Geometric Distribution . . . 46
3.2.4 Hypergeometric Distribution . . . 48
3.2.5 Negative Binomial Distribution . . . 48
3.2.6 Poisson Distribution . . . 49
3.2.7 Beta Distribution . . . 50
3.2.8 Cauchy Distribution . . . 53
3.2.9 Exponential Distribution . . . 54
3.2.10 Fisher-Snedecor Distribution . . . 55
3.2.11 Frechet Distribution . . . 56
3.2.12 Gamma Distribution . . . 57
3.2.13 Gumbel Distribution . . . 59
3.2.14 Laplace Distribution . . . 60
3.2.15 Logistic Distribution . . . 62
3.2.16 Log-Normal Distribution . . . 62
3.2.17 Normal Distribution . . . 63
3.2.18 Pareto Distribution . . . 64
3.2.19 Student-t Distribution . . . 65
3.2.20 Uniform Distribution . . . 68
3.2.21 Wald Distribution . . . 69
3.2.22 Weibull Distribution . . . 69

4 Value at Risk **71**
4.1 Market risk measurement . . . 71
4.2 Value at Risk . . . 73
4.2.1 VaR components . . . 74
4.2.2 Common steps: risk mapping . . . 75
4.2.3 A simple example . . . 78
4.3 Parametric approach . . . 79
4.3.1 Computation . . . 79
4.3.2 Example . . . 81
4.3.3 Advantages and disadvantages . . . 88
4.4 Historical simulation approach . . . 88
4.4.1 Computation . . . 89
4.4.2 Example . . . 90
4.4.3 Advantages and disadvantages . . . 97
4.5 Montecarlo simulation approach . . . 98
4.5.1 Computation . . . 99

4.5.2 Example . . . 103
4.5.3 Advantages and disadvantages . . . 118

5 Empirical performance of VaR models **121**
5.1 Relevance of the topic and literature review . . . 122
5.2 A comparison of VaR techniques: the Italian Stock market . . . 123
5.2.1 FTSE MIB . . . 123
5.2.2 Data . . . 124
5.2.3 Methodology . . . 125
5.2.4 Results . . . 130
5.2.5 Conclusions . . . 133
5.3 Replication of the study on S&P500 index . . . 133
5.3.1 S&P500 . . . 134
5.3.2 Data . . . 134
5.3.3 Methodology . . . 135
5.3.4 Results . . . 135
5.3.5 Conclusions . . . 138
5.4 The impact of distributional assumptions on VaR estimates . . . 139
5.4.1 Data . . . 139
5.4.2 Methodology . . . 139
5.4.3 Results . . . 140
5.4.4 Conclusions . . . 142
5.5 Conclusions . . . 142

A Proofs **147**
A.1 Proof of 3.1 . . . 147
A.2 Proof of Acceptance-Rejection method . . . 147
A.3 Proof of Ratio of uniforms relation . . . 147

B Codes **149**
B.1 Chapter 1 . . . 149
B.1.1 Buffon needle experiment . . . 149
B.1.2 Histogram of Buffon needle experiment . . . 149
B.1.3 Monte Carlo double . . . 150
B.2 Chapter 2 . . . 150
B.2.1 LCG . . . 150
B.2.2 LCG non full . . . 151
B.2.3 LCG LGM . . . 151
B.2.4 LCG with histogram . . . 152
B.2.5 LCG uniform . . . 152
B.2.6 MCG . . . 153
B.2.7 MCG Matrix . . . 153
B.2.8 W-H . . . 153

B.2.9 MRG32k3a . . . 154
B.2.10 KISS99 . . . 155
B.3 Chapter 3 . . . 155
B.3.1 Inverse transform-Continuous variable . . . 155
B.3.2 Inverse transform-Beroulli . . . 156
B.3.3 Inverse transform-Discrete 1 . . . 156
B.3.4 Inverse transform-Discrete 2 . . . 156
B.3.5 Alias method . . . 157
B.3.6 Acceptance-Rejection Positive Normal . . . 158
B.3.7 Bernoulli . . . 158
B.3.8 Binomial via Normal . . . 159
B.3.9 Binomial Recursive 1 . . . 159
B.3.10 Binomial Recursive 2 . . . 159
B.3.11 Geometric via Exponential . . . 160
B.3.12 Geometric via Uniform . . . 160
B.3.13 Hypergeometric . . . 161
B.3.14 Negative Binomial . . . 161
B.3.15 Poisson . . . 161
B.3.16 Beta(a,1) . . . 162
B.3.17 Beta(1,b) . . . 162
B.3.18 Beta(0.5,0.5) . . . 163
B.3.19 Beta(a,a) 1 . . . 163
B.3.20 Beta(a,a) 2 . . . 163
B.3.21 Beta Johnk algorithm . . . 164
B.3.22 Cauchy(0,1) . . . 165
B.3.23 Cauchy(0,1) Ratio of Uniforms . . . 165
B.3.24 Cauchy(0,1) Ratio of Normals . . . 166
B.3.25 Exponential . . . 166
B.3.26 F 1 . . . 166
B.3.27 F 2 . . . 167
B.3.28 Frechet . . . 167
B.3.29 Gamma Best . . . 167
B.3.30 Gamma Cheng Feast . . . 168
B.3.31 Gamma Chi-square . . . 169
B.3.32 Gumbel . . . 169
B.3.33 Laplace 1 . . . 170
B.3.34 Laplace 2 . . . 170
B.3.35 Laplace 3 . . . 170
B.3.36 Laplace 4 . . . 171
B.3.37 Logistic . . . 171
B.3.38 Log-Normal . . . 172
B.3.39 Normal Box-Muller . . . 172

B.3.40 Normal Rejection Polar method . . . 172
B.3.41 Pareto 1 . . . 173
B.3.42 Pareto 2 . . . 173
B.3.43 Student-t via Chi Square and Standard Normal . . . 174
B.3.44 Student-t via Beta . . . 174
B.3.45 Student-t via Ratio of Uniform . . . 174
B.3.46 Student-t via Polar 1 . . . 175
B.3.47 Student-t via Polar 2 . . . 175
B.3.48 Student-t via Polar Bailey . . . 176
B.3.49 Uniform(a,b) . . . 176
B.3.50 Wald . . . 176
B.3.51 Weibull . . . 177
B.3.52 Raileigh . . . 177
B.4 Chapter 4 . . . 178
B.4.1 Parametric VaR for Portfolio . . . 178
B.4.2 Historical simulation VaR for Portfolio . . . 179
B.4.3 Montecarlo simulation VaR for Portfolio . . . 181
B.5 Chapter 5 . . . 185
B.5.1 FTSE MIB Study . . . 185
B.5.2 S&P500 Study . . . 202
B.5.3 Distributional assumption Study . . . 219

List of Figures

1.1 Histogram of results for estimation of π 7
1.2 Graphical representation of an integral 12

2.1 LCG uniform histogram 22
2.2 Direct generation of a uniform distribution n=100 24

4.1 Normal Distribution of P&L 78
4.2 Evolution of USD vs. GBP exchange rate 2010-2019 83
4.3 Distribution of USD vs. GBP exchange rate 2010-2019 83
4.4 Evolution of USD T-bills 2010-2019 84
4.5 Distribution of USD T-bill rates 2010-2019 84
4.6 Evolution of UK T-bills 2010-2019 85
4.7 Distribution of UK T-bill rates 2010-2019 86
4.8 Historical and simulated USD vs. GBP exchange rate 92
4.9 Historical and simulated USD T-bill rates 92
4.10 Historical and simulated UK T-bill rates 93
4.11 Simulated evolution of the portfolio 94
4.12 Simulated distribution of P&L 94
4.13 USD 3-Month T-bill rates and Normal distribution 101
4.14 USD-GBP histogram 104
4.15 USD vs. GBP exchange rate and Normal distribution 105
4.16 USD vs. GBP exchange rate and Birnbaum-Saunders distribution . . . 106
4.17 USD vs. GBP exchange rate and Extreme value distribution 106
4.18 USD vs. GBP exchange rate and Gamma distribution 107
4.19 USD vs. GBP exchange rate and Weibull distribution 107
4.20 USD vs. GBP exchange rate and Logistic distribution 108
4.21 USD vs. GBP exchange rate and Generalized Extreme Value distribution 108
4.22 USD T-bill rates and Normal distribution 109
4.23 USD T-bill rates and Birnbaum-Saunders distribution 110
4.24 USD T-bill rates and Rayleigh distribution 110
4.25 USD T-bill rates and Exponential distribution 111
4.26 UK T-bill rates and Normal distribution 112
4.27 UK T-bill rates and Gamma distribution 112
4.28 UK T-bill rates and Rayleigh distribution 113
4.29 UK T-bill rates and Rician distribution 114

4.30 Simulated evolution of the portfolio 114

5.1 Evolution of FTSE MIB 25 October 2016-1 January 2018 124
5.2 Empirical distribution of FTSE MIB 126
5.3 VaR estimates from Variance-Covariance method and realized profits or losses 131
5.4 VaR estimates from Historical Simulation method and realized profits or losses 131
5.5 VaR estimates from Montecarlo Simulation method and realized profits or losses 131
5.6 Historical evolution of SP500 whole sample 135
5.7 Distribution of SP500 in-sample period 136
5.8 VaR estimates from Variance-Covariance method and realized profits or losses of SP500 136
5.9 VaR estimates from Historical Simulation method and realized profits or losses of SP500 137
5.10 VaR estimates from Montecarlo Simulation method and realized profits or losses of SP500 137
5.11 VaR estimates from Geometric Brownian Motion assumption and realized profits or losses 140
5.12 VaR estimates from Normal assumption and realized profits or losses . 140
5.13 VaR estimates from Log-Normal assumption and realized profits or losses 141

List of Tables

1.1 Distribution of absolute frequencies 5
1.2 Density 6

2.1 Multiplicative congruential generators 20

3.1 Distribution of r.v. X 38

5.1 Summary statistics of the FTSE MIB 25 October 2016-1 January 2018 . . 124
5.2 Summary statistics of the FTSE MIB in-sample period 125
5.3 Binomial test statistic values FTSE MIB 132
5.4 Kupiec's tests statistic values FTSE MIB 132
5.5 Christoffersen's tests statistic values FTSE MIB 133
5.6 Summary statistics of the S&P 500 25 October 2016-1 January 2018 . . . 134
5.7 Summary statistics of the S&P 500 in-sample period 135
5.8 Binomial test statistic values SP500 138
5.9 Kupiec's tests statistic values SP500 138
5.10 Christoffersen's tests statistic values SP500 138
5.11 Binomial test statistic values: distributional study 141
5.12 Kupiec's tests statistic values: distributional study 142
5.13 Christoffersen's tests statistic values: distributional study 142

Listings

1.1 Buffon needle experiment . . . 4
1.2 Histogram of Buffon needle experiment . . . 6
1.3 Monte Carlo double . . . 12
2.1 LCG . . . 20
2.2 LCG non full . . . 21
2.3 LCG LGM . . . 21
2.4 LCG with histogram . . . 22
2.5 LCG uniform . . . 23
2.6 MCG . . . 24
2.7 MCG Matrix . . . 26
2.8 W-H . . . 28
2.9 MRG32k3a . . . 29
2.10 KISS99 . . . 30
3.1 Inverse transform-Continuous variable . . . 37
3.2 Inverse transform-Bernoulli . . . 38
3.3 Inverse transform-Discrete 1 . . . 38
3.4 Inverse transform-Discrete 2 . . . 39
3.5 Alias method . . . 40
3.6 Acceptance-Rejection Positive Normal Distribution . . . 42
3.7 Bernoulli . . . 44
3.8 Binomial geometric . . . 45
3.9 Binomial via Normal . . . 45
3.10 Binomial Recursive 1 . . . 46
3.11 Binomial Recursive 2 . . . 46
3.12 Geometric via Exponential . . . 47
3.13 Geometric via Uniform . . . 47
3.14 Hypergeometric . . . 48
3.15 Negative Binomial . . . 49
3.16 Poisson . . . 49
3.17 Beta(a,1) . . . 50
3.18 Beta(1,b) . . . 51
3.19 Beta(0.5,0.5) . . . 51
3.20 Beta(a,a) 1 . . . 51
3.21 Beta(a,a) 2 . . . 52
3.22 Beta Johnk algorithm . . . 52

3.23 Cauchy(0,1) . . . 53
3.24 Cauchy(0,1) Ratio of Uniforms . . . 53
3.25 Cauchy(0,1) Ratio of Normals . . . 54
3.26 Exponential . . . 55
3.27 F 1 . . . 55
3.28 F 2 . . . 56
3.29 Frechet . . . 56
3.30 Gamma Best . . . 57
3.31 Gamma Cheng Feast . . . 58
3.32 Gamma Chi-Square . . . 59
3.33 Gumbel . . . 59
3.34 Laplace 1 . . . 60
3.35 Laplace 2 . . . 60
3.36 Laplace 3 . . . 61
3.37 Laplace 4 . . . 61
3.38 Logistic . . . 62
3.39 Log-Normal . . . 62
3.40 Normal Box-Muller . . . 63
3.41 Normal Rejection Polar method . . . 64
3.42 Pareto 1 . . . 64
3.43 Pareto 2 . . . 65
3.44 Student-t via Chi Square and Standard Normal . . . 66
3.45 Student-t via Beta . . . 66
3.46 Student-t Ratio of Uniform . . . 66
3.47 Student-t Polar 1 . . . 67
3.48 Student-t Polar 2 . . . 67
3.49 Student-t Polar Bailey . . . 68
3.50 Uniform(a,b) . . . 68
3.51 Wald . . . 69
3.52 Weibull . . . 70
3.53 Rayleigh . . . 70
4.1 Parametric VaR for Portfolio . . . 86
4.2 Historical simulation VaR for Portfolio . . . 95
4.3 Montecarlo simulation VaR for Portfolio . . . 115

List of Abbreviations

AWC	Add-With-Carry
BDF	Backward Difference Formula
CDF	Cumulative Distribution Function
CFL	Courant-Friedrichs-Lewy
CLT	Central Limit Theorem
CP	Compound Poisson
ECDF	Empirical Cumulative Distribution Function
GBM	Geometric Brownian Motion
HS	Historical Simulation
ICG	Inversive Congruential Generator
LCG	Linear Congruential Generator
LGM	Lewis, Goodam, Miller
MC	Monte Carlo
MCG	Matrix Congruential Generator
MCM	Monte Carlo Method
MG	Mixed Generator
MRG	Multiple Recursive Generator
MWC	Multiply-With-Carry
ODE	Ordinary Differential Equation
OTC	Over the counter
PDE	Partial Differential Equation
QMC	Quasi Monte Carlo
S&P500	Standard & Poor's 500
SDE	Stochastic Differential Equation
TVD	Total Variation Diminishing
VaR	Value at Risk
VC	Variance Covariance
WH	Wichmann-Hill

a.a.	almost all
a.e.	almost everywhere
a.s.	almost surely
ch.f.	characteristic function
d.f.	distribution function
iff	if and only if

i.i.d.	independent and identically distributed
inf	infimum
max	maximum value
m.g.f.	moment generating function
min	minimum value
pdf	probability distribution function
r.v.	random variable
sup	supremum

List of Symbols

$\mathbb{R}$	set of real numbers
$\mathbb{C}$	set of complex numbers
$\mathbb{N}$	set of positive integers
$\mathbb{Z}$	set of integers
$\mathcal{B}$	σ-field of one-dimensional Borel subsets
$\mathcal{F}$	set of all distributions
I	Identification function
$\emptyset$	empty set
$(\Omega, \Sigma, \mathbb{P})$	probability space
$\sigma(X_1, ..., X_n)$	sigma algebra generated by the random variables $X_1, ..., X_n$
$a \perp b$	a is singular relative to b
$\in$	belonging to
$\notin$	non-belonging to
$\exists$	there exists
$\subseteq$	subset of
$\equiv$	identical
$\|\|A\|\|$	norm of A
P	probability
$\mathbb{E}(X)$	expected value of the random variable X
$Var(X)$	variance of the random variable X
$Cov(X, Y)$	covariance of the random variables X and Y
$Corr(X, Y)$	correlation of the random variables X and Y
ρ	correlation coefficient
log	natural logarithm
$O(h^k)$	Landau symbol for $h \to 0$
$\sim$	distributed as
N	Normal distribution
U	Uniform distribution
Γ_α	Exponential distribution with parameter α
$\Gamma_{\alpha,\lambda}$	Gamma distribution with parameters α and λ
Π_λ	Poisson distribution with parameter λ
χ^2	Chi-square distribution
B_p	Binomial distribution
$F_{\beta,\rho}$	Stable distribution with parameters β and ρ
$K_{\alpha,\sigma}$	Cauchy distribution with parameters α and σ

$[a, b]$	closed interval
$[a, b)$	half-open interval
(a, b)	open interval
$\backsimeq$	topological equivalence
μ	mean
σ	standard deviation
σ^2	variance
Δ	small increment
P_X	probability (density) function of the random variable X
P_{XY}	joint probability (density) function of the random variables X and Y
$\bigoplus$	Bitwise binary exclusive-or
$x!$	factorial of x
x^-	multiplicative inverse of x
$\overline{x}$	sample mean of x
$\xrightarrow{p}$	convergence in probability
$\xrightarrow{a.s.}$	almost sure convergence
$\binom{n}{k}$	the k-th binomial coefficient in $(1+x)^n, n \in \mathbb{N}, 0 \leq k \leq n$
$\lvert x \rvert$	modulus of x
$\lceil x \rceil$	ceiling of x
$\lfloor x \rfloor$	floor of x
x_i	the i-th element of a structure
$x_{(i)}$	the i-th order statistics
$x^{(i)}$	the value of x at the i-th iteration

a mamma e papà

Introduction

Montecarlo methods are a widely used set of tools whose principal aim is to find a solution to problems that are by nature so complex that a solution may be difficult or impossible to be found with traditional analytical techniques. By applying a Montecarlo method, an approximation of the solution is determined by the recurrence to random number generation. The quality and reliance of the approximation usually depend on the number of simulated paths generated.

Montecarlo techniques are generic tools that are useful for the evaluation of very different problems. The birth of Montecarlo methods, for instance, is usually associated with the first development of the atomic bomb and, consequently, in the field of physical applications. Nowadays, Montecarlo simulations are applied to many different disciplines, and, among these, finance and risk management play a significant role.

In the field of risk management, Montecarlo techniques are mainly applied to the estimation of a very relevant market risk measure: Value at Risk (VaR). The concept of VaR is central in risk management both from the perspective of financial firms, which use this quantity for capital allocation and risk control purposes and from a regulatory perspective. The Basel accords (1996) in fact, show a clear preference for the estimation of capital requirements for market risk using the VaR measure. As shown in the following chapters, VaR can be computed according to many different approaches. Among these, the most used ones are the Variance-Covariance, Historical, and Montecarlo simulations. Those approaches come necessarily with strengths and weaknesses that will be underlined and commented. Nevertheless, an important objective of this work is to draw some conclusions in terms of the superiority of one of the methods compared to the others. It is important to keep in mind that it is not possible to define one method as superior compared to the others concerning all dimensions considered. The choice, in fact, usually depends on many variables (such as the characteristics and size of the portfolio, the firm's risk profile, the time horizon). In abstract terms, banks are set free to choose the model that best suits specific objectives.

When Montecarlo simulation is to be applied, a good random number generator must be available. If this is not the case, the VaR estimate that results is necessarily biased. This is the reason why the first three chapters of this work are devoted to the description of Montecarlo techniques with a clear focus on random number generation. Those descriptions are necessary if we are interested in the quality of VaR estimates. Subsequently, the concept of VaR and the three approaches to its

computation are introduced in Chapter 4. Together with general considerations, a practical implementation for the computation of VaR is given.

Finally, Chapter 5 analyzes the performance of the three VaR approaches over multiple dimensions. The idea is to look empirically at the performance of the models so to make some conclusions. The major takeaway of the analysis is that the increased complexity of the Montecarlo method comes with improvements in terms of performance. This conclusion should be taken consciously since it is necessarily influenced by the study performed. For this reason, the limitations of this study are clearly set out.

Chapter 1

Introduction to Montecarlo simulation

Montecarlo methods are a comprehensive class of stochastic simulation techniques that employ the generation of random numbers in order to estimate deterministic quantities. Montecarlo methods are widely used in many different fields of studies. The common feature is that the problem to which Montecarlo simulation seeks to find a solution is the expected value of a random variable. Realizations of those random variables are generated by a stochastic process that Montecarlo simulates. In this way, the expected value of the random variable is computed by the sample generated through the Montecarlo technique. This non-rigorous and straightforward description of a trivial Montecarlo algorithm highlights the major weakness of this well-known technique. It has to be clear already from the first page that Montecarlo techniques do not provide an exact solution to any problem; they provide a reliable approximation if some conditions are met. However, the solution is always an approximation of the actual expected value. It follows, then, that Montecarlo techniques should be used when, given the complexity of the problem, it is not possible to define the exact solution in closed form. The reliability of the approximation, of course, depends on the number of trials that are performed. This treat is typical and common to all numerical techniques, to which Montecarlo belongs. In the following chapters, a more formal definition will be given, and various methods to reduce estimation biases will be discussed.

1.1 Origins

In literature, it is not always recalled that Montecarlo methods are extremely ancient, in the fact that the first technique belonging to this family was proposed in 1733 by George-Louis Leclerc Compte de Buffon (1707-1788). Buffon, a naturalist, mathematician, and cosmologist, wanted to estimate the value of π through an experiment. The experiment he designed has the characteristics of what we call now a Montecarlo simulation. This famous experiment is called "Buffon's needle" and it is based on the idea that it is possible to estimate the value of π by flipping a needle on

a surface characterized by the presence of parallel and equidistant lines. By repeating this flipping, it is possible to observe how many times the needle crosses a line empirically. We turn now to the formalization of this problem in order to explain why it is considered the father of Montecarlo techniques [1]. Define a needle with length L and a flat surface characterized by a bundle of parallel and equidistant lines. The distance between each couple of lines is defined d. We assume, by construction, that $d \geq L$. The random variable at the heart of the experiment is X_n= number of successes in n trials, i.e., the number of times the needle touches or crosses a line in n flips. Imagine the needle crosses a line. Then it is possible to draw another line that passes through the center of the needle and parallel to the lines on the surface. We define D as the distance of this parallel line to the nearest line on the surface and θ the acute angle the needle generates with the bundle of lines. Then $D \in [0, d/2]$ and $\theta \in [0, \pi/2]$. The needle will cross or touch the line only if $D \leq \frac{L}{2\sin\theta}$. Now we are ready to define the probability of success in a trial, i.e., the probability of the needle crossing or touching a line: $P(success) = \frac{\int_0^{\frac{\pi}{2}} \frac{L}{2}\sin\theta d\theta}{\frac{L}{2}\frac{\pi}{2}} = \frac{2L}{\pi d}$. By estimating, trough the experiment, the probability of success, it is straightforward to solve for π: $\pi = \frac{2L}{dP(success)}$. The result is then an empirical estimation of π. At that time this was a great success since it was not conceivable to estimate π trough a random experiment. When Buffon formalized this experiment, its execution was, of course, physical. Today it is possible to simulate it by the use of computers. Specifically, by choosing some parameters, it is possible to ask the program to simulate some variables. Then the result will be an empirical estimation of π. First, we set, without loss of generality but with significant savings in terms of computations, $d = 2$ and $L = 1$. The number of trials is set $n = 20000$. Then the program will generate the two random variables D and θ, respecting the intervals defined above. A success is recorded any time $D \leq \frac{L}{2\sin\theta}$. The empirical estimation of π is then simply given by $\pi = \frac{2L}{dP} = \frac{1}{P} = \frac{1}{\frac{successes}{n}} = \frac{n}{successes}$. The code employed is the following:

LISTING 1.1: Buffon needle experiment

```
1
%% Buffon needle experiment: estimation of pi
3
%% simulation
5 % set d=2 and L=1
clc
7 clear all

9 n=20000;
D=rand(1,n);% vector of n pseudorandom %numbers in the range [0,1)
11 theta=rand(1,n)*pi/2;% vector of n pseudorandom
```

[1] Shonkwiler R. and Mendivil F. 2009. Explorations in Monte Carlo Methods (1st ed.). Springer Publishing Company, Incorporated.

```
numbers in the range [0,pi/2)
succ=D<=sin(theta)/2; % vector of 0 if false and 1 if true
sum(succ) %show the number of hits
pi_hat=n/sum(succ) %our estimated value of pi!
```

If we run several times this code, we get many different estimated values for π. Each of those numbers is close to the true 3.14159265.... Consequently, for every repetition of the simulation, we can determine the "goodness" of the estimation. However, this is not generally the case since in simulation contexts the variable to be estimated is not known in advance. That is, $\pi = 3.14159265...$ is unknown. In a typical setup, then, it would be useful to visualize all the results obtained in different simulations, in order to understand where the "true" value of the estimated quantity (π) stands. The basic assumption is, of course, that we run the simulation exercise several times. Once we obtained the k estimated values, it is possible to histogram the results in order to get a visual representation. A histogram is a handy tool in the fact that it helps to understand the (estimated) shape of the distribution of results.

Hystogram. A histogram is a graphical representation of a distribution of quantitative data. Consider a statistical distribution with values $X = (x_1, x_2, \ldots, x_{k-1}, x_k)$. The interval in which the variable takes values is then $[x_1, x_k]$. Assume that the number of times the variable assumes each value x_i is known as n_i, where $\sum_{i=1}^{n} n_i = N$, the size of the sample. Then, what is typically available is a chart in the form:

TABLE 1.1: Distribution of absolute frequencies

value	number of occurrencies
x_1	n_1
x_2	n_2
...	...
x_k	n_k

This is known as the absolute frequency of the variable X. The relative frequency, f_i, is $f_i = \frac{n_i}{N}$. It is possible to divide the interval of existence of X in sub-intervals in the form $[v_0, v_1), [v_1, v_2), \ldots, [v_{k-1}, v_k)$. The number and length of the intervals is a choice of the agent, and it is usually tricky. It is convenient to set the intervals such that they have the same length. For what concerns the number, a good balance should be found since if intervals are too many the quality and readability of the graph may be altered. If, on the other hand, they are too few, it may be difficult to draw considerations, since the length of each interval would be wide. Once the classes (intervals) have been defined, it is necessary to define the density of the frequency for each class, that is $d_i = \frac{n_i}{v_{i+1} - v_i}$. At, the end what we need to draw a histogram is the content of Table 1.2.

TABLE 1.2: Density

class	absolute frequency	density
$[v_0, v_1)$	n_1	$\frac{n_1}{v_1 - v_0}$
$[v_1, v_2)$	n_2	$\frac{n_2}{v_2 - v_1}$
...	...	...
$[v_{k-1}, v_k)$	n_k	$\frac{n_k}{v_k - v_{k-1}}$

We are ready to draw the histogram of the distribution that is a graph made of adjacent rectangles. Each rectangle represents a class: its height is given by the density, the base is given by the width of the class and the area is the frequency.

Recall that the goal of the experiment is to estimate π. In order to run this experiment, it is necessary to implement the Montecarlo algorithm in Code 1.1. This Montecarlo algorithm is itself an experiment since it simulates randomly D and θ. Every time we run the Montecarlo simulation, we refer to it by trial. In order to understand if the estimated value of π is reliable it is necessary to run a second experiment, i.e., run the Montecarlo simulation m times, that is performing m trials, where m is sufficiently large. In this way, it will be possible to look at the distribution of the estimated values and verify how close they are to $\pi = 3.14159265\ldots$. Then, we have to set the number of repetitions of the original experiment, n, and the number of times we want to repeat the experiment itself, m. Once m estimations for π have been obtained, the results are presented in a histogram, in order to get an idea on the distribution of π. The MATLAB code employed to get these results is the following.

LISTING 1.2: Histogram of Buffon needle experiment

```
%% Buffon needle experiment: estimation of pi

%% histogramming results

n=100;
m=2000;
for i=1:m
D=rand(1,n)
theta=pi/2*rand(1,n)
succ=D<=sin(theta)/2
pi_hat(i)=n/sum(succ) %vector of estimated values of pi
end
```

```
%descriptive analysis of results
hist(pi_hat) %histogram of distribution
mean_pi_hat=n/sum(pi_hat) %sample mean
dev=(pi_hat-mean_pi_hat).*(pi_hat-mean_pi_hat) %squared deviations
from the mean
var_pi_hat=sum(dev)/(n-1) %sample variance
sdev_pi_hat=sqrt(var_pi_hat) %sample standard deviation
```

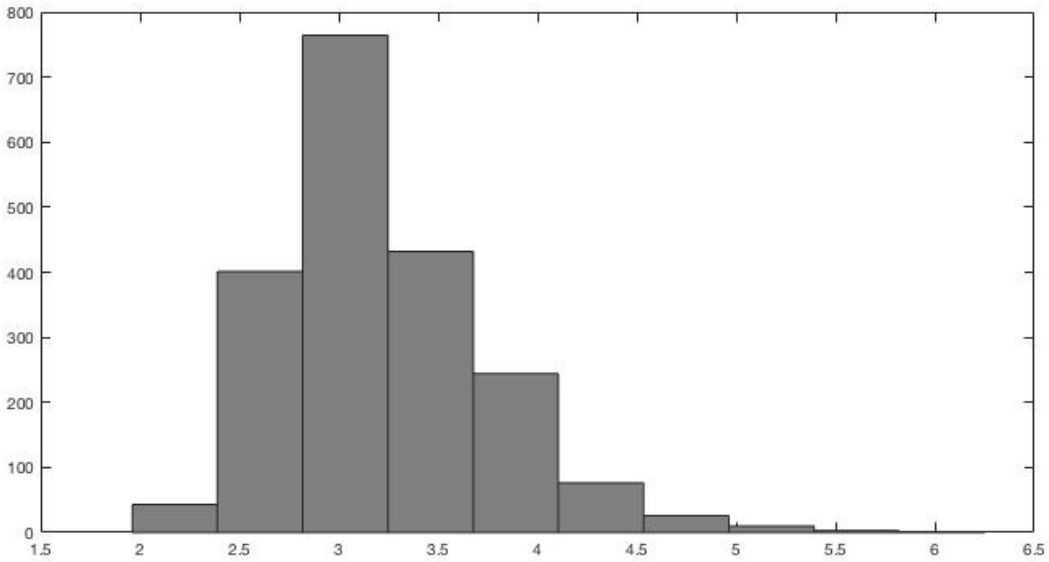

FIGURE 1.1: Histogram of results for estimation of π

In Figure 1.1, the histogram obtained by running code 1.2 is presented. This histogram, as described above, is an important indicator of the statistic reliability of the estimation of π. As outlined in the figure, this form of distribution presents the very typical bell of a Normal distribution, and we see that the distribution is concentrated around the true value $\pi = 3.14159265....$

1.2 History

As described in the previous paragraph, Montecarlo methods appeared for the first time already in the XVIII century, thanks to Buffon's contribution. However, the method employed for the estimation of π, while representing a Montecarlo approach, was extremely different from what today we call Montecarlo techniques. The reason is that at the heart of Montecarlo methods is the increased computation ability of modern calculators. It is not by chance that the first modern Montecarlo algorithm was implemented only when the first calculator was available, as will be clear in the following lines.

As said, the first embryonal form a Montecarlo technique was developed in 1777 by Buffon. After that, we need to wait for one century and a half before seeing the

implementation of a new, but related, algorithm belonging to the Montecarlo family. At the beginning of 1900, an English chemist, statistician and mathematician, William Sealy Gosset (1876-1937) found a way to estimate the distribution of the correlation coefficient and of the t-statistic. This technique was based on the simulation of random numbers and, thus, it is today numbered among the Montecarlo algorithms. Gosset used the penname of A. Student and he is the father of the Student-t distribution. A few years later, probably inspired by this work, a group of scientist will change the pace of development of statistical and simulation techniques forever, giving life to what today we call the Montecarlo method. While a statistician, Gosset, gave the first modern implementation of this method, we will have to wait several years before seeing Montecarlo techniques coming back to the field of statistical analysis. For half a century, all developments in this direction have focused on physical (and nuclear) applications. In any case, it is essential to underline that early developments of the Montecarlo Simulation, in the XVIII and XIX centuries, were significantly different from what we call today a Montecarlo technique. Not only calculators were not available, but, most importantly, the approach was substantially different. While today the essence of a Montecarlo technique is finding a solution to a deterministic model, by finding its probabilistic counterpart and solving it using simulations, first implementations focused only on deterministic problems.

The research program that led to the implementation of Montecarlo took place during the Second World War in Los Alamos, New Mexico, USA. Before that, in 1930, Enrico Fermi's (1901-1954) research in the field of controlled fission presented many common points with future developments of the Manhattan project. The United States' Government created a secret project, called Manhattan, intending to develop the technology required for the construction of the atomic bomb. The Head of the Manhattan project was Robert Oppenheimer (1904-1967), a well-known American physician. In 1943, Oppenheimer recruited Nicholas Constantine Metropolis (1915-1999), a young physician. He will be one of the leading personalities in the development of the Montecarlo method. In 1943, there was an urge by the Government, addressing the Manhattan project, to arrive at the development of the first atomic bomb. In this context, the possibility to generate random trajectories was thought to be very important to be able to estimate ex-ante the point in which a bomb would collapse, once thrown from an airplane. At the time, it was indeed a very ambitious goal since calculators were not yet available. At the first point, only traditional digital computation could be employed. This implied that hand calculators were manually used by technicians and, consequently, the power of simulations was significantly limited. It was for that reason that, in parallel to the programs aimed at creating the first atomic bomb, efforts in the direction of atomized calculators were made. Coding and programming were about to appear for the first time, changing the lives of humans forever. After the war, in 1948, it was clear that, in order to be able to implement a random number generator, it was first necessary to develop a digital computer, in the modern sense of the word. Metropolis took the

head of a team which created the MANIAC, Mathematical Analyzer, Numerical Integrator, and Computer. At that point, the team was ready to develop Montecarlo techniques, thanks to this new computing power. Two new important names appear in this context. The first one is Stanislaw Ulam (1909-1984), a Polish mathematician very active in the research that led to the construction of the first atomic bomb. The second one, John von Neumann (1903-1957), a Hungarian mathematician, physicist and computer scientist, gave essential contributions in many different fields of sciences. From mathematics to probability, from topology to economics, from quantum physics to game theory, from computer science to dynamics. Both the scientists, indeed, were part of the Manhattan project since its beginnings and took leading positions in the development of the atomic bomb. Following the issues left open by the development of the atomic bomb, these two personalities, together with Metropolis, had the intuition that, especially in physics, where a problem has a solution which is too difficult to find analytically, it may be possible to find it via simulation. This was a big novelty and a great achievement for almost every natural science. In 1949 Metropolis and Ulam published the paper that described for the first time Montecarlo methods. It is usually said that Metropolis chose the name as a reference to the city of Montecarlo, home of a great Casino, and, consequently a clear reference to the concept of randomness, the heart of the Montecarlo method. However, there is no agreement in this respect. According to another version, for example, the name comes from the fact that Ulam's uncle used to ask money to his family to go to bet at Montecarlo's Casino. It is important to note that in the Monte Carlo Method (1949), the technique was only applied to physics, and it was not even considered a statistical technique. It was only in 1953 when Metropolis and other academics published an article, Equations of State Calculations by Fast Computing Machines, that the process of simulating random numbers was formally and extensively described. This paper is considered the first rigorous description of Montecarlo techniques, and it still has a great value for modern statisticians. In 1958, W.F. Bauer published a paper in which he recognizes that, while Montecarlo techniques had an old and long history, it was only with the article by Metropolis and Ulam (1949) that the method was formalized. Bauer underlines the fact that Montecarlo methods require high frequency, automatized computers, able to generate large random samples. He even provides the information that, in 1958, a computer was able to generate 200 random numbers in one second[2]. From that date, there have been several developments in the field of statistical simulation, in general, and Montecarlo algorithms, in particular. Most of those developments were in the field of natural sciences, especially physics. However, by the end of the last century, the first academic papers related to the application of Montecarlo techniques to Economics were published. Following the continuous evolution and development of modern calculators and programs, Montecarlo techniques are constantly improved.

[2]Bauer, W. (1958). The Monte Carlo Method. Journal of the Society for Industrial and Applied Mathematics, 6(4), 438-451.

1.3 Applications

As described above, Montecarlo simulation's first application was in physical problems. Today, even if physics remains one of the most critical areas of application of Montecarlo methods, several other disciplines use Montecarlo tools for the solution of many different problems. The principal reason is that Montecarlo techniques are beneficial whenever a problem has no clear analytical solution. In all these cases, it should be possible to find an approximation of the solution by recurring to the simulation of random numbers. Of course, as will be specified in the following chapters, it is not necessarily that easy, in the fact that several conditions should be met in order to be able to apply the Montecarlo method and, most importantly, the reliability of the estimated solution should always be tested. In any case, Montecarlo techniques offer a convenient way to solve the problems mentioned above and, thus, are employed by many different disciplines. From physics to psychology, from biology to economics, from thermo-dynamics to medicine. In general, Montecarlo methods are highly appreciated by natural scientists. In this paragraph, the main applications of Montecarlo techniques will be presented, following the work of Metropolis and Ulam(1949) and Bauer(1958).

One of the first applications of the method is to combinatorial analysis, a common interest for both mathematics and applied sciences. The need to use a technique such as Montecarlo comes from the interconnections between mathematics and probability theory. In any combinatorial problem, it is necessary to know the probability of an event and, in some cases, it may not be handy to estimate it. Montecarlo methods provide a solution to those kinds of problems because they allow the scholar to perform a simulation of the event several times and to estimate the probability of the event as the ratio between the number of successes and the number of trials, not differently from what we did in Section 1.1 with the Buffon's needle experiment. Of course, the result will not be the true exact probability, but, as the number of trials increases, it is possible to prove (mainly based on the Law of Large Numbers) that Montecarlo simulations provide a good approximation.

Another possible application is connected to the quantification of the volume of a high dimensional region. This problem is of great interest for all natural sciences. If we are looking for the volume of a region defined over n dimensions, where n is sufficiently large, say $n > 5$, it may not be handy to evaluate multiple integrals. In order to solve the problem, we may apply a Montecarlo method based on the selection of random points and the analysis of the position of those random points. Specifically, the number of points that falls within the relevant region should be counted. Again, the estimation of the volume that follows should converge in probability to the actual volume.

A further area of interest is connected to cosmic rays and matrixial algebra. In some problems related to this topic, particles full of energy tend to create cascade

events until the point in which available energy falls below a certain threshold. Scientists are interested in the outcome of this process, called Markoff Chain, which is especially difficult to model analytically. A solution can be found only using matrixial algebra, but the order of matrices is usually huge. It is then possible and useful to apply a Montecarlo method, based on random experiments that describe possible future evolutions of the system.

Another well-known application of Montecarlo methods focuses again on particle physics and it is at the heart of Metropolis and Ulam's paper (1949). Many physical problems start from the assumption that there is a surface in which particles are able to reproduce. Such a problem can be represented in a simple way by the equation $\frac{du(x,y,z)}{dt} = a(x,y,z)\delta u + b(x,y,z)u(x,y,z)$. Analytical solutions to this equation are especially difficult to find. For this reason, Metropolis and Ulam proposed a Montecarlo approach. This method is based on the simulation of random behaviors for all those variables that are considered independent by the scholar. All other variables are determined based on the random behavior of the independent variables, according to a pre-specified equation. Consequently, the method is simply based on the iteration for n times of the same procedure. First, for all stochastic variables, a random path is generated. Second, for all deterministic variables, the value is computed according to the equation, where the inputs are the outputs of the simulation in the previous phase. In this way, it is not necessary to evaluate the behavior of the system analytically, but it is only necessary to make statistical inference on the random values that were generated.

It is evident from the above description that one of the most important strengths of Montecarlo methods is that it allows evaluating integrals that are difficult or impossible to be dealt with analytically. Then, it is worth spending some time on the application of the method for the computation of integrals. Suppose you want to evaluate the definite integral $\int_a^b f(x)dx$. It is well known that the graphical representation of this problem is the area under the curve of $f(x)$ on the interval $[a,b]$, we call this area C. Define the line $y = M$ s.t $M > f(x)\ \forall x \in (a,b)$. The area of the rectangle with basis (a,b) and height M is nothing but $M(b-a)$. Then, it is evident from Figure 1.2 that $C < M(b-a)$. In general, if $f(x)$ is not especially complex, it should be possible to compute this integral analytically, in order to get the exact result. However, it may not be possible to find a solution in closed form to the integral of $f(x)$. In all these cases, numerical techniques in general, and Montecarlo techniques in particular can provide useful tools in order to find an approximation of the relevant integral. In this instance, Montecarlo techniques are based on the random selection of points in the rectangle of area $M(b-a)$. If, for every trial, the point lies inside the region C, a success is recorded. Then, we formally define the random variable of successes, $S = 1$ if $s \in C$, $S = 0$ otherwise. The random variable in which we are interested in is the ratio of the number of successes to the number of trials, $N : \frac{S}{N}$. The expected value of this random variable, $E(\frac{S}{N}) = \frac{C}{M(b-a)}$. If we perform N trials, the expected value of S is $E(S) = \frac{NC}{M(b-a)}$ and the variance is

$\sigma^2 = \frac{NC}{M(b-a)}(1 - \frac{C}{M(b-a)})$. Then, for the variable of interest $\frac{S}{N}$, the standard deviation is only $\sigma' = \frac{\sigma}{\sqrt{N}}$. This variable is extremely important for general considerations on Montecarlo methods. It is possible to see from the standard deviation, in fact, that the accuracy of estimation increases with the number of trials. It is important

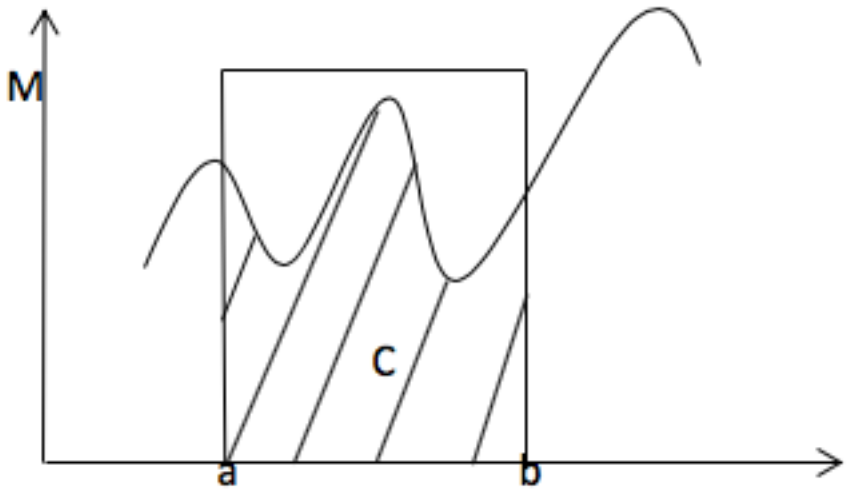

FIGURE 1.2: Graphical representation of an integral

to note that, when we are dealing with one-dimensional integrals, Montecarlo techniques will hardly be needed. At the same time, they are essential for the solution of multi-dimensional integrals. Even if we are dealing with high-dimensional integrals, however, the idea of the algorithm remains that explained in the above lines. The theoretical finding at the heart of the application of this method is the mean-value theorem. Suppose, for simplicity but without loss of generality, to have a two-dimensional integral in the form $\int_\omega f(x,y)dxdy$. This integral is nothing but the surface of the function $f(x,y)$ lying in the area ω. We call this area $A(\omega)$. Montecarlo algorithm described in the following code performs what has been described in words in the above lines. Namely, it defines the rectangular area R and draws random points in R. Then, the ratio of success to trials is computed. Based on this ratio, an approximation of the integral is returned.

LISTING 1.3: Monte Carlo double

```
% Source: Stochastic Simulation and Applications in Finance
%with MATLAB Programs
%By Huu Tue Huynh, Van Son Lai, Issouf Soumare
function result = MonteCarlo_double(f, g, x0, x1, y0, y1, n)
%
% Monte Carlo integration of f over a domain g>=0, embedded
```

```
% in a rectangle [x0,x1]x[y0,y1]. n^2 is the number of
% random points.

% Draw n^2 random points in the rectangle
x = x0 + (x1 - x0)*rand(n,1);
y = y0 + (y1 - y0)*rand(n,1);
% Compute sum of f values inside the integration domain
f_mean = 0;
num_inside = 0; % number of x,y points inside domain (g>=0)
for i = 1:length(x)
for j = 1:length(y)
if g(x(i), y(j)) >= 0
num_inside = num_inside + 1;
f_mean = f_mean + f(x(i), y(j));
end
end
end
f_mean = f_mean/num_inside;
area = num_inside/(n^2)*(x1 - x0)*(y1 - y0);
result = area*f_mean;
end
```

1.4 The need for simulation techniques

As pointed out in the previous paragraph, scholars in almost every discipline have always been challenged to find solutions to problems that are especially difficult to solve analytically. Consider a real-world phenomenon; it may be the relationship between interest rates and unemployment or the one among atomic particles. We call this phenomena system of interest. A scholar aims to understand how this system works in reality. Since the endogenous forces able to modify and alterate the state of the system may be many, it is necessary to represent the system via a model, which is a (mathematical) simplified and stylized representation of reality. Of course, once a model is available, the most crucial goal is to solve it. A deterministic model is a model in which no form of randomness is involved. This implies that, given the initial point or initial exogenous variables, the output of the system will always be the same. When it is possible to find the solution of a deterministic system, there is no need to refer to probability theory and stochastic calculus. When the solution of a deterministic system is found we are in the best possible scenario, since we are sure that the equilibrium condition will last as long as the exogenous variables remain unchanged. Unfortunately, two situations may materialize. First, it may not be possible to find a deterministic model that genuinely describes the system of interest. Second, even if a deterministic model was defined, it may not be possible to find its solution. In both cases, an efficient solution can be the definition of a probabilistic

(or stochastic) model that corresponds to the same system of interest. A probabilistic model is a model in which randomness plays a role and, thus, the relationship between the variables of interest cannot be expressed exactly. Once the probabilistic model has been defined, it should usually be possible to combine Probability Theory with mathematical tools in order to find the solution.

However, in many fields of studies, there are problems for which it is not possible to find a solution analytically. In all these cases, stochastic simulation can be used as a tool to deeply understand the behavior of the real world system. In general terms, given a stochastic differential equation, it should always be possible to solve it with deterministic techniques, as long as the dimension of the state space is sufficiently small. When this is the case, it is convenient to use deterministic techniques rather than numerical methods since the result that will be obtained is expected to be more precise. However, when the dimension of the state space increases, the complexity of the problem follows and numerical techniques may be the only way to find at least an approximation of the solution. Following the integral example in the previous section, consider an integrable function f over the domain $[0,1]^d$. We want to find a solution to the integral $I = \int_{[0,1]^d} f(x_1, \ldots, x_d) dx_1, \ldots, dx_d$. A typical deterministic method for multi-variate integrals is the Quadrature method. If this technique works, it should be considered as an optimal alternative since it allows finding a precise solution. However, this method is sensitive to the smoothness of the function f. For this reason, it may be necessary to apply a numerical technique based on simulation. As specified above, in order to apply a simulation method, it is necessary to transform a deterministic model in the probabilistic correspondent representation. In this case $I = E(f(U_1, \ldots, U_d))$., where U_i are i.i.d uniform random variables in the interval $[0,1]$. By the Strong Law of Large Numbers, it follows that it is possible to approximate I by $\frac{1}{N \sum_{i=1}^{N} f(U_1^l, \ldots, U_d^l)}$. This result represents the foundation of every Montecarlo technique, given that we are approximating a deterministic quantity by averages of random values. Of course, the "goodness" of the accuracy increases with N, which represents the number of trials.

Chapter 2

Generation of uniform random numbers

In order to be able to perform every kind of Montecarlo simulation, it is necessary to be able to generate random numbers. When we generate random numbers, we are basically performing a simulation exercise and, usually, we impose that the generated random numbers obey to a pre-specified statistical distribution. Montecarlo simulations are usually associated with every kind of technique that uses the generation of random numbers in order to find the solution to a deterministic problem. This is the reason why a considerable part of this dissertation is devoted to random number generators. In the end, Montecarlo simulations or methods can easily be described as methods aimed at the identification of the solution of a deterministic problem, based on simulations, i.e., on the generation of several random trajectories.

Almost all random number generation functions that are pre-built in modern software give as a result a stream of numbers distributed according to the Uniform Distribution in the interval $(0,1)$. It is important to stress that a uniform random number generator should be able to produce random numbers distributed according to a Uniform distribution in $(0,1)$, that is, excluding both 0 and 1. A random variable distributed according to the uniform distribution is characterized by a probability distribution as $p(x) = 1$ if $0 < x < 1$ and $p(x) = 0$ otherwise. Since the Uniform is the most basic statistical distribution available and since all other random number generators start from the algorithm employed for the Uniform, it makes sense to start our description by uniform random number generators.

In abstract terms, a uniform random number generator is an algorithm able to produce a series of i.i.d Uniform random variables in the interval $(0,1)$, $u_1, u_2, \ldots \sim U(0,1)$. A sequence $(u_n)_{n\geq 1}$ of $(0,1)$valued real numbers is a sequence of random numbers if there exists a probability space $(\Omega, \mathcal{F}, P)$, a sequence $U_n, n \geq 1$ of i.i.d. uniform random variables and $\omega \in \Omega$ such that $u_n = U_n(\omega) \forall n \geq 1$. In words, a process (or sequence of numbers) is said to be random if the conditional probability of the next event, given the previous history, is equal to the unconditional probability [1]. Additionally, a random number generator should be able to generate a sequence

[1] Gentle, J.E. 2004. Random Number Generation and Monte Carlo Methods. Springer.

$u_1, ..., u_n$ of random variables that are not only uniformly distributed in the interval $(0, 1)$, but also all mutually independent [2].

This definition already points out one of the principal problems that come with simulations: the problem of quasi-randomness. Computers are, in fact, not able to generate i.i.d. sequences of truly random variables, whatever the distribution. For this reason, scholars and practitioners usually employ simulation techniques anyway, well aware of this limitation, and then, once a sequence of random variables has been generated, perform some statistical tests in order to verify the goodness of fit of the generated distribution. If the results of the test can be considered satisfying, then the sequence can be used for the analysis. The problem stands from the fact that computer software generates random numbers starting from some deterministic algorithm. If the algorithm is deterministic, then it is not possible to consider the generated sequence as truly random, but only quasi-random. In the end, what is used in simulation exercises is a sequence of i.i.d. (pseudo)random variables whose properties are indistinguishable from those of a true i.i.d. sequence of random variables. As a consequence, all modern algorithms provide as a result an i.i.d. sequence of (pseudo)random numbers.

As stated above, the default option is the generation of random numbers from a uniform distribution. In programs such as MATLAB those algorithms run in background. This implies that if the user wants to generate a sequence of i.i.d. Uniform random variables, he or she does not need to understand how these (pseudo)random numbers are generated, since it suffices to run the command rand(n). However, it is very instructive to understand how these algorithms work because, in this way, it is possible to explain where the problem of quasi-randomness comes from and, consequently, the limitations of the simulation approach. For this reason, we now introduce the main algorithms employed by calculators in order to generate random numbers. Most of those algorithms are based on modular arithmetic.

2.1 Common considerations

Generators of uniform random variables simulate realizations of a uniformly distributed random variable in the interval $(0, 1)$. We focus on the generation of uniformly distributed random numbers for two reasons. First, if we can generate uniformly distributed random numbers, it is possible to apply transformation methods, so to obtain random numbers from other distributions. Second, it is useful to start from the Uniform distribution due to its simplicity, avoiding analytical complications of the other distributions, which may harm deep understanding. Every generic uniform random number generator is an algorithm able to produce a stream $U_1, ..., U_n$ of uniformly distributed random variables in the interval $(0, 1)$, all mutually independent. Independence is a relevant argument and it is closely linked to

[2] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

the concept of predictability. If all pairs in the sequence $U_1, ..., U_n$ are independent this implies that it should not be possible to predict any U_t by simply using all past information, i.e. $U_1, ..., U_{t-1}$. This implies that the algorithm [3] can generate a stream of unpredictable numbers[4].

2.1.1 Generic recursive generators

Most of the generators that are usually employed are based on multiple recursion. Multiple recursion implies that previous numbers are determinants of the following one, but in a fashion that appears to be random. A generic recursive generator is in the form:

$$x_t = f(x_{t-1}, ..., x_{t-k}) \tag{2.1}$$

f is the generation law. It may be an elementary function, as in the case of linear generators, or a more complex one, as in the case of mixed generators. k is the order of the generator and denotes how far we go in the past before generating the current number. In order to implement such a generator, it is necessary to set the initial value, x_0, which is called seed. The choice of the seed is crucial in the fact that it has a significant impact on the period of the sequence. By period we mean the length of the sequence of different numbers. At some point, in fact, the generated numbers will start repeating exactly in the same sequence. Of course, it is convenient to choose the parameters so that the generator has the longest possible period. If we take the same law or function f, the same order k and the same seed x_0, then the sequence generated by the software will always be the same. We are employing a deterministic algorithm. Recursive generators usually give as output a sequence of pseudo-random integers, which are then scaled in order to get a sequence of uniformly distributed pseudo-random numbers in the interval $(0, 1)$. The set of integers generated is denoted by $\mathbf{I}$.

2.1.2 Modular arithmetics

Since all recursive generators are based on modular arithmetics, it is convenient to introduce it briefly. Given a variable x and a modulus m, we call modulo of x with modulus m, $xmodulo(m)$, the remainder of x when divided by m. The modulus m must be strictly positive. For example, set $x = 4$ and $m = 2$, then $4modulo(2) = 0$. If $x = 5$ and $m = 2$, then $5modulo(2) = 1$. Given two real numbers $a, b \in \mathbb{R}$, they are said to be congruent $modulo(m)$ if their difference is an integer divisible by m: $a \equiv bmodulo(m)$. This congruence relation is symmetric ($a \equiv bmodulo(m)$ implies $b \equiv amodulo(m)$), reflexive ($a \equiv amodulo(m)\ \forall a$) and transitive ($a \equiv bmodulo(m)$ and $b \equiv cmodulo(m)$ implies $a \equiv cmodulo(m)$) [5]. Modular arithmetics is a very useful

[3] Gentle, J.E. 2004. Random Number Generation and Monte Carlo Methods. Springer.

[4] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

[5] Gentle, J.E. 2004. Random Number Generation and Monte Carlo Methods. Springer.

tool also when it comes to random variables. Let $U(0,1)$ be a uniformly distributed random variable in the interval $(0,1)$. Define the random variable X as $X = (aU + b)modulo(1)$, where $b \in \mathbb{R}$ and a is an integer $\neq 0$, then $X \sim U(0,1)$.

2.2 Linear congruential generators

2.2.1 Definition

The Linear congruential generator (LCG) is a widely used algorithm for the generation of random numbers. It was first proposed by Lehmer in 1951 and it is based on modular arithmetic. This generator is usually not the best solution to be employed, but it is on the basis of other, more complex and efficient, generators. The implementation of this algorithm is fairly simple since it is based on a basic linear function. Define $x_t \in [0, \ldots, m-1]$ as

$$x_t = (ax_{t-1} + b)modulo(m), t = 1, 2, \ldots. \tag{2.2}$$

At any point in time, then, the value depends solely on the period immediately before. As said before, the modulus m must be a real strictly positive integer, while a, the coefficient or multiplier, and b, the increment, are integers $< m$. In order to implement this function on software it is enough to write a simple loop, choosing a, b, m and the seed, i.e., the initial state x_0, which must be positive and lower than m. The relation in 2.2 is called congruence relation and will always generate non-negative numbers, integers and lower than m. If the goal is to generate random numbers in the interval $(0,1)$, then it is enough to apply the transformation $u_t = \frac{x_t}{m}, \forall t$. The result will be a stream of pseudo-random numbers $u_i \in [0, \frac{(m-1)}{m}], \forall i$. If we set $b = 0$, then we usually call the algorithm Multiplicative congruential generator. It is in the form:

$$x_t = (ax_{t-1})modulo(m), t = 1, 2, \ldots. \tag{2.3}$$

Each sequence of random numbers generated according to 2.2 can contain at most m different values. According to this statement, it is easily concluded that the greater m is chosen, the best the random sequence should be. However, it is not possible to draw this straightforward conclusion since, as m increases, the costs (in time and required computing power) increase too. In addition, the interaction among the parameters a, b, m plays a crucial role in the determination of the efficiency of the generator. For these reasons, it is necessary to investigate the properties of linear congruential generators and optimal parameter choices.

2.2.2 Properties and parameter choice

Depending on the parameter choice, i.e., x_0, a, b, m, at some point the sequence of numbers will necessarily repeat. Specifically, once a number in the sequence repeats all the sequence will repeat itself in the same form. This is, of course, a source of

concern and it is useful to introduce a first consideration when it comes to the choice of parameters in linear congruential generators: period length. Given any random number generator, the maximum period is the maximum length of a sequence with non-repeating numbers. Of course, the goal is to obtain a generator with the longest period length. The maximum period of a generator critically depends on the choice of the modulus m since the maximum period of a linear congruential generator is $m-1$ (in this case we say that the generator has full period or full-cycle [6]). Unfortunately, the solution is not as simple as intuition may suggest, i.e., choosing a very large modulus m, since the interaction with the other parameters x_0, a, b should be taken into account too. Let m be set, then it is possible to investigate for which values of x_0, a, b the maximum period is reached. The following theorem establishes an important result for LCGs [7].

A linear congruential generator in the form 2.2 has full period $m-1$, if b is relatively prime to m (i.e. their only common divisor is one), $a = 1 modulo(p)$ if p is a prime factor of m, and $a = 1 modulo(4)$ if 4 is a factor of m.

A first consequence of this theorem is that if m is a power of 2, as it usually is in modern computers, it is sufficient to use a b which is odd and $a = 1 modulo(4)$. In the case of a multiplicative congruential generator ($b = 0$), with m prime, full period $m-1$ is reached if $a^{m-1} - 1$ is a multiple of m and $a^j - 1$ is not a multiple of m $\forall j \in [1, ..., m-2]$. Given the simplicity of the conditions connected to the multiplicative case, MCGs are more common than LCGs.

While period length is undoubtedly a big concern when choosing a generator, there are still several other considerations [8]. First, since in Montecarlo simulations several generations of random numbers are required, the generator must be fast enough. Second, the generator must be able to produce a sequence of numbers that are at least close to a uniform distribution. It is challenging to evaluate an algorithm in this respect. In the sense that it is not possible to define whether the generated sequence will be close to the uniform distribution simply based on the wording of the algorithm. For this reason, practitioners usually employ goodness-of-fit and independence tests on the generated random numbers [9]. Finally, portability is a very relevant consideration. Scholars and practitioners tend to use different software packages when running their work. Nevertheless, the same generator (given the same seed) must produce the same stream of random numbers, whatever the software used.

Following these considerations, many couples of multipliers and moduli were proposed in literature. No one is superior to the other with respect to all properties and, thus, the choice should depend on the specific simulation to be performed. The

[6] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

[7] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

[8] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

[9] Those tests are described at the end of this chapter.

following table [10] summarises the most employed generators. They are all multiplicative since b is usually set to be 0.

TABLE 2.1: Multiplicative congruential generators

m	a	Author(s)
$2^{31}-1$	16807	Lewis et al. (1979)
$2^{31}-1$	39373	L'Ecuyer (1988)
$2^{31}-1$	742938285	Fishman and Moore (1986)
$2^{31}-1$	950706376	Fishman and Moore (1986)
$2^{31}-1$	1226874159	Fishman and Moore (1986)
$2^{31}-1$	630360016	Payne, Rabung and Bogyo (1969)
2147483399	40692	L'Ecuyer (1988)
2147483563	40014	L'Ecuyer (1988)

It is not by chance that the modulus for most of these generators is $2^{31}-1$. Computers today, in fact, usually have word length of 32 bits and thus the maximum number that can be defined is exactly $2^{31}-1$.

2.2.3 MATLAB implementation

Implementing a LCG on Matlab is fairly simple since it is only necessary to set up a single loop. However, it is important to note that most packages, including Matlab, usually come with pre-built functions able to generate random numbers distributed according to the Uniform. In Matlab such a command is rand(n). In the following explanation we will keep for simplicity m small in order of magnitude. However, consideration related to full-cycle are still valid, as explained in section 2.2.2.

Let's set $m = 32, a = 5, b = 3, x_0 = 11$, the code that generates random numbers is the following:

LISTING 2.1: LCG

```
1
%% LCG
3
clc
5 clear all
n=100
7 m=32;
a=5;
9 b=3;
```

[10] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

```
x_t=11*ones(n);

for i=2:n
x_t(i)=mod(a*x_t(i-1)+b,m)
end

x_t=x_t(:,1)
```

By running this code we always obtain the same stream of numbers 26,5,28,15,14,9,16, 19,2,13,4,23,22,17,24,27,10,21,12,31,30,25,0,3,18,29,20,7,6,1,8,11. This is indeed a very efficient choice because a full cycle of modulus m is reached. This means that, given a, b, m, x_0, the algorithm is able to generate $m = 32$ different values. As stated above, it is not easy to find a combination of parameters that is able to generate a full cycle. For example, if we set $a = 6, b = 11, m = 32, x_0 = 10$, according to the following code, the sequence generated is 10,7,21,9,1,17 and then the stream is repeated. This implies that a full cycle has not been reached.

LISTING 2.2: LCG non full

```
%% LCG non full

clc
clear all

n=100
m=32;
a=6;
b=11;

x_t=10*ones(n);

for i=2:n
x_t(i)=mod(a*x_t(i-1)+b,m)
end

x_t=x_t(:,1)
```

Another famous choice is due to Lewis, Goodman and Miller (1969), $a = 7^5, b = 0, m = 2^{31} - 1 = 21474836474$.

LISTING 2.3: LCG LGM

```
%% LCG LGM

clc
```

```
clear all

n=100
m=21474836474;
a=7^5;
b=0;

x_t=ones(n);

for i=2:n
x_t(i)=mod(a*x_t(i-1),m)
end
x_t=x_t(:,1)
```

As specified in 2.2.2. the pseudo-random numbers that are generated by those algorithms are all non-negative integers. If the aim is to obtain random numbers drawn from the uniform distribution then it is enough to apply the transformation $u_t = \frac{x_t}{m}, \forall t$. The result will be a stream of pseudo-random numbers $u_i \in [0, \frac{(m-1)}{m}]$.

In the example of the full cycle LCG it is only necessary to add a line, so to apply this transformation. It is also possible to histogram the results so to have a first flavor of the closeness of the generated distribution to the Uniform. Note that since $m = 32$ is small, even if the generator has full cycle, the stream will not be sufficiently close to a uniform distribution. This because, as specified above, as m increases (ceteris paribus) the goodness of the generator increases too.

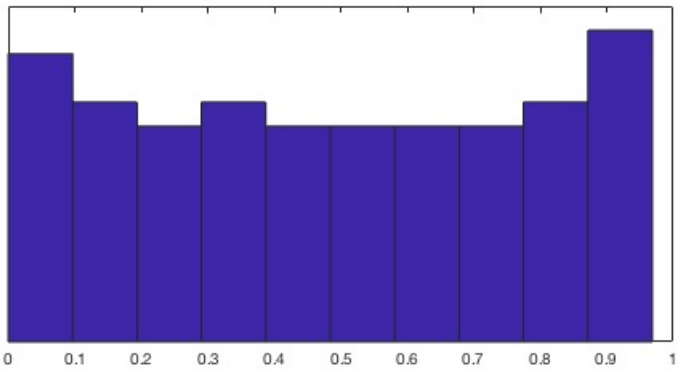

FIGURE 2.1: LCG uniform histogram

LISTING 2.4: LCG with histogram

```
%% LCG

clc
clear all
n=100
```

```
m=32;
a=5;
b=3;

x_t=11*ones(n);

for i=2:n
x_t(i)=mod(a*x_t(i-1)+b,m)
end

x_t=x_t(:,1)

u_t=x_t/m

hist(u_t)
```

Indeed, it is also possible to apply this algorithm to directly generate uniformly distributed random numbers, with no need to apply the above-mentioned transformation, with $u_t \in (0,1) \forall t$. However, this usually not considered the most efficient way of generating random numbers:

$$u_t = (au_{t-1}) modulo(1), t = 1, 2, \ldots. \tag{2.4}$$

LISTING 2.5: LCG uniform

```
%% LCG Uniform

clc
clear all
n=100
m=1;
a=7;

u_t=0.2*ones(n);

for i=2:n
u_t(i)=mod(a*u_t(i-1),m)
end

u_t=u_t(:,1)

hist(u_t)
```

Indeed, this is immediately evident if we look at the histogram of the generated distribution. It appears to be far from a uniform when $n = 100$. As usual, those inconsistencies decrease as we increase the number of trials n.

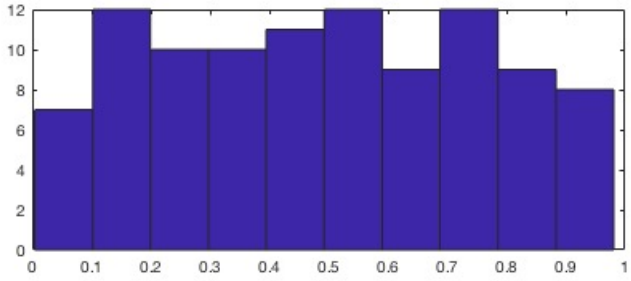

FIGURE 2.2: Direct generation of a uniform distribution n=100

2.2.4 Add with carry and Multiply with Carry

In 1991, Marsaglia and Zeman proposed an innovative kind of linear congruential generator, called add-with-carry. This generator is in the form:

$$x_t = (x_{t-s} + x_{t-r} + b_t) modulo(m) \tag{2.5}$$

$b_1 = 0, b_{t+1} = 0$ if $x_{t-s} + x_{t-r} + b_t < m$, $b_{t+1} = 0$ otherwise. This generator turns out to be very efficient since, for some choices of the parameters, it can reach the period of 10^{43}. The multiply with carry is a simplification of this generator and it is in the form:

$$x_t = (ax_{t-1} + b_t) modulo(m). \tag{2.6}$$

where b_t is such that $b_t = \lfloor (ax_{t-k} + b_{t-1}/m) \rfloor$ for some $t \geq k$.

2.3 Multiple recursive generators

Linear congruential generators are widely used because they employ a simple linear function and a modular reduction. However, as specified at the beginning of this chapter, the function f can well be any function. One natural evolution of LCGs comes then from the intuition of combining several linear functions with modulo reduction. Generators in this form are usually called Multiple recursive generators (MRG). The idea is that at each point in time the random value depends not only on the value in the last period, but on the last $t - k$ periods. MRGs are usually identified with their order, k. A generic MRG is in the form:

$$x_t = (a_1 x_{t-1} + a_2 x_{t-2} + \cdots + a_k x_{t-k}) modulo(m), t = k, k+1, \ldots \tag{2.7}$$

In this case we are then missing the increment b and the multipliers are $a_i, i = 1, \ldots, k$ with $a_i \in [0, \ldots, m-1]$. Following the definition of this generator, the seed is not a scalar but a vector of dimension k, where k, the order, is a choice of the user. A simple multiple recursive generator can be implemented with codes like the one below.

LISTING 2.6: MCG

```
%% MCG

clc
clear all

m=32;
a_1=5;
a_2=6;

x_t=11*ones(m);

for i=3:m
x_t(i)=mod(a_1*x_t(i-1)+a_2*x_t(i-2),m)
end
%uniformly distributed random numbers
u_t=x_t/m
```

Note that these generators are not so common in our days, because, compared to the other available algorithms, they tend to be quite slow. In order to get a fast generator, it is in fact necessary to set most of the multipliers $a_i = 0, 1, -1$. Efficiency considerations led many scholars to analyze for which values of multipliers and modulo the maximum period is reached. Several and all valid are the proposals. One very famous example was provided by L'Ecuyer, Blouin and Couture (1993). They propose a MRG $k = 5$ with $a_1 = 107374182$, $a_2, a_3, a_4 = 0$ and $a_5 = 104480$. The modulo is the typical $2^{31} - 1$. This generator has been proven to perform well. However, LCGs are still preferred in most simulation exercises. Another well-known choice was proposed by Deng and Ling in 2000. They employ a flexible generator in which $a_1 = 1$, $a_k \neq 0, 1$ and $a_{2,\dots,k-1} = 0$. One final consideration is that, as in the case of LCG, MRG algorithms generate non-negative integers. If instead, we are interested in the generation of random numbers from a uniform distribution, it is sufficient to apply the following transformation, $u_t = \frac{x_t}{m}, \forall t$. Analogously, the code is easily modified accordingly.

2.4 Matrix congruential generators

Linear and multiple recursive generators are indeed feasible if the aim is to generate more than a sequence of random numbers in parallel. It is possible in fact to represent those generators in a matricial form. Given a modulus m and an invertible kxk matrix $\boldsymbol{A}$, a MRG can be written in the form:

$$\boldsymbol{x_t} = (\boldsymbol{A}x_{t-1} + b)modulo(m), t = 1, 2, \dots \tag{2.8}$$

b and x_{t-1} are $kx1$ vectors, all elements of those vectors are $\in [1, m-1]$. However, as in the case of LCGs, b is often chosen to be the zero vector. The output of this algorithm is $\boldsymbol{x_t}$, which is a $kx1$ vector $\in [1, m-1]$. Again, it is possible to generate random numbers uniformly distributed in the interval $(0, 1)$. In order to obtain an efficient matrcial generator, it is critical the choice of the matrix $\boldsymbol{A}$. The maximum possible period is $m^k - 1$. In general terms, $\boldsymbol{A}$ is often chosen to be a sparse matrix, with many elements equal to $0, 1, -1$. A famous example was given by Deng and Ling (2000) that proposed a generator with $b = 0$ and $\boldsymbol{A}$ defined as

$$\begin{bmatrix} a_1 & -1 & 0 & \dots & 0 \\ 0 & a_2 & -1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -1 \\ -1 & 0 & 0 & \dots & a_k \end{bmatrix}$$

As linear congruential generators can be modified in multiple recursive generators in order to introduce dependence on multiple periods, so it is possible in case of matrix congruential generators:

$$\boldsymbol{x_t} = (\boldsymbol{A_1} x_{t-1} + \dots + \boldsymbol{A_j} x_{t-j}) modulo(m), t = 1, 2, \dots \tag{2.9}$$

LISTING 2.7: MCG Matrix

```
%% MCG matrix

clc
clear all

m=32;
A=[7,-1,0,0
0,9,-1,0
0,0,4,-1
-1,0,0,6];
det(A) % check that A is invertible

x_t=11*ones(m);

for i=2:m
x_t(i)=mod(x_t(i-1),m)
end
%uniformly distributed random numbers
u_t=x_t/m
```

2.5 Inversive congruential generators

Inversive congruential generators were proposed for the first time by Echenauer and Lehn in 1986 and are generally considered a complex family of generators since they employ non-linear functions. Consequently, given their computational complexity, they are not widely used. Inversive generators make use of the multiplicative inverse function x^-. The multiplicative inverse function of x modulo m is defined as $1 \equiv x^- x modulo(m)$. This function is defined only for all non zero x relatively prime to m. The inversive congruential generator is then defined as:

$$x_t = (ax_{t-1}^- + b) modulo(m_1), t = 1, 2, ... \tag{2.10}$$

where $x_t \in [0, m-1]$.

2.6 Mixed generators

As outlined above, literature has proposed many different random number generators. Each of them comes with strengths and weaknesses. For this reason, a huge body of research focuses on the implementation of combined generators able to overcome the remaining open issues. The basic framework is that same categories of generators are run in parallel and then combined to produce only one vector of uniform random numbers. This innovative technique allows to maximize the length of the cycle of generators and, in general, to improve their efficiency. This result, of course, comes with an increased level of complexity.

In general, mixed generators are considered superior both to LCGs and to MRGs. They are in fact usually associated with longer periods. However, long periods are not necessarily an advantage, since excessive length may harm computability. Consequently, it is not possible to define whether to prefer a very long or very short period length. If this is the case, then it is necessary to base the selection of the best random number generator on different considerations. For example, Greenberg (1962) underlines the importance of the concept of serial correlation when choosing among different generators. The basic result is that, since serial correlation of generated numbers depends on the parameters, it makes sense to choose the parameters so to minimize serial first-order autocorrelation. The (first-order) autocorrelation coefficient, ρ, can be expressed as:

$$\rho = \frac{1}{a} - \frac{6b}{am}(1 - \frac{c}{m}) + K.$$

The main finding of Greenberg's work is that very small and large values of a are to be avoided and, specifically, a good choice of a is close to $\sqrt{m}$ since this reduces ρ, irrespective of b.

2.6.1 Wichmann Hill generator

One of the first examples of mixed generators was proposed in 1982 by Wichman and Hill. This generator uses three LCGs and then combines them to generate a single stream of random numbers. The three LCGs are defined as follows:

$$\begin{aligned} x_t &= a_1 x_{t-1} modulo(m_1), \\ y_t &= a_2 y_{t-1} modulo(m_2), \\ z_t &= a_3 z_{t-1} modulo(m_3). \end{aligned} \tag{2.11}$$

Where $a_1 = 171, a_2 = 172, a_3 = 170$ and $m_1 = 30269, m_2 = 30307, m_3 = 30323$. Once this triplet has been generated, it is transformed in only one sequence of uniformly distributed in the interval $(0, 1)$ random numbers by applying the following LCG:

$$U_t = \frac{x_t}{m_1} + \frac{y_t}{m_2} + \frac{z_t}{m_3} modulo(1). \tag{2.12}$$

In order to implement this kind of generator, the following code can be run:

LISTING 2.8: W-H

```
%% WH

clc
clear all

m_1=30269;
a_1=171;
m_2=30307;
a_2=172;
m_3=30323;
a_3=170;

x_t=11*ones(100);
y_t=11*ones(100);
z_t=11*ones(100);



for i=2:m_1
x_t(i)=mod(a_1*x_t(i-1),m_1)
end

for i=2:m_2
y_t(i)=mod(a_2*y_t(i-1),m_2)
end

for i=2:m_3
```

```
29 z_t(i)=mod(a_3*z_t(i-1),m_3)
   end
31
   u_t=mod(x_t./m_1+y_t./m_2+z_t./m_3,1)
```

The Wichman Hill algorithm is able to achieve a very large period (10^{32}), and thus, thanks also to its computational simplicity, it is today widely employed in simulation exercises.

2.6.2 L'Ecuyer mixed generator

In 1988 L'Ecuyer proposed a new mixed generator, derived from the combination of three linear congruential generators:

$$\begin{aligned} x_t &= 40014 x_{t-1} modulo(2147483563), \\ y_t &= 40692 y_{t-1} modulo(2147483399), \\ z_t &= (x_t - y_t) modulo(2147483563). \end{aligned} \tag{2.13}$$

The stream of uniformly distributed random variables is then obtained by:

$$u_t = 4.656613 z_t 10^{-10} \tag{2.14}$$

2.6.3 Other mixed generators

It is also possible to combine MRGs. One example is the generator called MRG32k3a, proposed by L'Ecuyer. This generator combines two MRGs of order 3. The MRGs are in the form:

$$\begin{aligned} x_t &= (a_2 x_{t-2} - a_3 x_{t-3}) modulo(m_1), \\ y_t &= (b_2 y_{t-2} - b_3 y_{t-3}) modulo(m_2) \end{aligned} \tag{2.15}$$

Where $a_2 = 1403580, a_3 = 810728, b_2 = 527612, b_3 = 1370589$ and $m_1 = 2^{32} - 209, m_2 = 2^{32} - 22853$. Then, the two generators are combined via:

$$u_t = \begin{cases} \frac{x_t - y_t + m_1}{m_1 + 1} if x_t \leq y_t \\ \frac{x_t - y_t}{m_1 + 1} if x_t > y_t \end{cases} \tag{2.16}$$

This generator is thought to be highly efficient in the fact that the period is close to 3x10^{57}. The following code[11] implements this algorithm.

LISTING 2.9: MRG32k3a

```
2 %MRG32k3a.m
  m1=2^32-209; m2=2^32-22853;
4 ax2p=1403580; ax3n=810728;
```

[11] Kroese, D.P., Taimre, T., & Botev, Z.I. (2011). Handbook of Monte Carlo Methods.

```
ay1p=527612; ay3n=1370589;

x=[12345 12345 12345]; % Initial x at -1, -2, -3
y=[12345 12345 12345]; % Initial y at -1, -2, -3

n=100; % Compute the sequence for N steps
u=zeros(1,n);
for t=1:n
x_t=mod(ax2p*x(2)-ax3n*x(3),m1);
y_t=mod(ay1p*y(1)-ay3n*y(3),m2);
if x_t <= y_t
u(t)=(x_t - y_t + m1)/(m1+1);
else
u(t)=(x_t - y_t)/(m1+1);
end
x(2:3)=x(1:2); x(1)=x_t; y(2:3)=y(1:2); y(1)=y_t;
end
```

An alternative is the combination of generators of different type. This is done for example by Marsaglia, that proposed KISS99 [12] (Keep It Simple and Stupid) generator. KISS generator combines two multiply-with-carry with a congruential generator:

LISTING 2.10: KISS99

```

% KISS99
%Source: Handbook of the Montecarlo method, 2011
% Seeds: Correct variable types crucial

A=uint32(12345); B=uint32(65435); Y=12345; Z=uint32(34221);
n=100; % Compute the sequence for N steps
u=zeros(1,n);
for t=1:n
% Two Multiply with Carry Generators
A=36969*bitand(A,uint32(65535))+bitshift(A,-16);
B=18000*bitand(B,uint32(65535))+bitshift(B,-16);
% MWC: Low and High 16 bits are A and B
X=bitshift(A,16)+B;
% CONG: Linear Congruential Generator
y = mod(69069*y+1234567,4294967296);
% SHR3: 3-Shift Register Generator
z=bitxor(z,bitshift(z,17));
z=bitxor(z,bitshift(z,-13));
z=bitxor(z,bitshift(z,5));
% Combine them to form the KISS99 generator
KISS=mod(double(bitxor(x,uint32(y)))+double(z),4294967296);
u(t)=KISS/4294967296; % u[0,1] output
```

[12] Kroese, D.P., Taimre, T., & Botev, Z.I. (2011). Handbook of Monte Carlo Methods.

```
end
```

2.7 Statistical tests

When a random number generator is implemented, the main objective is to produce an output whose distribution is at least close to the one needed. In this chapter, as outlined at the beginning, we focused on the most basic distribution available, the Uniform. Even if there are many considerations when we choose among different generators (portability, period length, theoretical foundation), the first and most important constraint is the capability of the generator to produce a distribution that is very close to the desired one. For this reason, in conclusion of this chapter, it is necessary to describe various kinds of statistical tests that are usually performed in order to assess the quality of random number generators. The introduction of this topic in the context of uniform random number generators is not by chance. Indeed, since every random number generator starts from the Uniform, the quality ultimately depends on the closeness of the output stream to a uniform distribution.

The huge body of literature concerned in random numbers generation describes and proposes various kind of tests. In general terms, two different categories of tests are available. First, it is possible to analyse the qualities of a random number generators starting directly from the algorithm, with no need to run the simulation. However, those kind of tests are time consuming and outdated. Consequently, the focus is on the other category. The second kind of tests proposed by literature concerns the output. The idea behind those test is relatively simple: given the random output produced by the generator, it is only necessary to evaluate if this output is close enough to a genuine Uniform distribution. This can be easily done via many different statistical tests. If those tests are performed many times, then it is possible to have a precise idea regarding the quality of the generator. The statistical instruments employed, then, are not ad-hoc constructed for simulation exercises, but they are indeed the typical tools used whenever it is necessary to assess the shape of a distribution.

As specified in Chapter 1, a first way to evaluate an output stream is to plot in a histogram the distribution and compare it to the desired one. This technique is of course non-rigorous nor precise, but it is an optimal starting point since it gives a very communicative visual idea of the quality of the generator. If, for example, the histogram results to be very distant from the uniform distribution, the agent should conclude that such a generator is not the best choice. Of course, the reverse is not necessarily true. If, for example, the generated histogram is very close to the uniform distribution, it is anyway necessary to perform some tests in order to be able to draw that same conclusion in a more scientific and precise way.

A second non rigorous but yet effective way to look at the distribution of generated data is to focus on sample moments. If we expect the generated sample to be

distributed close enough to a uniform $U(0,1)$ then we expect the sample mean $\overline{\mu}$ to be close to 0.5 and the sample variance $\hat{\sigma^2}$ to be close to $\frac{1}{12} = 0.0833$. In order to test if this is the case it is enough to run a t-test for the mean or an F-test for the variance.

Stressing the fact that the main objective of a solid (uniform) random number generator is to produce a stream of random numbers whose distribution is indistinguishable from a uniform $U(0,1)$, the available statistical tests to be performed belong to two main families:

- **Static tests**: those focus on the generated outcome as a whole, thus not considering the order of generation;
- **Dynamic tests**: those focus on the generated outcome as a sequence. Consequently, the order of generation plays a significant role.

2.7.1 Static tests

Static tests generally concern the shape of the distribution.

All statistical tests presented in this section are goodness of fit tests. Goodness of fit tests assess whether an empirical distribution fits the expected theoretical distribution. In this specific case, goodness of fit tests verify whether the generated distribution resembles the Uniform $U(0,1)$. In all the following tests, then, the null hypothesis, H_0 will then be that the random numbers generated by the algorithm are uniformly distributed in $(0,1)$, i.e.,

$$H_0 : \ X \sim \ U(0,1).$$

The alternative hypothesis, H_a or H_1, encompasses all different occurrences, this means that the alternative hypothesis is that the random sequence has not the distribution mentioned above:

$$H_a : \ else.$$

Rejecting the null hypothesis is not the desired outcome since it would imply that the generated distribution is not close to the Uniform. In this context the aim is not to find enough evidence to reject H_0.

While all those tests will then have same hypotheses, tests statistics and procedures will of course differ. In general, all tests should allow the agent to draw the same conclusion, but they are characterized by different degrees of accuracy.

All tests can also be performed only on fractions of the distribution, in order to eventually verify whether there are areas of the generated distribution in which the fit is closer.

Chi-Squared Goodness of fit tests Those tests are by far the most used goodness of fit tests. A Chi-Squared goodness of fit test is a non-parametric test whose aim is to determine whether the theoretical distribution (in this case, the Uniform

$U(0,1)$) fits the empirical one. From a technical point of view, the interval of definition of the variable is divided in sub-intervals. In the case of uniform $U(0,1)$ intervals can be defined as $(0, 0.01)$ $[0.01, 0.02)$, $[0.02, 0.03)$,...,$[0.99, 1)$. Note that the choice of the number of intervals, k, is up to the agent. For each interval I_i, the number of points drawn from the empirical distribution that belongs to that interval, x_i, is computed. At the heart of the test lies then the comparison between the empirical counts, x_i, and the count of expected point in the theoretical distribution, e_i. In informal terms, if the empirical counts are sufficiently close to the expected ones, then there is not enough evidence to reject the null hypothesis H_0. For this test, the test statistic is in the form:

$$\chi_c^2 = \sum_{i=1}^{k} \frac{(x_i - e_i)^2}{e_i}.$$

Under the null hypothesis, H_0, the distribution of this test statistic is χ^2 with $k-1$ degrees of freedom. Looking at the definition of the test statistic, it is evident that, the lower its value, the smaller the distance between empirical and expected values, the lower the likelihood that the empirical distribution is significantly different from the theoretical one. In formal terms, in order to perform the test, it is necessary to compute the p-value, that is the probability that a χ^2 with $k-1$ degrees of freedom has values $\leq$ than the observed statistic. Then the conclusion is drawn with the standard tools of hypothesis testing, i.e., the p-value is compared to the probability of type I error, α. If p-value $< \alpha$, then H_0 is rejected.

Kolmogorov-Smirnov Tests The Kolmogorov-Smirnov test is another widely used technique to assess the shape of a distribution. It is again a non-parametric test used to compare one-dimensional probability distributions. In this case, the comparison is not between counts but directly between the cumulative distribution functions. On the one hand, there is the empirical cumulative distribution function $U_n(x)$. On the other, the cumulative distribution function of the uniform theoretical distribution $U(x)$. The test is based on the analysis of the difference between these two distributions: $KS_n = \sup |(U_n(x) - U(x)|$. This metric is called Kolmogorov distance. sup is the supremum of the distances. Once the test statistic has been computed, it can be compared to the relevant critical value in order to reject or not the null hypothesis. Tables with critical values are available for most distributions.

It is also possible to perform other goodness-of-fit tests using the difference $|(U_n(x) - U(x)|$. For example, the Cramer-von Mises statistic can be used:

$$W^2 = \int_{-\infty}^{\infty} (U_n(x) - U(x))^2 dU(x).$$

Another possibility is to use the Anderson-Darling statistics:

$$A^2 = \int_{-\infty}^{\infty} \frac{(U_n(x) - U(x))^2}{U(x)(1 - U(x))} dU(x).$$

The Anderson-Darling test is often employed in the analysis of random number generators.

2.7.2 Dynamic tests

Dynamic tests investigate correlation patterns in the generated data.

Run tests Run tests are non-parametric tests generally employed to assess the degree of autocorrelation in a sequence of numbers. In order to state that the generated sequence is random, we expect a low or absent autocorrelation (that is, correlation with lagged values). They are considered as the most precise form of tests for randomness. The test statistic can be defined in different ways. First, given the sequence of generated data, a run is defined as a sequence of a symbol $+$ or $-$. Tests statistic cane both be defined based on the number or length of runs.

Test for autocorrelations Another way to look at the structure of the generated sample is to consider autocorrelations. Autocorrelations can be defined for different values of the lag k, but it is beneficial if $\forall k$ those autocorrelations are close to 0. This would, in fact, imply that there is not a clear dependence over time in the data generated. For a generic lag k, the autocorrelation is computed as:

$$r_k = \frac{\sum_{i=1}^{n}(u_i - 0.5)(u_{i-k} - 0.5)}{\sum_{i=1}^{n}(u_i - 0.5)^2} \tag{2.17}$$

Chapter 3

Generation of random numbers from other distributions

This chapter introduces how random numbers drawn from a non-uniform distribution can be generated. The starting point for every implementation is always the set-up of a uniform random number generator algorithm. The reason lies in the fact that every random number generator employs at least one uniform-generator, that is modified and combined in order to produce random numbers obeying to the desired distribution. Even more importantly, once transformations have been applied, the quality of the generator still critically depends on the quality of the uniform generator. In formal terms, in order to generate realizations of a generic random variable X distributed according to a generic probability distribution, it is necessary first to generate uniformly distributed random numbers in $(0,1)$. Second, it is necessary to transform these numbers via the function g, i.e. generate $X = g(u_1, ..., u_k)$. Where g is a function from $(0,1)^k$ to $\mathbb{R}^d$. This chapter should be then interpreted as a natural follow-up of the previous one. This chapter is fundamentally divided into two sections: in the first one, "exact" and "universal" methods are described. Those methods are "universal" because they can be used for the generation of (almost) every distribution if the specific requirements are met. They also are "exact" because once applied, the sample generated has exactly the desired distribution[1]. The second part describes specific (and more efficient) methods for the generation of random variables drawn from the most used distributions. If exact transformations are too complex to be implemented, then approximation methods are used. Among these, the most famous one is Markov Chain Montecarlo (MCMC).

3.1 Exact methods

3.1.1 Inverse-transform method

Consider a generic random variable X with cumulative distribution function, CDF, $F(x)$ defined as :

$$F(x) = P(X \leq x)$$

[1] Of course assuming that the uniform generator is exact.

where P is the probability operator. By definition, the CDF is non-decreasing and right-continuous. Following the fact that F is non-decreasing, it is possible to define the inverse function F^{-1} as:

$$F^{-1}(y) = inf\{x : F(x) \geq y\}, 0 \leq y \geq 1. \tag{3.1}$$

If X is a discrete random variable with continuous CDF F, it can be shown that the random variable

$$U = F(x).$$

For a formal proof refer to Appendix A.1. Then, only via the definition of the cumulative distribution function, a relationship between uniform random variables and continuous random variables has been defined. Specifically, given a continuous random variable X, it is related to a uniform random variable $U(0,1)$ via the following:

$$X = F^{-1}(U) \tag{3.2}$$

This relationship is usually referred to as the Inverse continuous distribution function technique. Then, it follows immediately that in order to generate a stream of random variates distributed as X, it is only necessary to follow two steps. First, a stream of uniformly distributed random numbers $U(0,1)$ is generated, then the transformation is applied: $X = F^{-1}(U)$. The result is a stream of numbers with the required distribution X. This method is generic since it can be applied to any distribution; the requirement is only to know the CDF F and to be able to compute the inverse F^{-1}. This method has received great attention in literature since it is of easy comprehension and implementation. However, it is not always the best choice. There are cases, for example, in which the inverse CDF is very complex (or even impossible) to evaluate. In these cases, it may be better to apply other methods (such as those described in the following pages) or even to stick to the inverse transformed method by solving the equation:

$$F(x) - u = 0.$$

In order to practically understand how the method works for continuous random variables, consider a random variable whose probability distribution function is defined as follows:

$$f(x) = \begin{cases} 2xifx \in [0,1] \\ 0otherwise \end{cases} \tag{3.3}$$

The cumulative distribution function is just $F(x) = \int_0^x 2zdz = x^2, \forall x \in [0,1]$. Applying the transformation, the inverse cumulative distribution function is just:

$$F^{-1}(u) = \sqrt{u}, \forall u \in [0,1]. \tag{3.4}$$

In order to generate a stream of random variables distributed according to X, it is necessary to generate a stream of uniform random variates $U(0,1)$ and then apply the transformation in 3.4. This procedure is implemented in the code below:

LISTING 3.1: Inverse transform-Continuous variable

```
%% Inverse transform-Continuous variable

clc
clear all

n=1000; %set number of repetitions

U=rand(n,1);%generate uniform

X=sqrt(U);%generate desired random variable

%plot histogram of generated distribution
hist(U)
hist(X)
```

The inverse transform method has just been defined for continuous random variables. Only in this case, in fact, it is possible to mathematically (via analytic or algorithms) compute the inverse of the cumulative distribution function. There are indeed many distributions for which the method is in principle non-applicable, since it is impossible to define the inverse of F in closed form, i.e. it is impossible to solve w.r.t. x

$$F(x) = \int_{-\infty}^{x} f(z)dz = u \tag{3.5}$$

However, it is also possible to define an analogous method applicable to discrete random variables. Define a discrete random variable X that takes values $x_1 < x_2 < x_3 < ...$ with probabilities $p_1, p_2, p_3, ...$ respectively. The points $x_1, x_2, x_3, ...$ are usually referred to as mass points. Note that by definition $\sum_i p_i = 1$. The (discrete) distribution function is defined as :

$$P(x) = \sum_{x_i \leq x} p_i \tag{3.6}$$

The first step of the implementation of the method is just the generation of a stream of uniformly distributed random numbers, as in the continuous case. Then, in the second step, the algorithm should search for the smallest number k s.t. $P(x_k) \geq u$. Then the output is $X = x_k$.

A well known discrete distribution is the Bernulli. Given the parameter $p \in (0,1)$, the distribution function is given by:

$$p(x) = p^x(1-p)^{1-x}, x = 0, 1.$$

The generation of a stream of random numbers distributed according to Bernoulli, is simply generated by the code below:

LISTING 3.2: Inverse transform-Bernoulli

```
%% Inverse transform method Bernulli

clc
clear all

p=0.2%choose value for the parameter
n=1000%set number of repetitions

u=rand(n,1);%generate uniform

%set up loop for generation of Bernoulli

for i=1:n
 if u(i)<p
  u(i)=0
 else u(i)=1
end
end
```

In order to draw some efficiency conclusions consider another example. Define a discrete random variable X characterized by the following probability function:

TABLE 3.1: Distribution of r.v. X

x_i	p_i
1	0.1
2	0.2
3	0.3
4	0.4

We can run two different codes in order to generate random numbers according to this distribution. The first example is the following:

LISTING 3.3: Inverse transform-Discrete 1

```
%% Inverse transform- Discrete 1

clc
clear all

n=1000 %set number of repetitions
p=[0.1,0.2,0.3,0.4] %vector of probabilities

```

```
x=zeros(n,1) %preallocate the space

for i=1:n
x(i)=min(find(rand<cumsum(p)))
end
```

This code simply transforms in programming language the definition of the method that was just given. In fact, in the loop, Matlab looks for the smallest number k s.t. $P(x_k) \geq u$. The generation of uniform random numbers is included directly inside the loop. This code is able to generate a correct result, but there is a faster alternative:

LISTING 3.4: Inverse transform-Discrete 2

```
%% Inverse transform- Discrete 2

clc
clear all

n=1000 %set number of repetitions
p=[0.1,0.2,0.3,0.4] %vector of probabilities

[dummy,x]=histc(rand(1,n),[0,cumsum(p)])
```

3.1.2 Alias method

The Alias method was proposed by Walker in 1977 and it is an alternative to the Inverse-transform method applicable only to discrete distributions. It is usually more efficient than the Inverse-transform, since it avoids the second part of the algorithm (search), that is usually highly time consuming. This method is based on Walker's finding that every discrete distribution characterized by m mass points can be described as a weighted average of m (discrete) distributions, each one with only two mass points. Define a random variable X with m mass points $x_1, x_2, ..., x_m$ and associated probabilities $p_1, p_2, \ldots, p_m$; $p_i > 0 \ \forall i$ and $\sum_i p_i = 1$. Now we distinguish two cases. First, if $p_1 = p_2 = ... = p_m$ then clearly the random variable X can be described as a weighted average of m 2-point random variables with weights equal to $p_i \ \forall i$. If instead probabilities are not equal, i.e. $\exists p_i \neq p_j$ for some $i \neq j$, then $\exists i, j$ s.t. $p_i < \frac{1}{n}$ and $p_j \geq \frac{1}{n}$. At the end we construct m intervals $i = 1, 2, ..., m$ with a two point distribution and mass points i and a_i (alias values). The respective probabilities of the mass points are q_i (cut-off values) and $1 - q_i$.

The Alias method is divided into two parts. The first one is the set-up and it is usually the most complex one. In this phase, the goal is to implement Walker's procedure, the one just described above. Let X be a discrete random variable with mass points $x_1, x_2, ..., x_m$ and associated probabilities $p_1, p_2, ..., p_m$. Define $q_i = mp_i$ with $i = 1, 2, ..., m$. Then define the existence spaces $A = \{i : q_i < 1\}$ and $B = \{i : q_i \geq 1\}$. Then we have to select some k, j s.t. $k \in A$ and $j \in B$ and impose $a_k = j$ and $q_j = q_j - (1 - q_k)$. Now two alternatives are possible:

- if $q_j < 1$ then j is removed from B and added to A. i is then removed from A.
- otherwise, j is kept in B.

Once the method has been set up, it is possible to proceed with the generation of random numbers. First, as usual, it is necessary to generate uniformly distributed numbers in the interval $U(0,1)$. Then we define $I = \lceil mU \rceil$. Then we generate another uniform random variable $V(0,1)$, if $v \leq q_i$, we set $X = I$, if not $X = a_i$. The following code clarifies the procedure:

LISTING 3.5: Alias method

```
%% Alias method (adapted from Dirp K. Kroese)

clc
clear all

p=rand(1,200);
p=p/sum(p);
n=size(p,2); %sample size
a=1:n;
q=zeros(1,n);%initialyze the space
q=n*p;
B=find(q>=1);
A=find(q<1);
while (~isempty(A) && ~isempty(B))
i = B(1);
j = A(1);
a(i) = j;
q(j) = q(j) -(1- q(i));
if (q(j) < 1)
A = setdiff(A,j);
B = union(B,j);
end
B = setdiff(B,i);
end
pp = q/n
for i = 1:n
ind = find(a == i);
pp(i) = pp(i) + sum((1 - q(ind)))/n;
end
max(abs(pp - p))
N = 10^6; % generate sample of  size N
X = zeros(1,N);
for i = 1:N
K = ceil(rand*n);
if (rand > q(K));
X(i) = a(K);
else
```

```
    X(i) = K;
39  end
    end
```

3.1.3 Acceptance-rejection method

We now turn to one of the most widely used methods in the generation of random numbers. Acceptance-rejection is again a general method, in the sense that it can be applied to almost every distribution, both in the continuous and in the discrete case. Indeed, this is one of the few methods that can also be applied to multivariate distributions [2]. The idea behind this method is to use a "proposal" random variable Y in order to generate a random sample drawn from X. The condition is that X and Y should have sufficiently close distribution functions.

Assume we want to generate a random sample drawn from the distribution X, characterized by density function f_x. In order to do this choose a random variable Y with density function g_y in a way such that, $\forall x$, we have that $cg_y(x) \geq f_x(x)$, where $c \geq 1$ is a constant. If we can find such a random variable Y, then we call its density function g_y majorizing density and the term cg_y majorizing function. In order to implement the method, first realizations of the random variable Y are generated (according to g_y), then a stream of uniformly distributed random numbers in $(0, 1)$ is generated. This stream will serve as a benchmark. Finally, the acceptance-rejection is performed. Specifically, if, at each realization, $u_i \leq \frac{f_x(y_i)}{cg_y(y_i)}$, then the value y_i is taken as element of the random vector for X (acceptance). If not, then that value is disregarded (rejection), and the process continues with the analysis of the value y_{i+1}. The algorithm must be performed until the moment in which the full sample n is generated. It can be proved that the generated sample is distributed with density function f_x. Proof is provided in Appendix A.2. The Acceptance-rejection method can thus be defined as an indirect method since it generates a random variable via the generation of another one.

While this method is fairly simple to understand and implement, efficiency considerations are necessary. Assume from now on that the generation of the proposal random variable Y is not subject to efficiency considerations, i.e., the most efficient generation algorithm is available. We will thus focus on the "core" acceptance-rejection technique. The efficiency measure of the method is given by the probability that an acceptance will occur. The reason is simple; the higher this probability, the higher the number of acceptances over a sample n. This implies that the algorithm will come to an end faster. The probability of acceptance is given by:

$$P(U \leq \frac{f_x(Y)}{cg_y(Y)}) = \int g_y(y) \int_0^1 I_{u \leq \frac{f_x(y)}{cf_y(y)}} du dy = \int \frac{f_x(y)}{c} dy = \frac{1}{c} \tag{3.7}$$

[2] Even if the efficiency of the generator decreases significantly as the number of dimensions increases.

The obtained probability of acceptance has a geometric distribution with parameter $\frac{1}{c}$. Consequently, in expectation, the number of trials necessary in order to generate a full sample is equal to the scalar c.

Of course, another element that may hamper the efficiency of this method is the choice of the proposal random variable Y. In general, the closest the distribution of Y to X, the faster the algorithm will be. However, since the selection of the best distribution of Y may be time-consuming, this variable is usually chosen to be very simple. Common choices are, in fact, uniform or exponential.

In order to explain how this method works, we consider a simple example in which the goal is to generate random numbers drawn from a positive normal distribution. It will be shown via a MATLAB implementation that this method is extremely efficient. The distribution function of a positive Normal is in the form:

$$f(x) = \sqrt{\frac{2}{\pi}} e^{-x^2/2}, x \geq 0 \tag{3.8}$$

First step is to choose an appropriate proposal distribution. We select the Exponential with parameter 1, whose distribution function is in the form e^{-x^2}. We set $c = \sqrt{\frac{2e}{\pi}}$, the efficiency measure is then $(\sqrt{\frac{2e}{\pi}})^{-1} = \sqrt{\frac{\pi}{2e}}$. The code below implements this technique.

LISTING 3.6: Acceptance-Rejection Positive Normal Distribution

```
%% Acceptance-Rejection Positive Normal Distribution
clc
clear all

n=100 %set number of repetitions
y=zeros(n,1) %preallocate the space
x=exprnd(1,n,1) % generate proposal
u=rand(n,1)% generate uniform

%implement AR

for i=1:n
if u(i)<=sqrt(2/pi)*exp(-x(i)^2/2)/sqrt(2*exp(1)/pi)
y(i)=x(i)
else y(i)=NaN
end
end

%% Efficiency

eff=sqrt(pi/2*exp(1))
```

If the distribution from which we want to draw random data is complex, it is still possible to employ a modified version of this method in order to gain in efficiency

terms, by making the algorithm faster. The modification is usually called squeeze. When we "squeeze" we arbitrarily define two (not only one) new density functions. First, the usual majorizing density, g. Second, a very simple function, s, s.t.:

$$s(x) \leq f(x) \leq cg(x), \forall x.$$

When a squeeze is implemented we usually do not check if $u_i \leq \frac{f_x(y_i)}{cg_y(y_i)}$, but instead if $u_i \leq \frac{s(y_i)}{cg_y(y_i)}$. This process is usually much faster.

3.1.4 Ratio of Uniforms method

The ratio of uniforms is another (yet efficient) method proposed for the first time by Kinderman and Monahan in 1977. It is applicable both to discrete and continuous distributions. This method is based on the comparison of two variables, and the advantage is that it is not necessary to know the distribution of the two variables when considered separately.

Let U and V be two random variables. Define a new random variable as their ratio $Y = V/U$. Now assume that the multivariate distribution (U, V) is uniformly distributed over the bi-dimensional region:

$$C = \{(u, v), s.t. u \in [0, \sqrt{h\frac{v}{u}}]\} \tag{3.9}$$

If h is a non-negative integrable function over the same set, then the density of the ratio $Y = V/U$ is proportional to h. Proof of this result is provided in Appendix A.3.

Then, in order to implement the method, we simply need to generate two i.i.d. random variables U and V, uniformly in $(0, 1)$. Then the first values have to be set as $u_1 = bu$ and $v_1 = c + (d - c)v$. Define the random variable to generate, X, as $X = V/U$. The last step involves an acceptance-rejection introduction: if $u_1^2 \leq h(x_1)$, then x is taken as realization, else the process continues with the next value.

It is also possible to define this method for discrete random variables. This was proposed for the first time by Stadlober(1990). In analogy with the continuous case, the first step is to generate two i.i.d. uniform random variables in $(0, 1)$, U and V. Then, x is this time defined as $x = \lfloor a + s(2v - 1)/u \rfloor$ and $y = u^2$. Then the acceptance-rejection is implemented: if $y_1 \leq p(x_1)$ then x is taken as realization, else the process continues with the next value.

3.2 Specific distributions

This section applies the methods described in the previous one to the generation of random variables from the most frequently used distributions, both in the continuous and discrete case. For each distribution, some examples of algorithms and Matlab implementation are given.

3.2.1 Bernoulli Distribution

A Bernoulli distribution is a discrete distribution that describes the outcome of events that can result either in a failure (0) or success (1).

The probability distribution function of a Bernoulli random variable is in the form:

$$f(x;p) = p^x(1-p)^{1-x} \tag{3.10}$$

p is the characteristic parameter of a Bernoulli and it is called probability of success, $p \in [0,1]$. The most used method for the generation of Bernoulli random variables is the inverse transform. After the generation of a uniform random variable in $(0,1)$, $U \sim U(0,1)$, for each realization of U the value u_i is compared to the parameter p; if $u_i \leq p$ a success is awarded, i.e. $x_i = 1$, otherwise a $x_i = 0$ is recorded. An example of the implementation in Matlab is given in the code below.

LISTING 3.7: Bernoulli

```
%% Bernulli
clc
clear all

n=100;
p=0.25;
u=rand(n,1);
x=zeros(n,1);

for i=1:n
if u(i)<=p
x(i)=1
else x(i)=0
end
end

bar(x)
```

3.2.2 Binomial Distribution

A Binomial is another discrete distribution. It describes the number of successes in a sequence of n Bernoulli random variables. A Binomial is in fact defined as the sum of n independent Bernoulli. The probability distribution function of a Binomial is in the form:

$$f(x;p,n) = \binom{n}{x} p^x(1-p)^{n-x} \tag{3.11}$$

where p is the probability of success in one trial and n is the number of trials, $p \in [0,1]$, n is a positive integer and $x = 0,1,2,...,n$. The generation of Binomial random variables is straightforward if one refers to the fact that a Binomial random variable can be described as the sum of n Bernoulli. It is then only necessary to generate n

i.i.d. Bernoulli random variables, $X_1, ..., X_n \sim Ber(p)$, and then take their sum $Bin = \sum_{i=1}^{n} X_i$. Such an algorithm (which is indeed equivalent to an inverse transform technique) is easy to implement and understand but it is likely to be poor in terms of efficiency. The reason is that, as n grows, the time required by the algorithm to run increases too. If n is big it is then possible to apply a geometric method, described in the following algorithm:

LISTING 3.8: Binomial geometric

```
%Binomial geometric (Adapted from D.P. Kroese)
n = 100; p = 0.1; mu = n*p; N = 10^5;
x = zeros(1,N); c = log(1-p);
for i=1:N
s = ceil(log(rand)/c);
xi = 0;
while s < n + 1
xi = xi+ 1;
s = s + ceil(log(rand)/c);
end
x(i)= xi;
end

xx = [floor(mu - 4*sqrt(mu)):1:ceil(mu + 4*sqrt(mu))];
count = hist(x,xx);
ex = binopdf(xx,n,p)*N;
hold on
plot(xx,count,'or')
plot(xx,ex,'.b')
hold off
```

It is also possible to generate a Binomial distribution trough a Normal, obtaining a reliable approximation. The reason is that a Binomial distribution with parameters n, p can be approximated by a Normal distribution with mean $np - 0.5$ and variance $np(1-p)$ if n is large enough. This powerful result comes from the central limit theorem. If n is large, it is then possible to approximate the Binomial r.v. by using a Normal. It will then be necessary to generate a r.v. distributed as a standard normal, $Z \sim N(0,1)$ and then approximate the Binomial as $Bin = max\{0, \lfloor np + 0.5 + Z\sqrt{np(1-p)} \rfloor\}$. This algorithm is implemented in the code below:

LISTING 3.9: Binomial via Normal

```
%% Binomial via Normal
clc
clear all

n=100;
p=0.1;
y=randn(1,n)';
```

```
x=zeros(1,n)';

for i=1:n
x(i)=max(0,floor(n*p+0.5+y(i)*sqrt(n*p*(1-p))))
end
```

This method is incredibly fast but it is approximate. For this reason, if the nature of the analysis implies the need for the use of exact techniques, two recursive generators are proposed.

LISTING 3.10: Binomial Recursive 1

```
%% recursive binomial generator 1 (Adapted from D.P. Kroese)
function x=binomialrnd(n,p)
% recursive binomial generator
if n<=10
x=sum(rand(1,n)<p);
else
k=ceil(n*p);Y=nbinrnd(k,p);% generate NegBin(k,p)
T=k+Y;
if T<=n
x=k+binomialrnd(n-T,p);
else
x=k-binomialrnd(T-n,p);
end
end
```

LISTING 3.11: Binomial Recursive 2

```
%% recursive binomial generator 2 (Adapted from D.P. Kroese)
function x=binomrnd_beta(n,p)

if n<=10
x=sum(rand(1,n)<p);
else
k=ceil(n*p);Uk=betarnd(k,n+1-k);% generate beta r.v.
if Uk<p
x=k+binomrnd_beta(n-k,(p-Uk)/(1-Uk));
else
x=k-binomrnd_beta(k-1,(Uk-p)/Uk);
end
end
```

3.2.3 Geometric Distribution

The Geometric is a discrete distribution that describes the number of trials that corresponds to the first success in a sequence of ∞ Bernoulli random variables. The

probability distribution function of a Geometric random variable is in the form:

$$f(x;p) = p(1-p)^{x-1} \tag{3.12}$$

where p is the probability of success of the Bernoulli and $x = 1, 2, 3, ..., +\infty$. The generation of Geometric random variables is based on a theoretic consideration related to the link between Geometric and Exponential random variables. Define an Exponential random variable Y with parameter $\lambda = -ln(1-p)$. In this case the ceil of this exponential is distributed as a Geometric random variable with parameter p, $\lceil Y \rceil \sim Geom(p)$. Then, in order to generate a Geometric, $Geom(p)$, it is only necessary to generate an Exponential with parameter $\lambda = -ln(1-p)$ and then apply the ceil function.

LISTING 3.12: Geometric via Exponential

```
%% Geometric 1
clc
clear all

n=100;
p=0.2;
y=exprand(-ln(1-p),1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=ceil(y(i))
end
```

Alternatively, it is also possible to generate a Geometric starting from a random Uniform. In this case, once $U \sim U(0,1)$ has been generated, it is only necessary to apply the transformation $Geom = \lceil ln(\frac{U}{1-p}) \rceil$:

LISTING 3.13: Geometric via Uniform

```
%% Geometric 2
clc
clear all
n=100;
p=0.2;
u=rand(1,n)';
x=zeros(1,n)';
for i=1:n
x(i)=ceil(ln(u(i)/ln(1-p))
end
```

3.2.4 Hypergeometric Distribution

The Hypergeometric is a discrete distribution and its probability distribution function is in the form:

$$f(x;n,r,N) = \frac{\binom{r}{x}\binom{N-r}{n-x}}{\binom{N}{n}} \tag{3.13}$$

where N, n, r are positive integers, $n \leq N, r \leq N$ and $x \in [max\{0, r+n-N\}, min\{n, r\}]$. This distribution can be described as follows. Consider a bowl with N stones in it. r stones are red. Take n stones from the N in the bowl randomly without replacement. The number of red stones that are drawn from the bowl in the n trials has an Hypergeometric distribution with parameters n, N, r. It follows that, in order to generate Hypergeometric random variables, it is only necessary to simulate the n trials without replacement and define the random variable X as the number of red stones that are taken.

LISTING 3.14: Hypergeometric

```
%% Hypergeometric (Adapted from D.P. Kroese)
N = 100; %total number of stones
n = 20; % take n stones
r = 30; % number of red stones
w = zeros(1,N);
w(1:r) = 1;
K = 10^5;
x = zeros(1,K);
for i=1:K
[s,ix] = sort(rand(1,N));
x(i) = sum(w(ix(1:n)));
end

xx = [0:n];
count = hist(x,xx);
ex = hygepdf(xx,N,r,n)*K;
clf
hold on
plot(xx,count,'.r')
plot(xx,ex,'ob')
hold off
```

3.2.5 Negative Binomial Distribution

The Negative Binomial is a discrete distribution with probability distribution function:

$$f(x;p,r) = \frac{\Gamma(r+x)}{\Gamma(r)x!}p^r(1-p)^x \tag{3.14}$$

where $r \geq 0$, $p \in [0,1]$, Γ is the complete Gamma function and $x = 0, 1, 2, ...$ If r is an integer (we then write $n = r$) the distribution is called Pascal and has a probability

distribution function in the form:

$$f(x; p, n) = \binom{n+x-1}{n-1} p^n (1-p)^x \tag{3.15}$$

If we interpret p as the probability of success of a Bernoulli, then this random variable can be seen as representing the number of trials that are necessary before achieving r successes. In order to generate random variables distributed according to the negative binomial (or Pascal), one has to note that the negative binomial can be represented as the sum of r (or n) geometric random variables with parameter p. This consideration leads to the following algorithm (which may be used only when r is not significantly large):

LISTING 3.15: Negative Binomial

```
%% Negative Binomial

clc
clear all
n=10;
r=10;
p=0.2;
u=rand(1,n);
y=zeros(1,n);
c=ln(1-p);

for i=1:n
y(i)=floor(ln(u(i)/c)
end

x=cumsum(y)
```

3.2.6 Poisson Distribution

The Poisson is a discrete distribution characterized by the following probability distribution function:

$$f(x; \lambda) = \frac{\lambda^x}{x!} e^{-\lambda} \tag{3.16}$$

where $\lambda > 0$ is the parameter of the Poisson (rate parameter) and $x = 0, 1, 2, ...$ This random variable is used to describe the distribution of arrival times. In order to generate a Poisson random variable it is common to rely on the generation of Gamma and Binomial random variables, as in the code below:

LISTING 3.16: Poisson

```
%% Poisson
clc
clear all

```

```
n=100;
l=0.2;
m=floor(7/8*l);
y=gamrnd(m,1,1,n)';
z=zeros(1,n)';
x=zeros(1,n)';

for i=1:n
if y(i)<=l
z(i)=poissrnd(l-y(i))
x(i)=m+z(i)
else x(i)=binornd(m-1,l/y(i))
end
end
```

3.2.7 Beta Distribution

The Beta is a continuous distribution with probability distribution function:

$$f(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \tag{3.17}$$

where $\alpha, \beta > 0$ are the shape parameters and $B(\alpha,\beta)$ is the complete beta function. The Beta distribution is one of those examples for which the generation technique to use critically depends on the shape parameters. Depending on their value, in fact, the algorithms range from very simple to extremely complex.

The simplest case is when α or β are equal to 1. In these two cases, in fact, it is only necessary to apply the inverse method. The first step is common: generate a stream of uniformly distributed random numbers in $(0,1)$, then if $\alpha \neq 1$ and $\beta = 1$ the transformation to be applied is $Beta = U^{\frac{1}{\alpha}}$. Conversely, if $\alpha = 1$ and $\beta \neq 1$, the transformation is in the form $Beta = U^{\frac{1}{\beta}}$. The codes below exemplify what just described:

LISTING 3.17: Beta(a,1)

```
%% Beta (alpha,1)
clc
clear all

n=100;
a=0.8;

u=rand(1,n)';
x=zeros(1,n)';
for i=1:n
x(i)=u(i)^(1/a)
end
```

LISTING 3.18: Beta(1,b)

```
%% Beta (1,beta)
clc
clear all

n=100;
b=0.8;

u=rand(1,n)';
x=zeros(1,n)';
for i=1:n
x(i)=1-u(i)^(1/b)
end
```

If both α and β are equal to 0.5 the distribution is called Arcsine and again a simple inverse method is applied. This time in the second step we have that $Beta = cos^2(\frac{\pi U}{2})$:

LISTING 3.19: Beta(0.5,0.5)

```
%% Beta (0.5,0.5)
clc
clear all

n=100;

u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=cos(pi*u(i)/2)^2
end
```

In the case in which the two parameters are equal and both greater than 0.5 we can choose between a polar method (that evaluates a cosine function) and a rejection method. The second one may be preferred in the case in which the evaluation of the trigonometric function is slow:

LISTING 3.20: Beta(a,a) 1

```
%% Beta (alpha,alpha) 1
clc
clear all

n=100;
a=0.7;

u_1=rand(1,n)';
u_2=rand(1,n)';
x=zeros(1,n)';
```

```
for i=1:n
x(i)=0.5*(1+sqrt(1-u_1(i)^(2/(2*a-1)))*cos(2*pi*u_2(i)))
end
```

LISTING 3.21: Beta(a,a) 2

```
%% Beta (alpha,alpha) 2
clc
clear all

n=100;
a=0.7;

u=rand(1,n)';
v=rand(1,n)'*2-1;
x=zeros(1,n)';

s=u.^2+v.^2

for i=1:n
if s(i)>1
x(i)=NaN
else x(i)=0.5+u(i)*v(i)/s(i)*sqrt(1-s(i)^(2/(2*a-1)))
end
end
```

Finally, if the parameters are unequal and lower than 1, Johnk's algorithm (1964) can be applied. It is an acceptance-rejection:

LISTING 3.22: Beta Johnk algorithm

```
%% Beta Johnk algorithm
clc
clear all

n=100;
a=0.8;
b=0.6;

u_1=rand(1,n)';
u_2=rand(1,n)';
v_1=ones(1,n)';
v_2=ones(1,n)';

for i=1:n
v_1(i)=u_1(i)^(1/a)
end
```

```
for i=1:n
v_2(i)=u_2(i)^(1/b)
end

w=v_1+v_2
x=zeros(1,n)';

for i=1:n
if w(i)>1
x(i)=NaN
else
x(i)=v_1(i)/w(i)
end
end
```

3.2.8 Cauchy Distribution

The Cauchy is a continuous distribution with pdf in the form:

$$f(x;\mu,\sigma) = \frac{1}{\pi\sigma(1+(\frac{x-\mu}{\sigma})^2)}. \tag{3.18}$$

When $\sigma = 1$ and $\mu = 0$ we call the distribution Standard Cauchy. The easiest way to generate the Standard Cauchy is trough the Inverse Transform method. After having generated a stream of uniformly distributed numbers in $(0,1)$ we apply the transformation $C = tan(\pi U)$ as exemplified in the following code:

LISTING 3.23: Cauchy(0,1)

```
%% Cauchy(0,1)
clc
clear all

n=100;
m=0;
s=1;

u=rand(1,n);
c=zeros(1,n);
for i=1:n
c(i)=tan(pi*u(i))
end
```

However, the most employed techniques use ratio of random variables to generate the Cauchy. Both Ratio of Uniforms and Normals are available and yield efficient results:

LISTING 3.24: Cauchy(0,1) Ratio of Uniforms

```
%% Cauchy(0,1) Ratio of Uniform
clc
clear all

n=100;
u=rand(1,n)';
v=rand(1,n)';
v=v-0.5;
x=zeros(1,n)';

for i=1:n
if u(i)^2+v(i)^2<=1
x(i)=v(i)/u(i)
else x(i)=NaN
end
end
```

LISTING 3.25: Cauchy(0,1) Ratio of Normals

```
%% Cauchy(0,1) Ratio of Normal
clc
clear all

n=100;
y=rand(1,n)';
v=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=y(i)/v(i)
end
```

3.2.9 Exponential Distribution

The Exponential is a continuous distribution with pdf:

$$f(x;\lambda) = \lambda e^{-\lambda x} \tag{3.19}$$

where $\lambda > 0$ is called rate parameter. The exponential is usually considered a specific case of the Gamma distribution where $\alpha = 1$ and $\beta = \frac{1}{\lambda}$. This distribution is at the heart of many finance applications because it is a well known example of memoryless distribution. Moreover, it is usually considered as the continuous counterpart of the geometric distribution[3]. Exponential random numbers are usually generated trough the Inverse Transform method by applying the transformation $E = \frac{-ln(U)}{\lambda}$, as in the code below:

[3]Kroese, D.P., Taimre, T., & Botev, Z.I. (2011). Handbook of Monte Carlo Methods.

LISTING 3.26: Exponential

```
%% Exponential

clc
clear all

l=1.2;
n=100;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=-log(u(i))/l
end
```

3.2.10 Fisher-Snedecor Distribution

The Fisher-Snedecor is a continuous distribution with probability distribution function in the form:

$$f(x;m,n) = \frac{\Gamma(\frac{m+n}{2})(\frac{m}{n})^{m/2}x^{(m-2)/2}}{\Gamma(\frac{m}{2})\Gamma(\frac{n}{2})[1+\frac{m}{n}x]^{(m+n)/2}} \tag{3.20}$$

where $x \geq 0$, $m,n > 0$ and integers. m,n are the degrees of freedom. This distribution is also known as F and it is frequently used in variance hypothesis testing. A random sequence of numbers distributed according to the F is usually generated trough the ratio of Chi-square distributions. In fact, given two Chi-Square random variables $X \sim \chi^2_m$ and $Y \sim \chi^2_n$ we have the following result:

$$\frac{X/m}{Y/n} \sim F(m,n).$$

Then, in order to generate a F it is only necessary to generate two streams of Chi-square distributed random numbers and then apply the transformation $F = \frac{X/m}{Y/n}$. This procedure is implemented in the code below:

LISTING 3.27: F 1

```
%% F 1
clc
clear all

m=5;
n=6;
k=100;
x=chi2rnd (m,k,1);
y=chi2rnd (n,k,1);
z=zeros(1,k);

```

```
for i=1:k
z(i)=(x(i)/m)/(y(i)/n)
end
```

This technique is not used frequently since it tends to be quite slow. An alternative is based on the generation of a stream of numbers distributed according to the Beta distribution and is presented below:

LISTING 3.28: F 2

```
%% F 2
clc
clear all

m=5;
n=6;
k=100;

b=betarnd (m/2,n/2,k,1)';
x=zeros(1,k)';

for i=1:k
x(i)=(b(i)*n)/(m*(1-b(i)))
end
```

3.2.11 Frechet Distribution

The Frechet (also known as type II extreme value) is a continuous distribution with probability density function in the form:

$$f(x;\alpha) = \alpha x^{-\alpha-1} e^{-x^{-\alpha}} \tag{3.21}$$

where $x > 0$ and $\alpha > 0$. α is called shape parameter. This distribution is used frequently when the maximum of i.i.d. random variables is considered. The generation of random numbers distributed according to the Frechet is based on the Inverse Transform method. The transformation to be applied is the following:

$$F = (-lnU)^{-\frac{1}{\alpha}}$$

LISTING 3.29: Frechet

```
%% Frechet(alpha,0,1)
clc
clear all

n=100;
a=3;
```

```
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=(-log(u(i)))^(-1/a)
end
```

3.2.12 Gamma Distribution

The Gamma is a continuous distribution with probability distribution function:

$$f(x;\alpha,\beta) = \frac{\beta^{\alpha} x^{\alpha-1} a^{-\beta x}}{\Gamma(\alpha)} \tag{3.22}$$

where $x \geq 0$, $\alpha, \beta > 0$, Γ is the Gamma function. α is called shape parameter and β is called scale parameter. The Exponential distribution is nothing but a special case of the Gamma with $\alpha = 1$ and $\beta = \frac{1}{\lambda}$. Another relevant case is when $\alpha = \frac{n}{2}$ and $\beta = \frac{1}{2}$. This distribution is then called χ^2 (Chi-square) and n is known as the number of degrees of freedom. An important implication is that Gamma generation techniques are in principle directly applicable to the generation of Chi-square distributions. Another important instance is when n is a positive integer. In this case, the distribution is usually called Erlang. One of the most used techniques for the generation of Gamma variables was proposed by Best and is feasible if $\alpha < 1$:

LISTING 3.30: Gamma Best

```
%% Gamma best
% adapted from D.P. Kroese
N = 10^5; alpha = 0.3;
d= 0.07 + 0.75*sqrt(1-alpha); b = 1 + exp(-d)*alpha/d;
x = zeros(N,1);
for i = 1:N
cont = true;
while cont
U1 = rand;
U2 = rand;
V = b*U1;
if V <= 1
X = d*V^(1/alpha);
if U2 <= (2-X)/(2+X)
cont = false; break;
else
if U2 <= exp(-X)
cont = false; break;
end
end
else
X = -log(d*(b-V)/alpha);
```

```
y = X/d;
if U2*(alpha + y*(1-alpha)) < 1
cont= false; break;
else
if U2 <= y^(alpha - 1)
cont= false;break;
end
end
end
end
x(i) = X;
end

clf
hold on
x = sort(x);
ecdf(x); % empirical cdf
y = gamcdf(x,alpha);  % cdf of gamma distribution
plot(x,y,'r')
hold off
```

In the case in which $\alpha > 1$, Cheng and Fest (1979) proposed an highly efficient algorithm based on the ratio of uniform method:

LISTING 3.31: Gamma Cheng Feast

```
%% Gamma Cheng Feast algorithm
clc
clear all

a=1.2;
n=100;
u_1=rand(1,n)';
u_2=rand(1,n)';

for i=1:n
v(i)=((a-(6*a)^(-1))*u_1(i))/((a-1)*u_2(i))
end

x=ones(1,n)';

for i=1:n
if 2*(u_2(i)-1)/(a-1)+v(i)+1/v(i)<=2
x(i)=(a-1)*v(i)
if 2*log(u_2(i))/(a-1)-log(v(i))+v(i)<=1
x(i)=(a-1)*v(i)
else x(i)=NaN
end
end
```

```
24 end
```

If the aim is to generate a Chi-square random variable with one degree of freedom, it is only necessary to note that it can be viewed as the square of a standard normal:

LISTING 3.32: Gamma Chi-Square

```
%% Gamma(1/2,1/2) (chi-square)

clc
clear all

n=100;
z=randn(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=z(i)^2
end
```

3.2.13 Gumbel Distribution

The Gumbel (also known as type I extreme value) is a continuous distribution with probability distribution function in the form:

$$f(x) = e^{-x-e^{-x}} \tag{3.23}$$

with $x \in \mathbb{R}$. The generation of Gumbel random variables is based on the Inverse-Transform method. Given a stream of uniformly distributed random numbers in $(0,1)$, the tranformation to be applied is $G = -ln(-lnU)$:

LISTING 3.33: Gumbel

```
%% Gumbel

clc
clear all

n=100;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=-log(-log(u(i)))
end
```

3.2.14 Laplace Distribution

The Laplace or double exponential is a continuous distribution with probability distribution function:

$$f(x;\lambda) = \frac{1}{2}e^{-\lambda|x|} \tag{3.24}$$

where $x \in \mathbb{R}$ and $\lambda > 0$. This distribution is widely used in Montecarlo techniques applied to finance since it has fatter tails that the Normal. Due to its popularity, a large body of research was devoted to random generation techniques connected to this distribution. Many different algorithms are thus available. One first example is based on the generation of a Beronulli random variable ($B \sim Ber(0.5)$) and an Exponential ($Y \sim Exp(1)$). The Laplace is then generated as $X = (2B-1)Y$:

LISTING 3.34: Laplace 1

```
%% Laplace 1

clc
clear all

n=100;
y=exprnd(1,1,n)';
x=zeros(1,n)';

p=0.5;
u=rand(1,n)';
b=zeros(1,n)';

for i=1:n
if u(i)<=p
b(i)=1
else b(i)=0
end
end
for i=1:n
x(i)=(2*b(i)-1)*y(i)
end
```

Alternatively, it is also possible to implement a simpler algorithm based on the Uniform. After having generated a stream of uniformly distributed random numbers in $(-0.5, 0.5)$, the transformation $X = sgn(U)ln(1-2|U|)$ is applied:

LISTING 3.35: Laplace 2

```
%% Laplace 2

clc
clear all

n=100;
```

```
u=rand(1,n)'-0.5;
x=zeros(1,n)';

for i=1:n
x(i)=sign(u(i))*log(1-2*abs(u(i)))
end
```

In order to describe the next possible algorithm, we have to note that, given two independent Exponential random variables with parameter 1, $W \sim Exp(1)$ and $V \sim Exp(1)$, $V - W \sim Laplace(0,1)$, then:

LISTING 3.36: Laplace 3

```
%% Laplace 3

clc
clear all

n=100;
v=exprnd(1,1,n)';
w=exprnd(1,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=v(i)-w(i)
end
```

Finally, using again two different random variables, if $E \sim Exp(1)$ and $Y \sim N(0,1)$, then $Laplace = Y\sqrt{2E}$:

LISTING 3.37: Laplace 4

```
%% Laplace 4

clc
clear all

n=100;
e=exprnd(1,1,n)';
y=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=y(i)*sqrt(2*e(i))
end
```

3.2.15 Logistic Distribution

The Logistic is a continuous distribution with probability distribution function:

$$f(x;) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{3.25}$$

with $x \in \mathbb{R}$. The most used generator for the Logistic is an example of Inverse-Transform and it applies the transformation $L = ln(\frac{U}{1-U})$:

LISTING 3.38: Logistic

```
%% Logistic

clc
clear all

n=100;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=log(u(i)/(1-u(i)))
end
```

3.2.16 Log-Normal Distribution

The Log-Normal distribution is a continuous distribution with probability distribution function defined as:

$$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(ln(x)-\mu)^2}{2\sigma^2}} \tag{3.26}$$

where $x > 0$, $\sigma > 0$ and $\mu \in \mathbb{R}$. μ is called location parameter and σ is called scale parameter. A very important property of this distribution is its relationship with the Normal. If $X \sim LogN(\mu, \sigma^2)$, then $lnX \sim N(\mu, \sigma^2)$. The most used algorithm for the generation of Log-normally distributed random numbers applies this property and thus if we generate $Y \sim N(\mu, \sigma^2)$ then we just need to apply $X = e^Y$:

LISTING 3.39: Log-Normal

```
%% Log-Normal

clc
clear all

n=100;
m=0;
s2=1;
```

```
y=randn(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=exp(y(i))
end
```

3.2.17 Normal Distribution

The Normal (or Gaussian) distribution is a continuous distribution with pdf:

$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.27}$$

The Standard Normal is characterized by $\mu = 0$ and $\sigma = 1$ and it is widely used in many applications due to its simplicity. If $Z \sim N(0,1)$, then $X = \sigma Z + \mu$. For this reason, algorithms are usually developed for the Standard Normal. The Standard Normal has pdf in the form:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{3.28}$$

Several algorithms are available for the generation of normally distributed random numbers due to the centrality of this distribution in statistical applications. One of the first examples uses the Box-Muller method to which a trigonometric transformation is applied. After the generation of two streams of i.i.d. uniformly distributed random numbers, $U_1, U_2 \sim U(0,1)$, the transformations to be applied are $X_1 = \sqrt{-2log(U_1)}cos(2\pi U_2)$ and $X_2 = \sqrt{-2log(U_1)}sin(2\pi U_2)$. X_1 and X_2 are then i.i.d. Standard Normal. An example of implementation of this algorithm is given in the code below.

LISTING 3.40: Normal Box-Muller

```
%% Normal Box-Muller

clc
clear all

n=100;
u_1=rand(1,n)';
u_2=rand(1,n)';
x_1=zeros(1,n)';
x_2=zeros(1,n)';

for i=1:n
x_1(i)=sqrt(-2*log(u_1(i)))*cos(2*pi*u_2(i))
x_2(i)=sqrt(-2*log(u_1(i)))*sin(2*pi*u_2(i))
end
```

```
16 plot(x_1)
```

This algorithm may not be optimal since the evaluation of trigonometric functions and square roots tend to increase computing times significantly. For this reason, a valid alternative is the rejection method described in the code below:

LISTING 3.41: Normal Rejection Polar method

```
%% Normal Rejection Polar method

clc
clear all

n=100;
v_1=rand(1,n)'*2+1;
v_2=rand(1,n)'*2+1;
r=v_1.^2+v_2.^2;
x_1=zeros(1,n)';
x_2=zeros(1,n)';

for i=1:n
if r(i)>=1
x_1(i)=NaN
x_2(i)=NaN
else
x_1(i)=v_1(i)*sqrt(-2*log(r(i)^2)/r(i)^2)
x_2(i)=v_2(i)*sqrt(-2*log(r(i)^2)/r(i)^2)
end
end
```

3.2.18 Pareto Distribution

The Pareto (or Lomax) distribution is a continuous distribution with pdf:

$$f(x;\alpha,\lambda) = \alpha\lambda(1+\lambda x)^{-(\alpha+1)} \tag{3.29}$$

where $x, \alpha, \lambda > 0$. α is called shape parameter, λ is called scale parameter.

Pareto generators are mostly based on the inverse transform method. Without loss of generality, we consider the case $\lambda = 1$ [4]. In this case, if we have $U \sim U(0,1)$, then we need to apply the transformation $X = U^{-(1/\alpha)} - 1$:

LISTING 3.42: Pareto 1

```
%% Pareto 1

clc
clear all
```

[4] The Pareto is a scale family of distributions.

```
5
n=100;
7 a=2;

9 u=rand(1,n)';
x=zeros(1,n)';
11
for i=1:n
13 x(i)=u(i)^(-1/a)-1
end
```

Another algorithm starts from the generation of deviates form an Exponential with parameter 1, if $Y \sim Exp(1)$, then $X = e^{Y/\alpha} - 1$:

LISTING 3.43: Pareto 2

```
%% Pareto 2
2
clc
4 clear all

6 n=100;
a=2;
8 y=exprnd(1,1,n)';
x=zeros(1,n)';
10
for i=1:n
12 x(i)=exp(y(i)/a)- 1
end
```

3.2.19 Student-t Distribution

The Student-t (or t) is a continuous distribution with pdf:

$$f(x;\nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\nu/2)}(1+\frac{x^2}{\nu})^{-(\nu+1)/2} \tag{3.30}$$

where $x \in \mathbb{R}$ and $\nu > 0$. ν is called number of degrees of freedom[5]. The Student-t has some important relationships with other random variables.

First, given a Standard Normal Z, $Z \sim N(0,1)$ and a Chi-square with ν degrees of freedom Y, $Y \sim \chi^2_\nu \equiv Gamma(\nu/2, 0.5)$, if these two random variables are independent, then we have that :

$$X = \frac{Z}{\sqrt{Y/\nu}} \sim t_\nu.$$

Based on this important relationship, it is possible to generate a Student-t via a Standard Normal and a Chi-square:

[5]Note that this number is not necessarily an integer.

LISTING 3.44: Student-t via Chi Square and Standard Normal

```
%% Student-t  Chi-square-NormaleSt

clc
clear all

n=100;
v=5;

z=randn(1,n)';
y=gamrnd(v/2,0.5,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=z(i)/sqrt(y(i)/v)
end
```

Another important property is that, if $X \sim t_{2\alpha}$ then the random variable B has a Beta distribution where,

$$B = \frac{1}{2}(1 + \frac{X}{\sqrt{2\alpha + X^2}}) \sim Beta(\alpha, \alpha).$$

Then, once a random stream of numbers distributed as $B = Beta(\nu/2, \nu/2)$ is available, it is only necessary to apply $X = \sqrt{\nu}\frac{B-0.5}{\sqrt{B(1-B)}}$:

LISTING 3.45: Student-t via Beta

```
%% Student-t  Beta

clc
clear all

n=100;
v=5;

y=betarnd(v/2,v/2,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)= sqrt(v)*(y(i)-0.5)/(sqrt(y(i)*(1-y(i))))
end
```

Another possibility is to apply a ratio of uniform method (proposed by Kinderman and Monahan in 1980):

LISTING 3.46: Student-t Ratio of Uniform

```
%% Student-t  Ratio of uniform

clc
```

```
clear all
n=100;
v=5;

z=rand(1,n)';
u=rand(1,n)'*2*sqrt(v)+sqrt(v);
x=zeros(1,n)';
w=z.^(1/v);

for i=1:n
if w(i)^2+u(i)^2/v<=1
x(i)=u(i)/w(i)
else x(i)=NaN
end
end
```

Finally, three alternatives of the polar method are also available:

LISTING 3.47: Student-t Polar 1

```
%% Student-t  Polar 1

clc
clear all

n=100;
w=5;
u=rand(1,n)';
v=rand(1,n)';

t=u.*2*pi;
r=sqrt(w*(v.^(-2/w)-1));
x=r.*cos(t);
y=r.*sin(t);
```

LISTING 3.48: Student-t Polar 2

```
%% Student-t  Polar 2

clc
clear all

n=100;
q=5;
x=zeros(1,n)';
u=rand(1,n)'*2+1;
v=rand(1,n)'*2+1;
w=u.^2+v.^2;

for i=1:n
```

```
14 if w(i)>1
   x(i)=NaN
16 else x(i)=sgn(u(i))*sqrt((u(i)^2/w(i))*q*(w(i)^(-2/q)-1))
   end
18 end
```

This last method was proposed by Bailey (1994):

LISTING 3.49: Student-t Polar Bailey

```
1 %% Student-t Rejection Polar method

3 clc
  clear all
5
  n=100;
7 ni=4;
  v_1=rand(1,n)'*2+1;
9 v_2=rand(1,n)'*2+1;
  r=v_1.^2+v_2.^2;
11 x=zeros(1,n)';

13 for i=1:n
   if r>=1
15 x(i)=NaN
   else x(i)=v_1(i)*sqrt((ni*(r(i)^(-8/ni)-1)/r(i)))
17 end
   end
```

3.2.20 Uniform Distribution

The Uniform is a continuous distribution with pdf:

$$f(x;a,b) = \frac{1}{b-a} \tag{3.31}$$

where $x \in [a,b]$. The generation of Uniform random deviates in (a,b) follows from the generation of uniformly distributed random numbers in $(0,1)$. For those techniques please refer to 2. If $U \sim U(0,1)$ in order to generate $X \sim U(a,b)$ it is only necessary to apply the transformation $X = a + (b-a)U$, as in the code below:

LISTING 3.50: Uniform(a,b)

```
   %% Uniform(a,b)
2
   clc
4  clear all

6  n=100;
   a=2;
```

```
b=3;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=a+(b-a)*u(i)
end
```

3.2.21 Wald Distribution

The Wald (or Inverse Gaussian distribution) is a continuous distribution with probability distribution function in the form:

$$f(x;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}} e^{-\frac{\lambda(x-\mu)^2}{2x\mu^2}} \tag{3.32}$$

where $x, \mu, \lambda > 0$. μ is called location parameter and λ is called scale parameter. A widely used algorithm for the generation of Wald random numbers is the following:

LISTING 3.51: Wald

```
%% Inverse Gaussian MSGH method

clc
clear all

m=4;
l=2;
n=100;
z=rand(1,n)';
y=z.^2;
x_1=m+(m^2*y)/(2*l)-m/(2*l)*sqrt(4*m*l*y+m^2*y.^2);
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
if u(i)<=m/(m+x_1(i))
x(i)=x_1(i)
else x(i)=m^2/(x_1(i))
end
end
```

3.2.22 Weibull Distribution

The Weibull is a continuous distribution with probability distribution function in the form:

$$f(x;\alpha,\lambda) = \alpha\lambda(\lambda x)^{\alpha-1} e^{-\lambda x^{\alpha}} \tag{3.33}$$

where $x \geq 0, \alpha, \lambda > 0$. α is called shape parameter and λ is called scale parameter.

The easiest way to generate a Weibull distribution is by the inverse method, where, once $U \sim U(0,1)$ is available, the transformation $X = (-ln(U))^{\frac{1}{\alpha}}$ is performed.

LISTING 3.52: Weibull

```
%% Weibull

clc
clear all

n=100;
a=1.2;
b=1;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=(-log(u(i)))^(1/a)
end
```

A special case of the Weibull is when $\alpha = 2$ and $\lambda = \frac{1}{\sigma\sqrt{2}}$. In this case the distribution is called Rayleigh and a useful generation algorithm is based again on the inverse method where $X = \sigma\sqrt{-log(U)}$

LISTING 3.53: Rayleigh

```
%% Rayleigh Inverse CDF

clc
clear all

n=100;
b=1;
a=2;
s=sqrt(2/b);
u=rand(1,n)';
r=zeros(1,n)';

for i=1:n
r(i)=s*sqrt(-log(u(i)))
end
```

Chapter 4

Value at Risk

After having described Montecarlo techniques, we turn now to their most important application to risk management: Value at Risk (VaR). VaR is an extensively used risk measure that has recently received significant regulatory attention. While it is primarily defined with respect to market risk, it should be noted that it can also be applied to other categories of risk (i.e., credit risk). In this chapter, the concept of VaR and its applications to risk management will be introduced. This implies a focus on the necessary inputs and computational methods. The object of the analysis will be the three most common approaches to VaR computation: Variance-Covariance, Historical and Montecarlo simulations.

4.1 Market risk measurement

Financial and non-financial firms deal every day with multiple categories of risk[1]. In general terms, risk can be defined as the uncertainty connected to the future outcome of an investment. For financial firms, one of the main sources of risk that naturally arises from their activities is market risk. Market risk is formally defined by the Basel Committee for Banking Supervision (Minimum capital requirements for market risk, January 2019) as the risk of losses arising from movements in market prices. This risk then includes default risk (specifically, counterparty risk), interest rate risk, credit spread risk, equity risk, foreign exchange risk (FX) and commodity risk. Movements and co-movements in those market factors can significantly erode the value of a portfolio of financial assets. The banking and trading book share this category of risk.

After the global financial crisis, market risk started receiving growing attention both by regulators and by internal risk managers inside financial firms. The reason is that growing financial markets, complexity of new instruments, integration of global markets and technological improvements have contributed to increase the sources and complexity of market risk dramatically.

[1] Those categories of risk are usually described as credit risk (the risk to be unable to collect a credit due to the inability to repay of the counterparty), operational risk (risk of loss resulting from inadequate or failed internal processes, people and systems or from external events) and liquidity risk (the risk that a company or bank may be unable to meet short term financial demands).

In this context, the need for proper measurement of market risk is critical for many reasons. First, it is internally necessary, especially in big institutions, to be able to communicate quickly and effectively the magnitude of risk the firm is facing both in the short and long run. In this respect, simple and understandable measures of risk are needed in order to be able to inform management. An appropriate risk measure is also needed to set desired limits to a bank's risk exposure. Without a quantitative measure, it is, in fact, not possible to define the risk appetite of the firm. Finally, sticking to the firm perspective, risk measures associated to return and performance figures are necessary in order to evaluate how the firm is acting and, consequently, for resource allocation purposes. Risk measures are indeed also necessary for regulators and supervisors. Those figures need in fact to be communicated to competent supervisors so that they have a clear picture related to both micro and macro stability.

For all these reasons, literature has extensively focused on measures that are able to capture the degree of exposure to market risk of a firm or a portfolio. The most basic class of market risk measures are nominal measures. This class of indicators is extremely simple and it is not based on any computation (ex. 100 mil. EUR exposure to GBP or 200 mil. EUR exposure to LIBOR). In all these cases, risk is defined in face (or nominal) amounts. Those rudimental measures of risk are, unfortunately, optimal only if the bank's activity on financial markets is limited. If this is the case, in fact, the ease of computation should be privileged compared to the information produced by the measure. In all other cases, notional values are always not enough to build a sound market risk management strategy. What risk managers need in their day-to-day activity, in fact, is a risk measure that is informative in terms of potential losses, because the amount of the exposure is not informative if it is not coupled with a loss (or variability) figure. Moreover, notional amounts are defined only with respect to one exposure and thus are difficult to compare across asset classes. Comparisons and aggregations are indeed crucial in the activities performed by risk managers. For all these reasons, different metrics for market risk should be introduced.

Tons of market risk measures are available if one wants to assess the uncertainty connected to market movements. A first example is volatility. Volatility estimates the tendency of a random variable to change over time. It is usually estimated based on historical time series. If coupled with returns, volatilities give an essential idea about the performance of an investment. A related but in fact distinct class of measures is sensitivity. Sensitivity estimates the tendency of an instrument to move together with some market factor (interest rates, exchange rates, stock market indexes, commodity prices). Those measures are at the heart of portfolio construction. In the contest of market risk measures, it is necessary to introduce the concept of value at risk.

Traditional market risk measures have the drawback that they define risk in relative terms, thus accounting both for upside and downside potential. However, risk managers (and most of all investors) tend to care only about the downside potential

of risk. An unexpectedly high return, in fact, is for sure not a problem. Moreover, traditional risk measures are usually expressed in percentage terms. This implies that it is not possible to quantify in absolute values how much the portfolio may lose. Value at risk tries to solve those issues by providing risk managers and investors with an absolute amount that may be expected to be lost over a specified time period. We will then proceed with a formal characterization of value at risk.

4.2 Value at Risk

As stated above, Value at Risk is a widely used market risk measure. This measure is relevant because it summarizes in a single absolute number the total risk of a portfolio. VaR can be defined as the absolute maximum loss in value of an asset or portfolio of (financial) assets over a defined horizon with a specified probability (confidence level). Putting it differently, VaR gives information about how much a portfolio is expected to lose in absolute terms with a given probability and over a time horizon, assuming that the composition of the portfolio remains unchanged over that horizon. For example, if we know that, over a one-day holding period and at 99 % confidence level, the VaR of a portfolio or financial instrument is 50 mil. EUR, it means that there is 1% probability that the value of the asset or portfolio will drop by more than 50 mil. EUR over a day. In general terms, given a confidence level $1 - \alpha$ and a holding period t, the VaR is the value of loss that we expect to be exceeded with probability α during the period t. The confidence level $1 - \alpha$ is usually thought as denoting "normal" market conditions, while α represents unexpected (thus unlikely) market movements. VaR is then a measure that summarizes the absolute maximum amount of loss for a portfolio in normal market conditions. In more precise statistical terms, VaR represents a quantile of the distribution of profits and losses (P&L) of the portfolio.

VaR is a crucial measure in the assessment of the institution's exposure to market risk. On the one hand, institutions themselves use this measure in order to evaluate their risk profile. This implies that VaR can be used both ex-ante and ex-post. Institutions tend in fact to set limit values to VaR so to be able to manage their desired exposure to adverse market movements. Ex post, VaR is a widely used measure for portfolio evaluation. On the other hand, regulators and supervisors have historically heavily relied on value at risk. The Basel Committee for Banking Supervision (1996), for example, imposes that capital requirements for market risk are estimated using Value at risk models. In this cornerstone, banks are set free to choose which model to use in the estimation of VaR. It is then clear that banks and financial institutions have a clear incentive in developing accurate models for the estimation of value at risk in order to avoid underestimations or overestimations of capital requirements.

VaR has just been defined as the absolute maximum loss in value of an asset or portfolio of assets over a defined horizon with a specified probability. The primary input that is needed is then the estimated distribution of profits and losses (P&L)

over the holding period. This enables us to estimate the absolute maximum loss. The result will be an amount denominated in the relevant currency. Even if this amount represents a loss, it is usually denoted in absolute terms, and thus a negative VaR implies that, even in an adverse scenario, a gain is expected. What is usually available to risk managers is the value of the portfolio of assets, P, over the relevant period $[t, T]$. In order to obtain the P&L input, one has to apply $P\&L_{t,T} = P_T - P_t$. Once the distribution of this variable is known, it is possible to implement the model and come up with an estimation of VaR.

It is indeed also possible to evaluate VaR is relative terms. This is especially convenient when it is necessary to draw comparisons among portfolios of different sizes. In this case, the input quantity is not the P&L distribution anymore, but the holding period log return. Given the holding period $[t, T]$ and the value of the portfolio P, the holding period log return is defined as $r_{t,T} = ln(\frac{P_T}{P_t})$. In this case, if 1-day VaR at 5% confidence level is 2 %, it implies that with 99% probability the portfolio will have a return higher than -2% over a 1-day horizon.

4.2.1 VaR components

The definition of VaR has already underlined its most important components. Those elements can also be seen as the necessary inputs for the computation of VaR.

The first element is the confidence level α. α represents the probability level starting from which we may observe losses that are higher than what predicted by the VaR. It is common in practice to set $\alpha \in [1\%, 5\%]$. However, there is no optimal value and the choice critically depends on the characteristics of the portfolio and institution. The choice of the confidence level has serious consequences if VaR is used for the evaluation of capital requirements. Capital requirements, in fact, have to reflect the degree of risk that the institution is undertaking. In this respect, the confidence level should reflect the degree of risk aversion of the institution. Consequently, as the level of risk aversion of the institution increases, α should decrease, reflecting the fact that the amount of capital needed to cover potential losses increases. If the aim is to compare VaRs of different institutions or portfolios, the choice of α is then not critical, but it has to be recalled that, in order to be able to compare VaR figures, they should be obtained for the same confidence level. This conclusion stems from the fact that VaR naturally increases as α decreases. It is, in fact, intuitive that losses that may occur with a probability of 1% are higher than the ones that may occur with probability 5%. Even if usually $\alpha \in [1\%, 5\%]$, there are some circumstances in which different values may be used. Specifically, high values such as 10% or 15% may be used to set VaR limits. Conversely, extremely low values (0.01%) may be used for economic capital allocation purposes.

The second critical element is the holding period t. This component is critical because, once the holding period is set, we are assuming that the portfolio will remain unchanged for the whole length of t. This explains why the typical holding

period is set to be 1-day. If it is longer, in fact, the assumption may be violated, especially for assets that are traded frequently. In abstract terms, the holding period may be chosen to be of every length. However, common choices are 1-day, 10-days, 1-month, 1-year. It is possible to define a relationship among VaR measures computed for different holding periods. Define VaR_1 as the VaR computed for a 1-day holding period. If we want to compute VaR with holding period t then we have to apply $VaR_t = VaR_1\sqrt{t}$. Note that this relation is only approximate. One of the most important considerations when choosing the holding period is liquidity. We have stated that a good holding period choice is 1-day since it is reasonable to assume that a portfolio will remain unchanged in its composition over 1-day. However, when a short holding period is used, we are assuming that markets are so liquid that positions can be closed out without incurring in significant losses. This assumption may not always be reasonable and thus it is convenient to compare VaRs with short holding periods to VaRs calculated with longer holding periods (such as one month). In general, it is appropriate to use long holding periods (1-month to 1-year) if the VaR exercise aims to plan capital allocation or if the portfolio contains illiquid assets.

The last element is probably the most discussed one. When computing VaR, it is necessary to specify the distribution of P&L (or in the same way of log returns). Only in this way it is possible to determine the maximum potential loss. Recall in fact that VaR is nothing but a quantile of the distribution of profits and losses (or returns) of the portfolio. To state in more rigorous terms, given the time horizon t, the confidence level α and the distribution of P&L, VaR is that number such that the probability of incurring in higher losses is equal to α or, conversely, that number such that the probability of incurring in lower losses is equal to $1-\alpha$:

$$P(P\&L_t \leq VaR) = 1 - \alpha.$$

Different models and different banks use different assumptions for the distribution of P&L underlying every VaR model. These different choices will be extensively described in this chapter and in 5.

4.2.2 Common steps: risk mapping

Methodologies for the computation of value at risk can broadly be divided into two categories. The first one comprises local valuation (or parametric) methods. These models evaluate the portfolio at date 0 and use derivatives in order to evaluate future movements. The primary example is the delta-normal method in which normal distribution assumed. The second category are full valuation methods. Those methods evaluate the portfolio in different future scenarios and they are mainly simulative models[2]. The two main examples of this category are historical simulation and

[2]Suhobokov, Alexander. (2007). Application of Monte Carlo simulation methods in Risk Management. Journal of Business Economics and Management. 8. 165-168.

the Monte Carlo method. These three models will be separately described in the following pages. Now we will focus on some similarities.

Even if models for the computation of VaR are different in many respects, they share a common set-up and infrastructure. It is usually thought, in fact, that in order to evaluate VaR, a three-step approach should be followed. The first necessary step is the evaluation of portfolio value at time 0. There are no significant differences among the three techniques in this respect. The second step is the critical one. In this phase, the future distribution of portfolio P&L (or returns) should be estimated over the holding period. Here differences among the models arise because each model makes different assumptions about the relevant random variable. Finally, once the (estimated) distribution is available, it is only necessary to select the right percentile. This gives the estimated value for VaR. Even in this final step, there is no relevant difference among the methods.

It is clear that the second step is especially critical, because wrong assumptions about future portfolio value may lead to a biased estimation of VaR. This is especially dangerous when VaR is used for capital allocation purposes. Irrespective of the assets that compose the portfolio, risk managers should balance the need for simple and understandable techniques with the need for a realistic representation of reality carefully. Even if the future is unknown, it is possible to leverage on some well-known facts about asset values and returns. First, even if normality assumption for asset returns is appealing for computational reasons, asset returns do not tend to display this behavior. Instead, returns distributions are characterized by fatter tails than predicted by the Normal distribution (Leptokurtotic distribution). In addition, when compared to the Normal distribution, returns tend to be negatively skewed. Finally, even if in most models it is assumed that volatilities and correlations are stable over time, they likely change. For example, the pattern of correlations seems to change during crisis periods significantly. Model developers should keep all these observations in mind when deciding the main assumptions for the definition of future portfolio value evolution.

In order to be able to define the assumption on the distribution of portfolio P&L (or returns) it is necessary to understand which the drivers of the value of the portfolio are. This activity is complicated but crucial because only if the definition of the drivers is granular, the estimated values are reliable. The selection of all the market factors that are able to influence the value of the portfolio and of the mathematical relationship that defines how this happens is referred to as risk mapping. Consequently, whatever the model to compute VaR is, a first necessary preparatory step is the identification of simple market factors that influence the value of the portfolio. Without this preliminary step, the computation of VaR is impossible due to computational burdens. Risk mapping usually applies a top-down approach. This implies that, given the overall complex portfolio, it should be divided into several simpler instruments, whose relation with primary market factors is clear and (hopefully) simple in mathematical terms. This step is indeed extremely complicated because

financial instruments as swaps, options, loans and exotics are influenced by an incredibly high number of market factors that are difficult to account for (for example because the payoff is non-linear). The process of risk mapping is especially important in parametric models. In these models, in fact, it is necessary to express in closed form the dependence of the portfolio on the primary market factors. In simulative models (Montecarlo and historical simulation), the portfolio is revaluated in every simulated scenario and thus a comprehensive value for the portfolio is immediately obtained. Nevertheless, the process of risk mapping is anyway necessary in order to make the correct assumptions about the distribution to choose (in the Montecarlo case) and about the formula to use in order to evaluate the portfolio at each scenario (in both cases).

We will consider an example in order to explain what risk mapping actually is and how it works in practical terms. This example will be used throughout this chapter. The instrument object of analysis is a forward contract. It will shortly be clear that this instrument can easily be seen as a portfolio. Assume a US-based bank entered into an exchange rate forward contract (from now on we will call this instrument FX). On the delivery date, the American bank will deliver 6 mil. USD and will receive 10 mil. GBP. We will assume that today's date is the 15th of January and that the contract has delivery date 15th of April. Accepting the 360 convention (i.e. 12 months with 30 days), the remaining time to delivery is 90 days, $T = 90$ days $= \frac{90}{360} = 0.25$ year. Our goal is to define the current value of this contract. To do so, it is necessary to define it as the sum of some simpler components, that are all the factors that are able to influence the FX. Start by analyzing the cash flows connected to this contract. The owner will receive 10 mil. GBP and pay 6 mil. USD on the delivery date. Then the present value (mark-to-market) of the contract is simply $PV(10mil.)GBP - PV(6mil.)USD$. Remember that the value of the investment shall be assessed with respect to the US investor, and thus the currency should be USD. Then, we can conclude that in order to compute this mark-to-market value, we need three figures: the exchange rate and the two discount rates (one per currency). The first one (exchange rate) is the spot exchange rate quoted as USD vs. GBP. In order to compute today's mark-to-market value, we need the current spot exchange rate, S. This is needed in order to convert the amount received in GBP in USD. In order to discount future cash flows, we also need interest rates. The two applicable interest rates are the US one and the UK one. The best choice would be three-months interest rates, r_{USA} and r_{UK}. We have concluded the risk mapping exercise because we have decomposed a complex contract (the forward) in two basic instruments that are directly and clearly influenced by observable market factors. The first instrument is a bond with face value of 10 mil. GBP, the second one is a short position on a bond with face value 6 mil. USD, both with maturity three months. The value at time 0 (today) of the contract is then:

$$S\frac{10GBP}{(1+r_{UK})^{0.25}} - \frac{6USD}{(1+r_{USA})^{0.25}}.$$

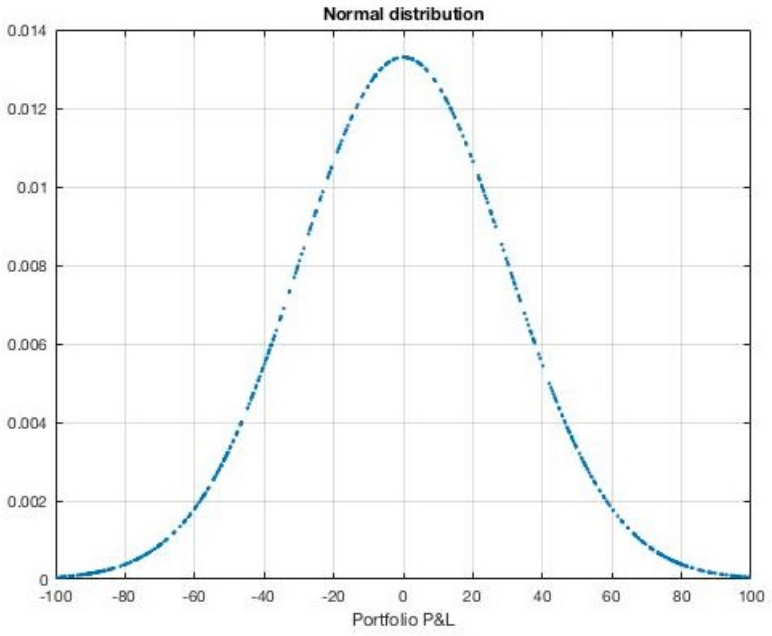

FIGURE 4.1: Normal Distribution of P&L

In order to compute this quantity it is only necessary to observe the current value of the three market factors S, r_{USA}, r_{UK}.

If the market factors able to influence the value of the portfolio have been identified, then it is necessary to make assumptions about their future behaviour, so to be able to assess the value of the portfolio in the future. As stated above, parametric and simulative models significantly differ in the way of making these assumptions. Differently, the risk mapping phase is common to the three approaches.

4.2.3 A simple example

Before the explanation of the various techniques for the computation of value at risk and in order to understand what the concept of value at risk actually means, we will introduce a merely explicative example.

Assume that the distribution of portfolio $P\&L$ over the holding period of 15 days is known. This is of course an unrealistic assumption if the risk mapping exercise has not yet been performed. Assume further that the distribution is as represented in Figure 4.1.

We have defined the value at risk as the absolute maximum loss in value of an asset or portfolio of assets over a certain horizon with a specified probability. If, for example, we set $\alpha = 0.05$ this means that we are interested in the maximum loss that will be incurred with 95% probability, i.e., there is a 5% chance, on average, that the incurred loss will be higher than what predicted by the value at risk. In our case, VaR is simply equal to the value of this (Normal) distribution that corresponds to the 95% percentile. It is easily found via Matlab using the code VaR=prctile(x,0.95), where x is the vector that represents the P&L.

4.3 Parametric approach

The parametric approach is broadly considered the most simple model for the computation of value at risk. The most important assumption of this model is that it is not necessary to re-evaluate the portfolio in the future because its variance[3] is sufficient for the computation of value at risk. As noted in the previous paragraph, portfolios held (for trading) by financial institutions can be incredibly complex. For the application of this method, this would imply a heavy computational burden. For this reason, a portfolio is usually decomposed in the sum of simple instruments or market factors (risk mapping). Those basic instruments will become the basis for the computation of portfolio variance (which depends on the composition of the portfolio and on variances and covariances of the instruments). We will use the variance-covariance method as an example of a parametric approach.

The variance-covariance method is based on some critical assumptions. Those assumptions are incredibly restrictive and they can be viewed as the reason for the limited use of this model. First, the variance-covariance method assumes that the joint distribution of the market factors that influence the portfolio is Normal. The reason is mainly to simplify computations. However, this assumption does not seem to be always appropriate. In the case of stock prices, for example, it is well known that their distribution has fatter tails than predicted by the Normal. The second assumption is that all market factors have zero autocorrelation, meaning that the past does not influence current values. Finally, it is assumed that correlations among market factors are constant. This assumption is especially relevant since it is well known that the pattern of variances changes significantly during periods of market turmoil (such as a crisis). Since this method is not able to account for these changes, it may lead to a significant underestimation of VaR (and thus of the level of risk in general).

4.3.1 Computation

The computation of value at risk according to the variance-covariance approach can be divided into some steps. Those steps are always the same, irrespective of the complexity of the portfolio to be considered. Of course, as the complexity of the portfolio increases, the procedure will be more elaborate and time-consuming.

Step 1. As expected, the first step is always risk mapping. This means that, given a portfolio or financial instrument, it should be decomposed in simpler components that are directly influenced by market factors. It is important to know that, especially for this method, risk mapping should be performed even if the instrument to be evaluated is already simple. For example, consider a 3 years coupon bond with annual coupons. It may well be considered as a simple and standard instrument. However, risk mapping can be performed even in this case, by representing the coupon bond as the sum of three zero-coupon bonds. The first ZCB has maturity one year and

[3] Of course, covariances are taken into account too, as will be shown in the following pages.

face value equal to the first coupon, the second ZCB has two years maturity and face value equal to the second coupon, the third ZCB has three years maturity and face value equal to the sum of the third coupon and the face value of the original three years coupon bond. Once this exercise is done, it is easy to see that this coupon bond is not only influenced by the yields of similar coupon bonds, but also by the yields of the three ZCBs. Of course, risk mapping involves not only the identification of the factors that are able to influence the portfolio, but also the definition of the formula that links those values to the portfolio. In the case of the coupon bond (C) and calling a ZCB with maturity i ZCB_i, this is simply $C = ZCB_1 + ZCB_2 + ZCB_3$.

Step 2. Once the market factors are available, it is necessary to make some assumptions about their distribution. In the delta-normal method (which is one of the most widely used) the assumption is that the distribution of the market factors over the relevant horizon is Normal. If we define $m(t, t+\Delta)$ to be the distribution of the market factor over the horizon Δ then we are assuming that $m(t, t+\Delta) \sim N(\mu, \sigma^2)$. If we accept the assumption that every market factor has a Normal distribution[4] then in this step the only goal is to estimate the parameters of the distribution for each market factor. In the case of normality, the parameters to be estimated are only two, namely the mean μ and the standard deviation σ. Recall that, given a Normal distribution $m(t, t+\Delta) \sim N(\mu, \sigma^2)$, the mean is defined as

$$\mu = E(r(t, t+\Delta))$$

and the standard deviation as

$$\sigma = \sqrt{Var(r(t, t+\Delta))}.$$

If the risk factors are more then one, then it is also necessary to estimate their correlation. Given two market factors $m_1(t, t+\Delta) \sim N(\mu_1, \sigma_1^2)$ and $m_2(t, t+\Delta) \sim N(\mu_2, \sigma_2^2)$, the correlation coefficient between these two variables, ρ_{m_1,m_2} is defined as:

$$\rho_{m_1,m_2} = \frac{Cov(m_1, m_2)}{\sigma_{m_1}\sigma_{m_2}}.$$

In practice, those parameters are estimated based on historical values of the market factors. While it may be acceptable to use history to estimate standard deviations, this may not be reasonable for means and correlations. History may not repeat and, especially during crisis periods, the VaR estimation based on wrong parameters may not be reliable anymore.

Step 3. At the end of Step 2, all information related to individual market factors is known. The next step is to aggregate the information so to know the parameters of the portfolio or instrument. Those parameters will be the necessary input for the computation of VaR. In order to be able to compute portfolio mean and

[4] It has already been established that this assumption may not be reasonable in many stances.

variance, normality assumption is crucial. It is well known that the sum of normally distributed random variables is again a Normal random variable. Assume that the relevant portfolio can be described as the algebraic sum of n market factors, $m_1, ..., m_n$. Assume further that all those market factors are normally distributed: $m_i(t, t+\Delta) \sim N(\mu_i, \sigma_i^2), \forall i$. The weights (in absolute or relative terms) of each market factor i are defined as w_i. Then, the distribution of the portfolio's P&L (or returns) is Normal with mean

$$\mu_P = \sum_{i=1}^{n} \mu_i$$

and standard deviation:

$$\sigma_P = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \sigma_i \sigma_j \rho_{m_i,m_j}}.$$

At the end of this step, we are thus able to determine μ_P and σ_P.

Step 4. Finally, we are in the position to compute the value at risk figure for the instrument or portfolio. Given the time horizon Δ, the mean of the portfolio, μ_P, the standard deviation of the portfolio, σ_P, and the confidence level, α, the VaR according to the parametric approach is given by:

$$VaR(t, t+\Delta) = -(\mu_P + z_{1-\alpha}\sigma_P) \tag{4.1}$$

where $z_{1-\alpha}$ is the $1-\alpha\%$ percentile of the Standard Normal distribution[5].

One necessary consequence of this set-up is that we have critically assumed that the portfolio can be described as the algebraic sum of some basic market instruments. Only in this way, in fact, it is possible to conclude that the distribution of portfolio P&L (or returns) is Normal. An important implication of this is that it is not possible to directly apply this method to portfolios that have non-linear relations with the relevant market factors. In the case of options, for example, some modifications will be needed.

4.3.2 Example

The computation of VaR trough the delta normal approach will be made clear via an example introduced early in this chapter.

Consider the forward contract defined in 4.2.2. Recall that, on the delivery date, the American bank will deliver 6 mil. USD and will receive 10 mil. GBP. The maturity of this contract is $T = 90$ days. It has been shown that the value of this instrument is influenced by three figures: the current spot exchange rate USD vs. GBP, S, and the two three-months interest rates (one per currency), r_{USA} and r_{UK}. This contract can thus be seen as the sum of a long position in a three-months GBP denominated zero-coupon bond with face value 10 mil. GBP and a short position in a

[5]Note that this formula can easily be adapted to the portfolio distribution that is chosen.

three-months USD denominated zero-coupon bond with face value 6 mil. USD. The value at time 0 (today) of the contract is then:

$$S\frac{10GBP}{(1+r_{UK})^{0.25}} - \frac{6USD}{(1+r_{USA})^{0.25}} \tag{4.2}$$

A good starting point is the computation of the current value of the portfolio. In order to calculate it, we need to observe, on the markets, the values for S, r_{USA} and r_{UK}. S is the exchange rate denominated EUR vs GBP, assume it is equal to $S_0 = 1.29$. r_{USA} is the rate on US three-month T-Bill, assume $r_{USA} = 2.3\%$. Finally, r_{UK} is the rate on the three-month English government bonds, assume $r_{UK} = 0.9\%$[6]. Now we have all the necessary inputs to compute the current value of the portfolio according to the formula

$$PV(P) = S_0\frac{10GBP}{(1+r_{UK})^{0.25}} - \frac{6USD}{(1+r_{USA})^{0.25}} = 6.91mil.USD.$$

Now that the current value of the portfolio is known, it is necessary to assume that all the market factors, S, r_{USA} and r_{UK}, are normally distributed. Once we make this assumption, it is necessary to estimate the parameters μ and σ for each of them. As mentioned early, we base this estimation on historical time series.

For the exchange rate USD vs GBP source of data is Banca d'Italia database, the window of estimation is 15/02/2010 to 15/02/2019 and the frequency of data is monthly. The estimated parameters are $\mu_S = 1.49$ and $\sigma_S = 13.34\%$. Figure 4.2 gives an overview of the evolution of this market factor.

The histogram in Figure 4.3 depicts the distribution of the exchange rate. The distribution is far from Normal in the fact that it displays fatter left tails and it is not centered around the mean. This lack of fitting between data and assumptions may have severe consequences on the reliability of VaR estimation.

The second market factor is the USD T-bill rate. For this dataset, the source is Yahoo finance. Frequency of data is monthly and the estimation period is 15/02/2010 to 15/02/2019. The estimated parameters are $\mu_{USA} = 0.042\%$ and $\sigma_{USA} = 0.19\%$. This rather low mean comes from the fact that at the beginning of the dataset yields on T-bills were extremely low, while, in more recent times, we have seen the inversion of the yield curve (i.e. yields on government bonds with longer maturities are lower than government bond rates with shorter maturities). This inversion is a concern when modelling since it is uncertain whether this situation will persist or if the curve will become again normal. Figure 4.4 describes the evolution of US 3-month T-Bill rates. The histogram in Figure 4.5 underlines that the distribution is clearly not close to the Normal, being it concentrated at extremely low values. This pattern depends, as explained above, from the data period.

The third and last market factor is the UK three-month T-bill rate. For this dataset, the source is Bank of England database. Frequency of data is monthly and

[6] All interest rates are annualized.

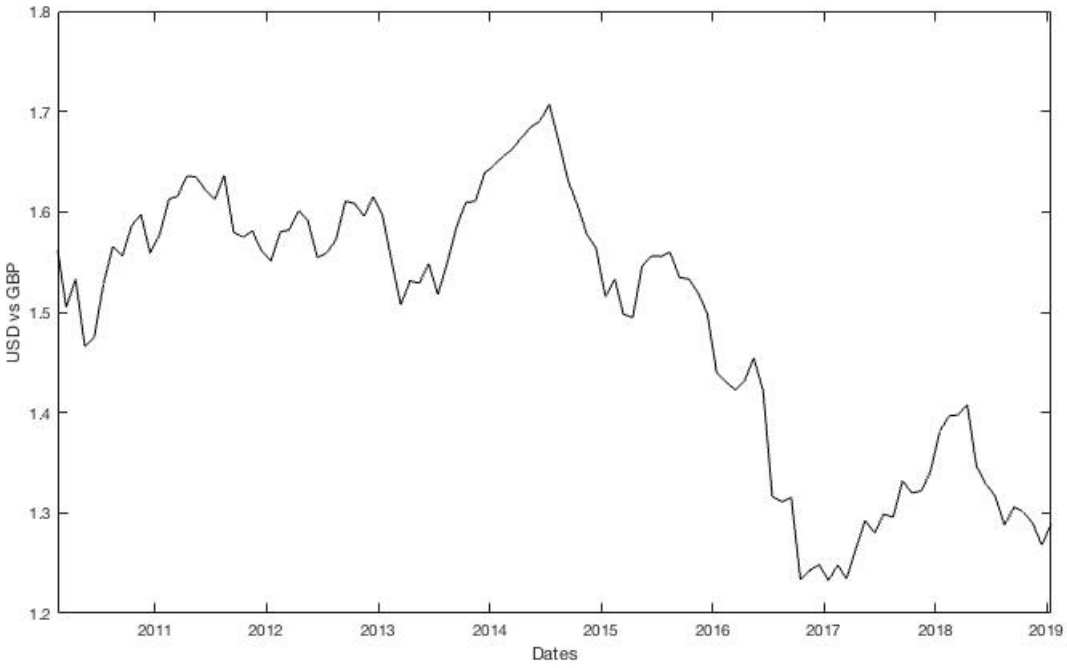

FIGURE 4.2: Evolution of USD vs. GBP exchange rate 2010-2019

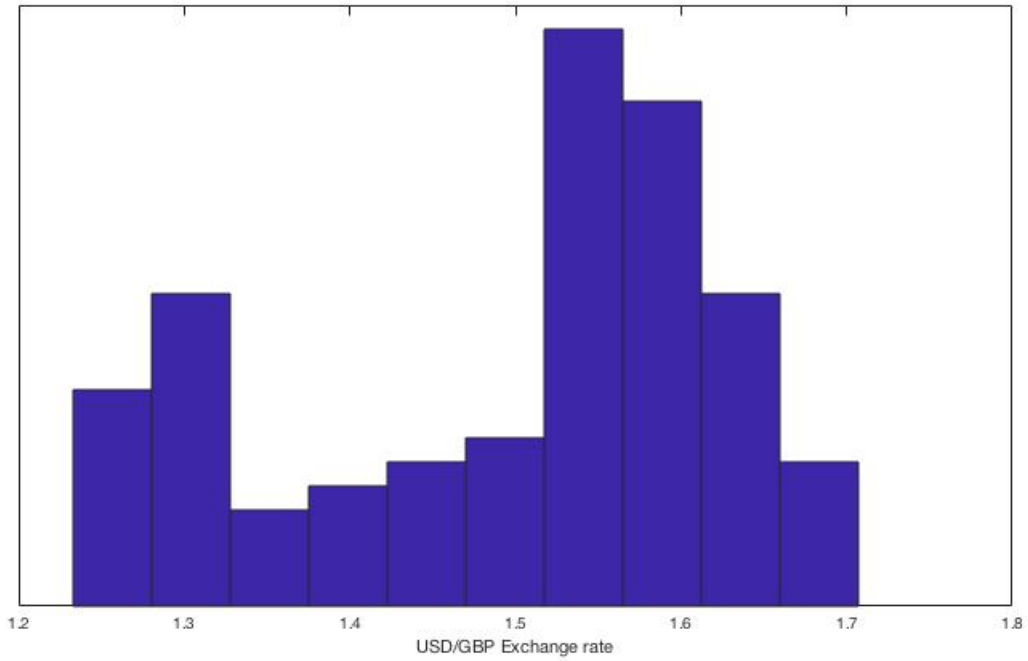

FIGURE 4.3: Distribution of USD vs. GBP exchange rate 2010-2019

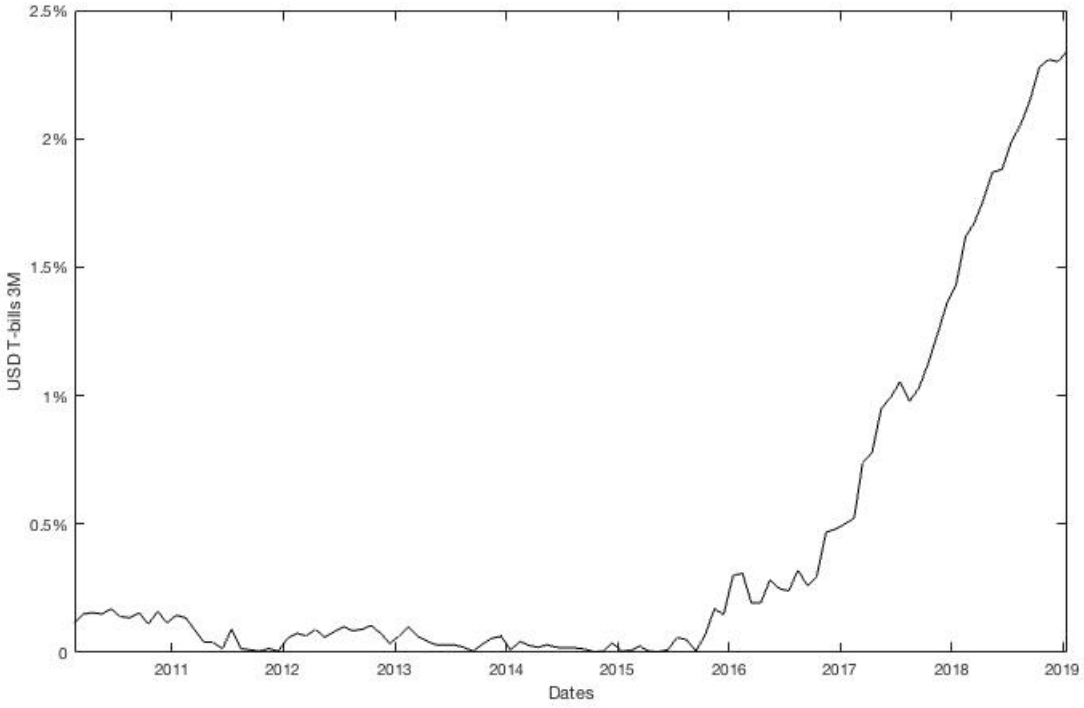

FIGURE 4.4: Evolution of USD T-bills 2010-2019

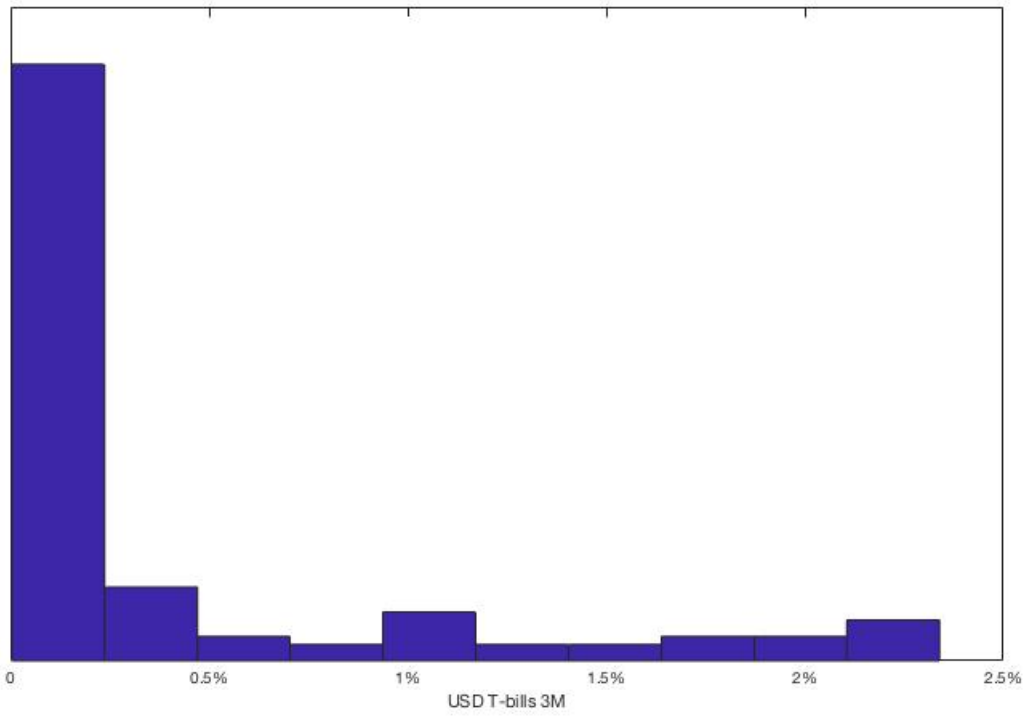

FIGURE 4.5: Distribution of USD T-bill rates 2010-2019

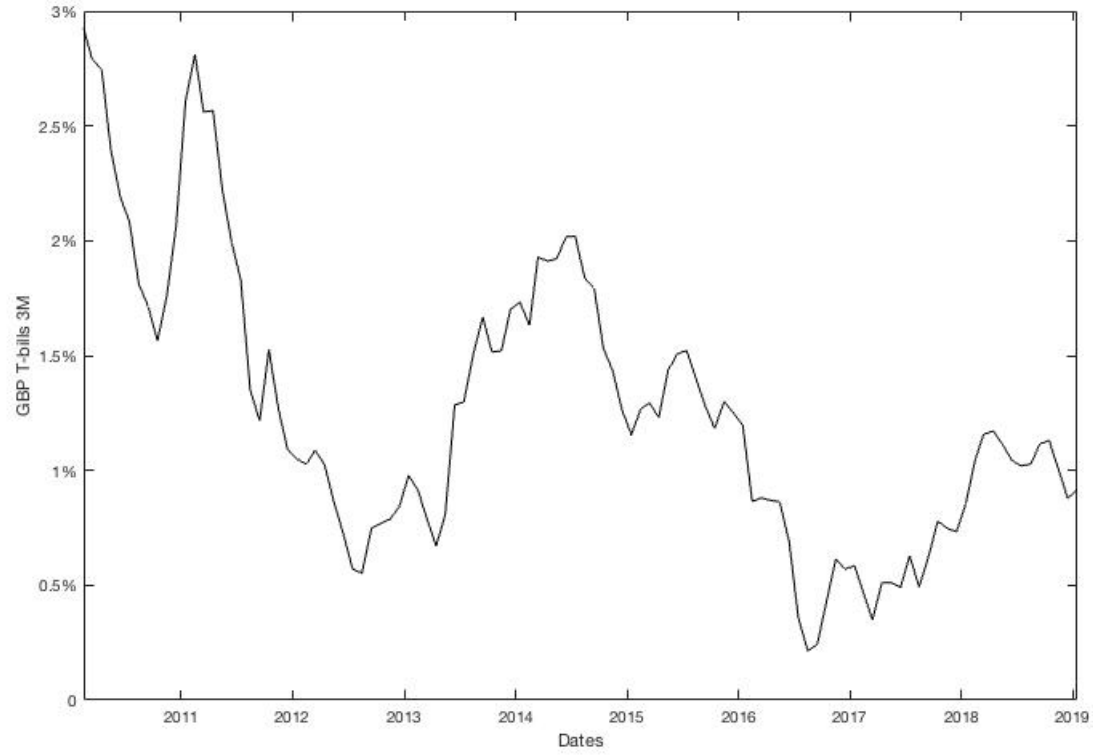

FIGURE 4.6: Evolution of UK T-bills 2010-2019

the estimation period is 15/02/2010 to 15/02/2019. The estimated parameters are $\mu_{UK} = 1.27\%$ and $\sigma_{UK} = 0.62\%$. Figure 4.6 describes the historical evolution of UK 3-months T-bills. The histogram in Figure 4.7 depicts a distribution which is closer to the Normal with respect to the other two but still with fatter tails and less bell-shaped.

The next step is to assume that all these three market factors are normally distributed, $S \sim N(\mu_S, \sigma_S^2)$, $r_{USA} \sim N(\mu_{USA}, \sigma_{USA}^2)$, $r_{UK} \sim N(\mu_{UK}, \sigma_{UK}^2)$, even if we have already underlined that this distribution is not fitted with historical data.

Since we have assumed that this portfolio can be represented as the sum of a long and a short position and thus that the relationship with the market factors is linear, we can conclude that the pay-off of this portfolio has a Normal distribution with mean μ_p and standard deviation σ_p.

In order to come up with an estimation of the portfolio standard deviation, we need to estimate the correlation coefficients among the three variables. Those correlations result to be: $\rho_{S,USA} = -0.71$, $\rho_{S,UK} = 0.62$, $\rho_{UK,USA} = -0.32$.

Applying the formula described in Step 3 above, we conclude that this portfolio is normally distributed as $P \sim N(21.4\%, 17.13\%^2)$.

Now, we have all the ingredients for the computation of VaR according to the following formula:

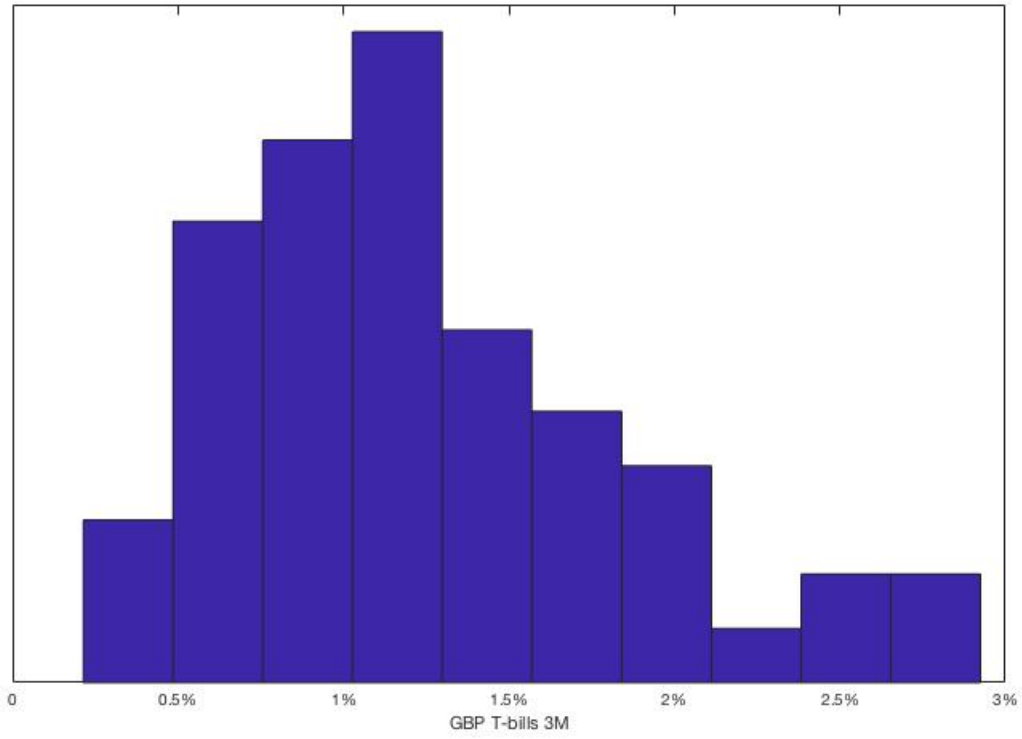

FIGURE 4.7: Distribution of UK T-bill rates 2010-2019

$$VaR(1-month) = -(\mu_p + z_{1-\alpha}\sigma_P).$$

Where $\mu_p = 21.4\%$, $\sigma_p = 17.13\%$ and $z_{1-\alpha} = 1.65$. If we choose $\alpha = 5\%$, the estimated VaR is then USD 0.07 mil.. This is the maximum loss expected to be incurred in 95% of cases.

The code below describes all computations that were performed:

LISTING 4.1: Parametric VaR for Portfolio

```
%% VaR Forward Var/Cov

%initial values

r_usa_0=0.023
r_uk_0=0.009
S_0=1.29
Q_us=6
Q_uk=10
T=0.25 %maturity
P_0=S_0*(Q_uk)/(1+r_uk_0)^T-Q_us/(1+r_usa_0)^T

t1=datetime(2010,1,15)
t = t1 + calmonths(1:108) %generate dates for plot
```

```
%% Exchange rate analysis

S=GBPUSDmontly(3:end)
m_S=mean(S)
s_S=std(S)

plot(t,S)
hist(S)


%% US t-bills

r_usa=USDBillsmonthly(3:end)*0.01

m_usa=mean(r_usa)
s_usa=std(r_usa)*12^(-1/2)
plot(t,r_usa)
hist(r_usa)

%% GBP Bills

r_uk=Ukbillsmonthly(3:end)*0.01

m_uk=mean(r_uk)
s_uk=std(r_uk)
plot(t,r_uk)
hist(r_uk)
%% Portfolio

covS_uk=corrcoef(S,r_uk)
corS_uk=covS_uk(1,2)
covS_us=corrcoef(S,r_usa)
corS_usa=covS_us(1,2)
covus_uk=corrcoef(r_usa,r_uk)
coruk_usa=covus_uk(1,2)

m_port=Q_us*m_usa+Q_uk*m_uk*m_S
s_port=sqrt(Q_us^2*s_usa^2+Q_uk^2*s_uk^2+s_S^2+2*Q_us*Q_uk
*coruk_usa*s_usa*s_uk+2*Q_us*corS_usa*s_usa*s_S+2*corS_uk*Q_uk*s_uk*s_S)

%alpha=5%

VaR=m_port-1.65*s_port %compute portfolio VaR
```

4.3.3 Advantages and disadvantages

From the above description, it is evident that one of the main advantages of the parametric approach is its simplicity. The delta normal method is, in fact, fast and straightforward to compute because it requires only the estimation of means, standard deviations and correlations. If for example, those parameters are immediately available in trusted datasets, it is not even necessary to estimate them.

Not only this measure is easy to compute, but also to explain. It may thus be a right candidate for senior management reporting. However, from its simplicity, some major weaknesses arise.

First, the delta normal approach critically depends on the normality assumption for all market factors and consequently, for the portfolio. As already pointed out, this assumption may not be reasonable in many cases. This is a significant weakness of the model because unjustified normality assumption may lead to an underestimation of VaR. This comes from the fact that empirical distributions of returns tend to have more outliers than predicted by the Normal. More complex distributions may be more fitted to data, but the use of these distributions increases the complexity of computations and thus the main advantage of this method. In order to assess whether this procedure is appropriate statistical tests for normality are advisable.

Another issue is connected to standard deviations. As described above, standard deviations of market factors are usually estimated based on historical time series. As every estimation based on historical data, standard deviations come with associated standard errors. If those errors are significant, it may not be possible to use them for the calibration of VaR. Moreover, estimation techniques assign the same weight to every observation while, in some cases, it may be desirable to overweight recent times. A connected issue comes from the fact that this model does not allow for time-varying standard deviations.

4.4 Historical simulation approach

The Historical simulation approach is another well-known technique for the computation of VaR. It is referred to as a non-parametric technique since, in the application of this model, no assumption regarding the shape or parameters of the market factors (and consequently of the portfolio) is made. The core idea is that history will exactly repeat itself and thus, past distribution of market factors are the best approximation for the future. Banks tend to do extensive use of this approach due to its simplicity. Since normality is not assumed, one positive aspect of this method is that it is able to account for fat tails and kurtosis, differently from the variance-covariance approach.

When the historical simulation method is applied, time series of the market factors are combined to produce the distribution of portfolio $P\&L$. Portfolio distribution at each date is, in fact, computed based on historical changes in the market factors. In more specific terms, percentage changes (over the relevant horizon) of

market factors are computed based on historical data. Once these changes are available, they are applied to the current value of the portfolio so to be able to reprice it. If we observe market factor values for $n+1$ days, then n percentage changes are available. The current value of the portfolio is then multiplied by those n percentages to estimate the distribution of future portfolio $P\&L$. n different values will characterize this distribution, and VaR is then simply computed as the value that corresponds to the $1-\alpha$ percentile.

An implication of this set up is that it will be necessary to estimate the parameters and correlations of the various distributions since those are already embedded in the repricing of the portfolio. Another necessary consequence is that this method can also be applied to instruments whose pay-off is nonlinear with respect to risk parameters. The following section describes how historical simulation works in practical terms.

4.4.1 Computation

The computation of value at risk according to the historical simulation approach can be divided into some steps. Those steps are always the same, irrespective of the complexity of the portfolio to be considered. Of course, as the complexity of the portfolio increases, the procedure will be more elaborate and time-consuming.

Step 1. The first step is, again, risk mapping. The portfolio should be deeply analyzed so to understand which market factors drive its value and which standardized positions can fully describe its behavior. Then, the estimation window should be decided. If the aim is to compute VaR over a one-day horizon, it will be necessary to use daily data. Conversely, if VaR is computed over a one-month holding period, data must be monthly and so on. Once the holding period has been set, then it is necessary to choose the length of the time series to use in the estimation. This choice is especially relevant. The further you go in the past, in fact, the more observations you gather. However, market conditions may well be changed. Consequently, a right balance between the number of data points and the representativeness of the sample should be found.

Step 2. Once this choice has been made, historical time series for the market factors should be obtained. Assume that, in Step 1, it was decided to collect data points for $n+1$ periods. If those data are available, then the following step is to compute changes in the market factor from one observation to the other. Depending on the chosen holding period, those changes may be daily, monthly and so on. If, for the market factor i, the available historical time series is defined as:

$$M_i = \{m_{i,1}, m_{i,2}, ..., m_{i,n+1}\}. \tag{4.3}$$

Then percentage changes, $r_{i,j}$ should be computed as $r_{i,j} = \frac{m_{i,j+1}}{m_{i,j}} - 1$ with $j = 1, 2, ..., n$. This procedure gives, as a result, the percentage changes of each market

factor over the estimation period. As the holding period increases, we may expect those percentages to increase.

Step 3. This step is the heart of the historical simulation approach. The goal of this step is the production of the future distribution of portfolio $P\&L$. In order to achieve this goal, it is necessary to use the output of the previous step. Given the n percentage changes of the k market factors, $r_{i,j}$, those will be combined with the actual (date 0) observable values. This means that the value of the market factors will be a result of their historical evolution but not equal to them. In practical terms, for each market factor i, the first percentage change, $r_{i,1}$, is multiplied by the current value of the factor, $m_{i,0}$, to obtain the first data point, $m_{i,1}$:

$$m_{i,1} = m_{i,0}(1 + r_{i,1}) \tag{4.4}$$

Once this number is available, following values are simply computed by multiplying the factor with the historical percentage change, i.e.:

$$m_{i,j} = m_{i,j-1}(1 + r_{i,j}) \tag{4.5}$$

Once the historical based evolution of the market factors has been computed for the n data points, it is only necessary to combine them to discover the distribution of portfolio value. This is based on the relationship that has been identified in the risk mapping phase. The result of this procedure is that portfolio distribution has not actual historical values, but a re-evaluation of them based on the current status. Finally, if $P = \{p_1, p_2, ..., p_n\}$ is the distribution of portfolio value, it is necessary to compute the $P\&L$ distribution. It is simply done by subtracting to the i-th value of the distribution today's actual value:

$$P\&L_i = p_i - p_0.$$

Step 4. The most important outcome of Step 3 is the series of portfolio profits and losses for the n observation periods. In Step 4 it is only necessary to order them in ascending order.

Step 5. Finally, as the ordered $P\&L$ distribution is available, the value that corresponds to the $1 - \alpha$ percentile must be selected. This number is, in fact, the estimated value at risk[7].

4.4.2 Example

The computation of VaR trough the historical simulation approach will be made clear via the same example used for the delta normal approach.

[7]Note that interpolation may be necessary

Consider the forward contract defined in 4.2.2. Recall that on the delivery date, the American bank will deliver 6 mil. USD and will receive 10 mil. GBP. The maturity of this contract is $T = 90$ days. It has been shown that the value of this instrument is influenced by three figures: the current spot exchange rate USD vs. GBP, S, and the two three-months interest rates (one per currency), i_{USA} and i_{UK}. This contract can thus be seen as the sum of a long position in a three-months GBP denominated zero-coupon bond with face value 10 mil. GBP and a short position in a three-months USD denominated zero-coupon bond with face value 6 mil. USD. The value at time 0 (today) of the contract is then:

$$S\frac{10GBP}{(1+i_{UK})^{0.25}} - \frac{6USD}{(1+i_{USA})^{0.25}}.$$

A good starting point is the computation of the current value of the portfolio. In order to calculate it, we need to observe, on the markets, the values for S, i_{USA} and i_{UK}. S is the exchange rate denominated EUR vs GBP, assume it is equal to $S_0 = 1.29$. i_{USA} is the rate on US three-month T-Bill, assume $i_{USA} = 2.3\%$. Finally, i_{UK} is the rate on the three-month English government bonds, assume $i_{UK} = 0.9\%$[8]. Now we have all the necessary inputs to compute the current value of the portfolio according to the formula

$$PV(P) = S_0\frac{10GBP}{(1+i_{UK})^{0.25}} - \frac{6USD}{(1+i_{USA})^{0.25}} = 6.91mil.USD$$

Set $\alpha = 0.05$, the confidence level. Now it is necessary to decide the estimation period, i.e., the number of observations we need in order to estimate future portfolio distributions. First, VaR will be computed with a one-month horizon. Consequently, historical observations need to be monthly. Assume that we need $n = 100$ observations in order to be confident about the future behavior of the portfolio. Since observations are collected on a monthly basis, the dataset will run from October 2010 to February 2019. Now we can turn to the analysis of the three market factors.

Data for the exchange rate USD vs GBP are monthly, from October 2010 to February 2019. Source is Banca d'Italia database. Once the dataset is available, $S = \{s_1, s_2, ..., s_{101}\}$, we need to compute percentage daily changes of the exchange rate: $r_{S,i} = \frac{s_{i+1}}{s_i} - 1, \forall i$. The result of this computation will be $n = 100$ monthly percentage changes in the exchange rate. Now it is necessary to apply those percentages to the current value of the market factor (and, in loop, to all preceding values) so to obtain the simulated evolution of the exchange rate: $S_{s,i} = S_{s,i-1}(1 + r_{S,i})$. The graph in Figure 4.8 depicts the comparison between historical and simulated exchange rates. As expected, the trend is the same, but actual values differ. This comes from the fact that historical simulation applies historical percentage changes to actual values.

The second risk factor is the yield on 3-month US T-bills. Data for US T-bills are monthly, from October 2010 to February 2019. Source is the Federal Reserve

[8] All interest rates are annualized.

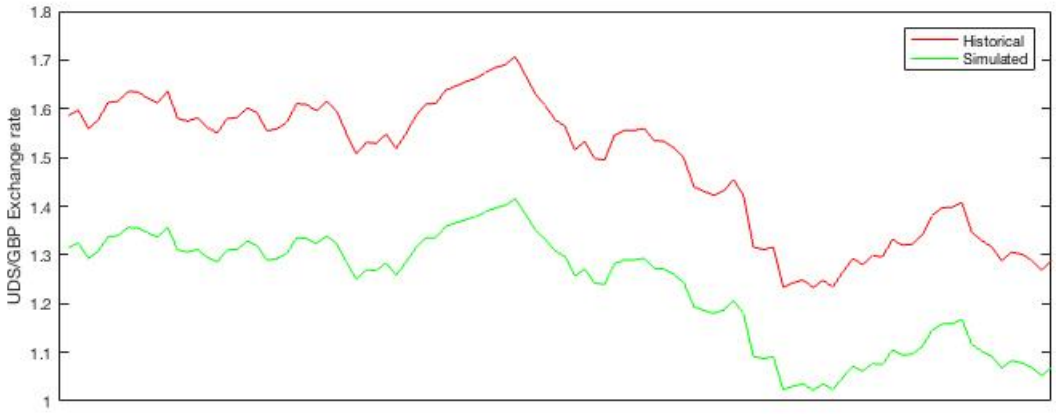

FIGURE 4.8: Historical and simulated USD vs. GBP exchange rate

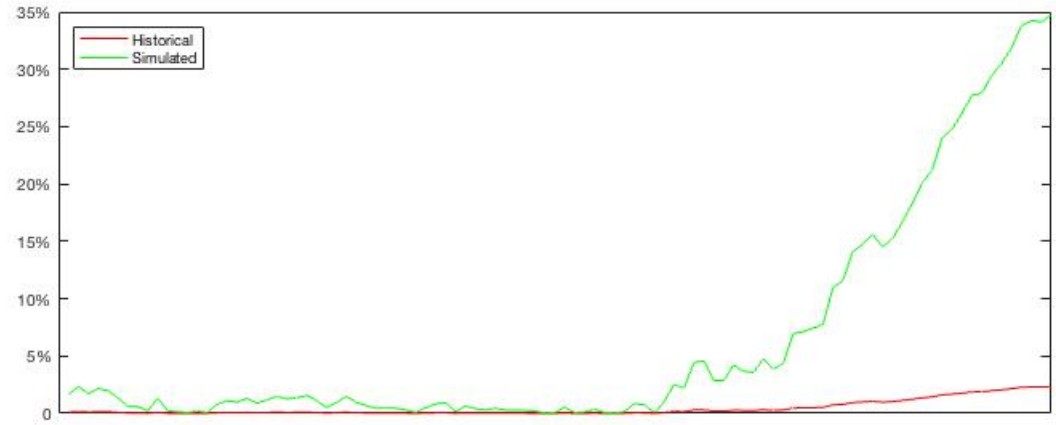

FIGURE 4.9: Historical and simulated USD T-bill rates

database. Once the dataset is available, $i_{USA} = \{i_{USA,1}, i_{USA,2}, ..., i_{USA,101}\}$ we need to compute percentage daily changes of the yield: $r_{USA,i} = \frac{i_{USA,i+1}}{i_{USA,i}} - 1, \forall i$. The result of this computation will be $n = 100$ monthly percentage changes in the US 3-month T-bill rates. Now it is necessary to apply those percentages to the current value of the market factor (and, in loop, to all preceding values) so to obtain the simulated evolution of the rate: $i_{USAs,i} = i_{USAs,i-1}(1 + r_{USA,i})$. Figure 4.9 depicts the comparison between historical and simulated T-bill rates. The trend is still the same, but the simulated path seems to be much more pronounced than the historical one. This comes from the fact that US rates, in the period considered, had experienced significant growth in percentage terms. If the starting value to which those rates are computed is high, then the positive trend is exacerbated.

The final risk factor is the yield on the 3-month UK T-bill. Data for UK T-bills are monthly, from October 2010 to February 2019. Source is Bank of England. Define the

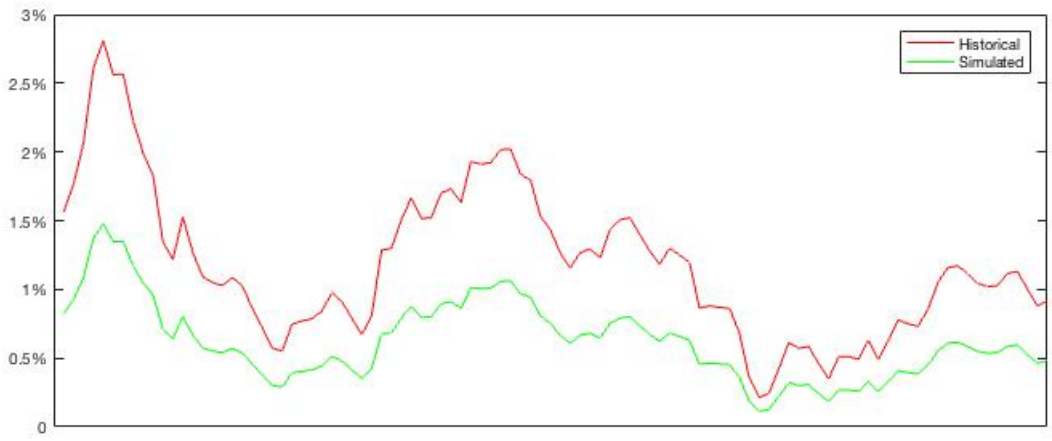

FIGURE 4.10: Historical and simulated UK T-bill rates

historical evolution of yields over the relevant period as

$$i_{UK} = \{i_{UK,1}, i_{UK,2}, ..., i_{UK,101}\}.$$

Next step is to compute percentage daily changes: $r_{UK,i} = \frac{i_{UK,i+1}}{i_{UK,i}} - 1, \forall i$. The result of this computation will be $n = 100$ monthly percentage changes in the UK 3-month T-bill rates. Now, it is necessary to apply those percentages to the current value of the market factor (and, in loop, to all preceding values) so to obtain the simulated evolution: $i_{UKs,i} = i_{UKs,i-1}(1 + r_{UK,i})$.

The comparison between historical and simulated T-bill rates is described in Figure 4.10. As in the exchange rate case, the trend of the two distributions is very close, but with different absolute values (the simulated ones being lower).

Now that the evolution of the simulated market factors has been computed, the last step is to re-evaluate the portfolio, based on those factors. For each scenario i the simulated value of the portfolio will be computed as:

$$P_i = S_{s,i}\frac{10GBP}{(1 + i_{UKs,i})^{0.25}} - \frac{6USD}{(1 + r_{USAs,i})^{0.25}}.$$

Based on this formula, the estimated distribution of the value of the portfolio is described in Figure 4.11.

Finally, it is possible to compute portfolio profits and losses as $P\&L_i = P_i - P_0$. The distribution of $P\&L$ is depicted in the histogram in Figure 4.12. Compared to the Normal, it has fatter left tails and non-zero kurtosis. Moreover, the right tail is longer than in the Normal case.

When this distribution is ordered from the highest gain to the greatest loss, the VaR is simply computed as the value correspondent to the 95-th observation. The estimation is then $VaR = 0.61$ mil. USD. This figure is much higher than predicted by the variance-covariance method. This not a surprise. If Normality is assumed in

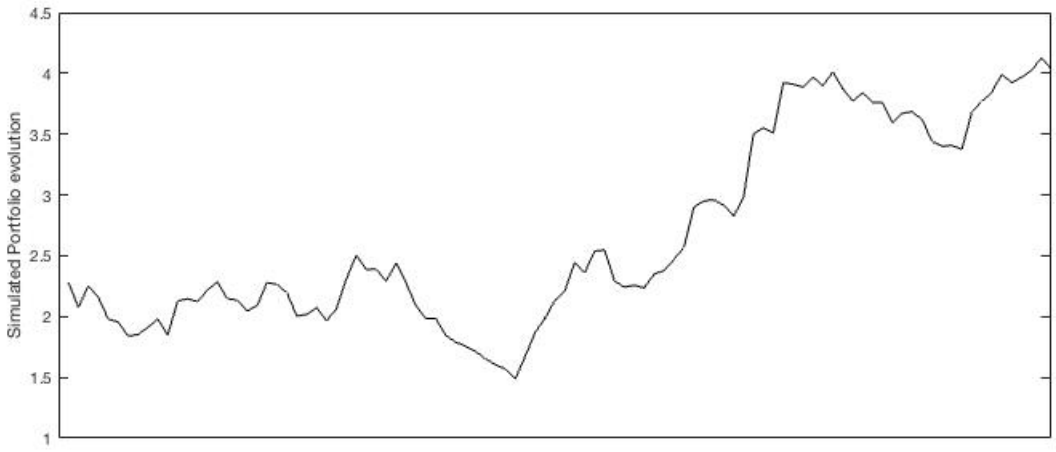

FIGURE 4.11: Simulated evolution of the portfolio

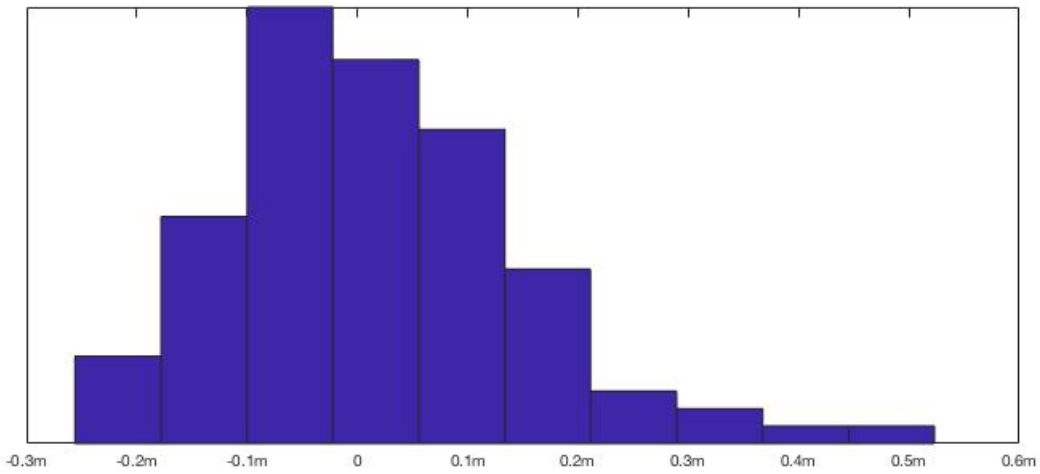

FIGURE 4.12: Simulated distribution of P&L

the presence of fatter tails and outliers, in fact, the result is an underestimation of Value at Risk.

The code below describes all computations that were performed:

LISTING 4.2: Historical simulation VaR for Portfolio

```
%% VaR Historical simulation

n=100 %target number of data points
a=0.05 %confidence level
i_usa_0=0.023 %initial value
i_uk_0=0.009 %initial value
S_0=1.29 %initial value
Q_us=6
Q_uk=10
T=0.25 %maturity
P_0=S_0*(Q_uk)/(1+i_uk_0)^T-Q_us/(1+i_usa_0)^T %initial value portfolio

t1=datetime(2010,10,15)
t = t1 + calmonths(1:100) %generate dates for plotting purposes

%% Exchange rate USD vs GBP

S=GBPUSDmontly(10:end)' %estimation dataset
r_S=zeros(1,n)' % initialization percentage changes

%generate percentage changes

for i=1:n
r_S(i)=S(i+1)/S(i)-1
end

S_1=S_0*(1+r_S(1)) %first simulated value
S_s=ones(1,n)'*S_1 %preallocate space

%generate simulated values

for i=2:n
S_s(i)= S_s(i-1)*(1+r_S(i))
end

%plot simulated vs historical values

hist(S_s)
figure;
plot( S(2:end), "r");
hold on;
plot(S_s, "g");
hold off;
```

```
%% US t-bills

i_usa=USDBillsmonthly(10:end)*0.01 %estimation dataset

r_usa=zeros(1,n)' % initialization percentage changes

%generate percentage changes

for i=1:n
r_usa(i)=i_usa(i+1)/i_usa(i)-1
end

i_usa_1=i_usa_0*(1+r_usa(1)) %first simulated value
i_usa_s=ones(1,n)'*i_usa_1

%generate simulated values

for i=2:n
i_usa_s(i)= i_usa_s(i-1)*(1+r_usa(i))
end

%plot simulated vs historical values

figure;
plot( i_usa(2:end), "r");
hold on;
plot(i_usa_s, "g");
hold off;

hist(i_usa_s)

%% GBP Bills

i_uk=Ukbillsmonthly(10:end)*0.01 %estimation dataset

r_uk=zeros(1,n)' %initialization percentage changes

%generate percentage changes

for i=1:n
r_uk(i)=i_uk(i+1)/i_uk(i)-1
end

i_uk_1=i_uk_0*(1+r_uk(1)) %first simulated value
i_uk_s=ones(1,n)'*i_uk_1

%generate simulated values
```

```
for i=2:n
i_uk_s(i)= i_uk_s(i-1)*(1+r_uk(i))
end

%plot simulated vs historical values

figure;
plot( i_uk(2:end), "r");
hold on;
plot(i_uk_s, "g");
hold off;
hist(i_uk_s)

%% Porfolio

P_0=(Q_uk)/(1+i_uk_0)^T-Q_us/(1+i_usa_0)^T*S_0

P=ones(1,n)'*P_0

%compute portfolio simulated values

for i=2:n
P(i)=(Q_uk)/(1+i_uk_s(i))^T-Q_us/(1+i_usa_s(i))^T*S_s(i)
end

plot(P)
Pl=zeros(1,n)

$compute profits and losses

for i=2:n
Pl(i)=P(i)-P(i-1)
end
Pl_ord=sort(Pl)%sort observations
hist(Pl_ord)

%VaR computation
VaR= Pl_ord(5)
```

4.4.3 Advantages and disadvantages

Historical simulation is by far the most straightforward approach in terms of understanding. In fact, its computation requires the distribution of *P&L*. A visual representation of this distribution makes it extremely easy to explain and interpret the concept of VaR. This is the main reason why banks (and especially small banks) tend

to prefer this approach. However, historical simulation comes with both advantages and disadvantages.

One evident pro is that historical simulations do not require any assumption regarding the distribution and parameters of market factors. This tends to have a positive impact on VaR estimates since, in this way, it is possible to account for non-normality (fat tails, kurtosis). This is unfortunately not possible in the parametric approach, and thus, especially for some asset classes, this method should be preferred.

A connected drawback is that the distribution of market factors depends solely on historical patterns. This implies the assumption that history is a reliable source of information for the future. If future market factors deviate from their history, the historical VaR gives unreliable estimates which may be misleading. In order to avoid this critical occurrence, it is necessary to look at volatilities. When the historical period is chosen, "calm" scenarios with exceptionally low volatility (in relative terms) should be avoided, in order to account for eventual market crashes. A connected issue is that, when the historical time series for the market factor is selected, each data point in that sequence has the same weight in the determination of the future distribution. If the analyst has some views about the future and if there are reasons to assume that the recent past would play a prominent role in the determination of future prices, then this should be taken into account in the calibration of the model.

Another limitation comes from data availability. Historical simulation can be implemented only if a long history of time series for market factors is available. For assets that are not frequently traded (or newly introduced assets), it is not possible to apply this model and thus the choice is between parametric and Montecarlo methods.

Finally, historical simulation is usually thought to be computationally efficient, but as the number of risk factors increases, it becomes extremely costly to re-evaluate the portfolio at each scenario. In those cases, a delta normal approach may be well fitted.

4.5 Montecarlo simulation approach

Montecarlo simulation is by far the most sophisticated approach to the computation of Value at Risk. When applying a Montecarlo technique, a numerical approach is implemented in order to produce future random paths for the market factors. This method is non-parametric because it is, in principle, history-free. Future evolutions of the market factors are generated via a pseudo-random number generator. Once the random paths have been generated, the portfolio is re-evaluated accordingly and the VaR is simply computed based on the simulated distribution of $P\&L$. In this respect, the Montecarlo approach is very close to the historical simulation. It is then critical, in implementing this technique, to choose the correct probability distribution that best describes the potential future evolution of market factors. The most

important consequence is that, since this distribution needs not to be Normal, it is possible to account for fat tails and kurtosis. However, in practical terms, history still plays a role. Practitioners, in fact, usually look at the distributions of past returns in order to decide which pseudo-random generator to use. This means that they try to match the simulated path with the historical one. Nevertheless, it is possible to incorporate views about the future evolution of market factors. This is the main advantage of the Montecarlo technique with respect to both the delta normal approach and the historical simulation. Historical and Montecarlo simulations are indeed very similar in terms of steps to be performed for their implementation. The main critical difference stems in the generation of the future distribution of market factors. In the historical simulation approach, the n possible future scenarios for the market factors are generated based on the percentage changes of the market factors in a given historical period. This implies that the distribution of the market factors is not the historical one, but it is determined by past evolution. In the Montecarlo method, instead, paths are generated randomly and set to obey to a specific distribution. This distribution may well be totally different from the historical one. Once the path of market factors has been determined, then historical and Montecarlo simulations have no differences in terms of implementation. Even if the Montecarlo technique tends to be superior in terms of performance compared to the other two, it is also computationally expensive. In order to get reliable estimations, in fact, the number of estimations performed should be quite high (order of thousands or hundreds of thousands). However, while the reliability of the estimation is increased, computation time and costs increase too.

4.5.1 Computation

As mentioned, Montecarlo approach to Value at Risk is computationally burdensome. The most costly step is the generation of pseudo-random paths for the market factors, but the most critical and conceptually complicated part is the choice of the distribution from which to generate pseudo-random paths for the market factors. If the choice is not able to correctly match the future evolution, then VaR estimates are unreliable.

The procedure for the computation of VaR according to Montecarlo simulations can again be divided into steps. Those steps will resemble the ones just described for the historical simulation approach. As already mentioned, critical differences lie in Step 3, where the generation of the distribution of market factors is performed.

Step 1. As usual, the first step is risk mapping. Given the portfolio for which VaR has to be determined, it should be decomposed in long and short positions on standardized market instruments. Those instruments need to be simple and directly influenced by observable market factors. This step is indeed quite technical. It is in fact not sufficient to determine in abstract terms which market factors influence the relevant portfolio. It is necessary to specify a linear or non-linear equation that maps the portfolio in the relevant standardized positions. This equation will, in

fact, be the starting point for the re-evaluation of the portfolio based on the random paths generated for the market factors in Step 4. Consider, for example, the forward contract described throughout the chapter. This contract is influenced by the US and UK money market and by the exchange rate. However, it is not sufficient to conclude this. It is necessary to specify the equation that directly links the value of the forward contract to the value of US and UK interest rates and the exchange rate USD vs GBP. Only when such an equation has been determined in closed form, the risk mapping exercise can be considered as concluded. An essential aspect of the Montecarlo method, that clearly distinguishes it from the delta normal approach is that it is suitable also for instruments whose pay-off is a non-linear function of the market factors (for example options). If this is the case, of course, computations will become much more complicated.

Step 2. This step is especially critical. Once the market factors have been identified, it is necessary to choose their distribution. This choice is difficult because it involves expert judgment regarding future behavior of markets. Moreover, a flaw in this process has serious consequences, since it invalidates VaR estimates. In practical terms, what analysts usually do is looking at historical distributions of market factors over different horizons. Based on historical behavior but also on their views about future evolutions, the distributions and parameters of market factors are determined. Parameters are usually selected based on historical time series, but they can also be modified reflecting views about the future. The clear advantage of this set-up is that distributions need not to be Normal, nor they must reflect historical paths. Analysts can freely choose the distribution they think is best fitted to the description of the future evolution of market factors. This is an improvement both of the delta normal approach and of the historical simulation. On the one hand, in fact, market factors are not forced to behave as a Normal distribution. It has already been pointed out that distributions of asset returns in real markets are far from Normal. They tend to be characterized by masses on the tails, that imply a higher concentration of extreme values (both positive and negative) compared to what predicted by the Normal, and by higher peaks. Distributions with those patterns are common in asset returns and are called leptokurtotic. For those distributions, Normal approximation is not appropriate and thus we should be able to produce more reliable estimations of VaR if we include this information in the distribution choice. This is not possible in the delta normal approach. On the other hand, there is no guarantee that history will repeat itself. If for example, we choose as estimation window a highly volatile or too "quiet" period, we may seriously invalidate the VaR estimate, by over or underestimation. Historical simulation is totally reliant on market past data and it is not possible to incorporate analysts' views on the simulated distribution of market factors. It is also true that even Montecarlo simulation relies to some extent on historical data. Time series of market factors are in fact used to understand their behavior and eventually estimate the parameters of the distributions. However, it is possible to deviate from historical paths (something that is instead not possible in

the case of the historical simulation) and include judgemental views on the future, both in terms of distributions and of parameters. This is considered one of the most important advantages of the Montecarlo method. In Figure 4.13, for example, the distribution of historical US 3-month T-bill rates is plotted against the Normal pdf. From the graph, it is evident that the Normal is not the best choice.

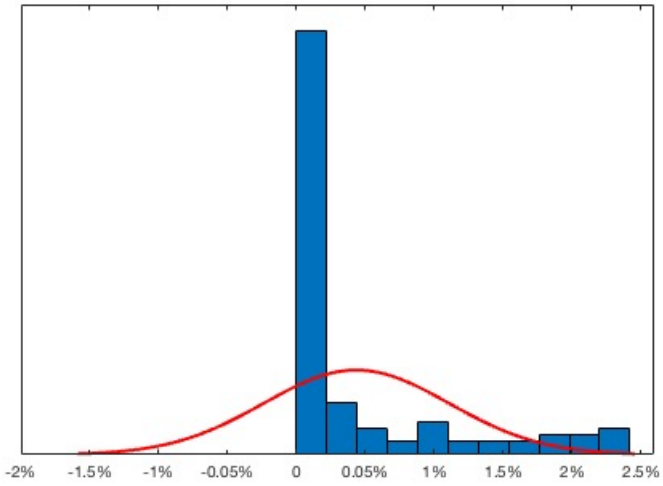

FIGURE 4.13: USD 3-Month T-bill rates and Normal distribution

As in the variance-covariance approach, but differently from the historical simulation, in this step, it is also necessary to choose the degree of correlations between the market factors. In the other two methods, correlations are historical. In the delta normal approach, in fact, correlations are specified in the computation of portfolio variance and are based purely on historical values. In the historical simulation approach, instead, correlations are not made explicit but are embedded in the generated paths for market factors and reflect past time series. In the Montecarlo approach, it is, of course, useful to look at historical correlations, but after that, correlations can be set as desired. This is another crucial advantage if we consider the fact that correlations tend to be time-varying, especially in periods of market turmoil. This implies that historical figures may not be a good estimation for future values. The Montecarlo set-up overrides this problem.

The outcome of this step will thus be the definition of the distribution of the various market factors $M_1, M_2, ..., M_k$ and their respective parameters: $M_i \sim X_i(\cdot)$, $\forall i$.

Step 3. This step lies out the differences between the Montecarlo and Historical simulation approaches. In this phase, in fact, pseudo-random paths for the market factors are generated, based on the distribution that was chosen in the previous step. Then, for each market factor, M_i, n hypothetical values are generated. The distribution of these values (including the parameters) obeys to the one that has been chosen. In general, the number of runs of the simulation, n, should be large enough, say $n = 10000$ or $n = 100000$. However, as n increases, computational costs increase dramatically too. The outcome is then a pseudo-random path in the form:

$M_{i,s} = \{m_{i,s,1}, m_{i,s,2}, ..., m_{i,s,n}\}, \forall i$. The main difference with respect to historical simulation is that values of the market factors are here determined directly. This means that, as long as the target distribution has been chosen, values for the market factors are immediately generated. These values represent possible future evolutions of the market factors. In the historical simulation, instead, the generation is indirect. Historical values of market factors are in fact used to compute historical percentage changes, and then those changes are applied to the current status to come up with possible future evolutions of the market factors.

From the point in which the path of market factors has been generated, there is no difference anymore between Historical and Montecarlo simulation.

After having generated the pseudo-random values for market factors, those have to be combined according to the equation specified in Step 1, so to be able to compute simulated portfolio values in the n scenarios. The outcome is then the same of historical simulation, but of course, the two distributions will be different, being the underlying assumptions different.

Recall that VaR is computed starting from the $P\&L$ distribution, not from portfolio values. Given the distribution of possible evolutions of portfolio values, $P = \{p_1, p_2, ..., p_n\}$, computed based on the evolution of market factors, the $P\&L$ distribution is simply given as the difference of these values from today's actual value of the portfolio: $P\&L_i = p_i - p_0, \forall i$.

Step 4. The most important outcome of Step 3 is the series of portfolio profits and losses for the n observation periods. In Step 4 it is only necessary to order them in ascending order.

Step 5. Finally, as the ordered $P\&L$ distribution is available, it is only necessary to select the value that corresponds to the $1 - \alpha$ percentile. This number is, in fact, the estimated value at risk[9].

A distinctive feature Montecarlo methods for VaR are based on the generation of random paths for market factors. This implies that every time the simulation is run, the estimate of VaR will be different because the pseudo-random values of the market factors will necessarily be different.

This does not happen in the variance-covariance approach nor in the historical simulation because in those models, here is no source of randomness.

Following this distinctive feature of Montecarlo techniques, it is possible to run the simulation several times, say k times. The outcome will be k different estimates of VaR. In order to improve VaR estimation, it is advisable to run the simulation several times and then take their average value. This value is, in general, more reliable than the one obtained by running only once the simulation.

In addition, since we have obtained k estimations of VaR, those can be considered as forming a vector. This vector has its variance and the standard error of the estimation can significantly be reduced by increasing the number of runs k.

[9]Note that interpolation may be necessary

4.5.2 Example

We focus again on the forward example made throughout the chapter. Now some steps will be followed so to explain the implementation of Montecarlo VaR.

Consider the forward contract defined in 4.2.2. As said, this contract involves the delivery by the American bank of 6 mil. USD and an incoming cash-flow 10 mil. GBP. Both cash-flows will occur in $T = 90$ days, the maturity of the contract. It has been shown that three figures influence the value of this instrument: the current spot exchange rate USD vs. GBP, S, and the two three-months interest rates (one per currency), i_{USA} and i_{UK}. This contract can thus be seen as the sum of a long position in a three-months GBP denominated zero-coupon bond with face value 10 mil. GBP and a short position in a three-months USD denominated zero-coupon bond with face value 6 mil. USD. The value at time 0 (today) of the contract is then:

$$S\frac{10GBP}{(1+i_{UK})^{0.25}} - \frac{6USD}{(1+i_{USA})^{0.25}}.$$

Again, we will start by computing the current value of the portfolio. In order to calculate it, we need to observe, on the markets, the values for S, i_{USA} and i_{UK}. S is the exchange rate denominated EUR vs GB,P assume it is equal to $S_0 = 1.29$. r_{USA} is the rate on US three-month T-Bill, assume $r_{USA} = 2.3\%$. Finally, r_{UK} is the rate on the three-month English government bonds, assume $r_{UK} = 0.9\%$[10].

Now we have all the necessary inputs to compute the current value of the portfolio according to the formula

$$PV(P) = S_0\frac{10GBP}{(1+i_{UK})^{0.25}} - \frac{6USD}{(1+i_{USA})^{0.25}} = 6.91mil.USD$$

Set $\alpha = 0.05$, the confidence level, while the holding period assumption for VaR computation will be 1-month. For what concerns simulation runs, we set $n = 1000$, meaning that for each market factor $n = 1000$ different evolution scenarios will be generated. Now that assumptions have been made, we can start the Montecarlo procedure.

We will start from the most complex part: the choice of the probability distribution for the risk parameters. Recall that in our case, the market factors are only three, but in general, for complex portfolios, this exercise should be replicated for every parameter able to influence the portfolio.

The first risk factor is the exchange rate USD vs. GBP. First, a good idea is to look at the historical evolution of this factor. A graphical depiction of this distribution is provided in Figure 4.14. Assume that, based on expert judgment, the distribution of this market factor will be close to the historical one, but with reduced mass on extreme high values, say from 1.38 on. Based on this view, we need to find the best distribution which can describe this random variable. When making this choice, it is

[10] All interest rates are annualized.

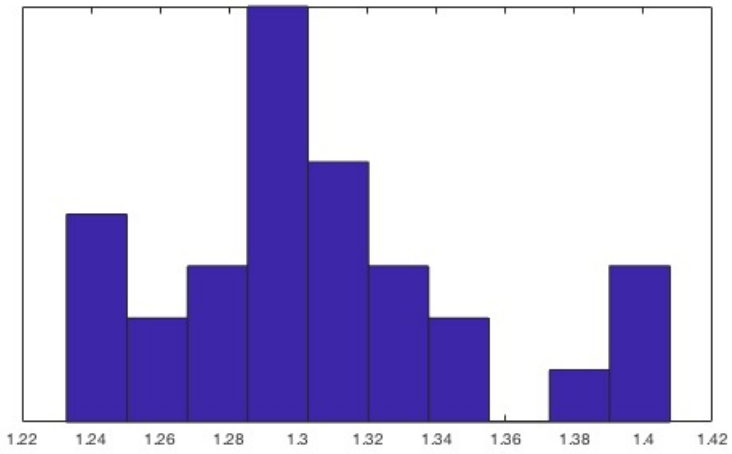

FIGURE 4.14: USD-GBP histogram

important to keep in mind that it will not be possible to find a statistical distribution that perfectly mimics the behavior of this factor. What is needed is the best possible approximation. In order to make this choice, the following statistical distributions were analyzed:

- Beta;
- Birnbaum-Saunders;
- Burr Type II;
- Exponential;
- Extreme value;
- Gamma;
- Generalized extreme value;
- Generalized Pareto;
- Inverse Gaussian;
- Logistic;
- Log-logistic;
- Log-normal;
- Nakagami;
- Negative Binomial;
- Normal;
- Poisson;

- Rayleigh;
- Rician;
- t location-scale;
- Weibull.

Keeping in mind the view that has been formulated about the future (i.e., less mass on extreme high values) the easiest and fastest way to understand which distribution better approximates the historical series of exchange rates is to compare the histogram of this time series with the pdf of all those distributions. Data for the USD/GBP exchange rate are from Banca d'Italia data stream. They are monthly and run from January 2003 to June 2019. Some distributions were immediately excluded due to total lack of fit with the historical distribution or because they require some assumptions on the data that were not respected (Beta, Burr Type II, Exponential, Generalized Pareto, Inverse Gaussian, Loglogistic, Lognormal, Nakagami, Negative binomial, Poisson, Rayleigh, Rician and t-location scale).

First, we will consider the fit of the Normal. Figure 4.15 plots the histogram of the historical distribution of this exchange rate and the pdf of a Normal. As can be seen, the Normal does not appear to be the best choice for at least two reasons. First, it fails to account for the high peak concentrated around the value of 1.3. Second, in the period considered low values of the rate (1.15-1.22) were not observed, while this distribution has significant mass allocated to these values. The conclusion is that we should continue looking for a distribution with a better fit.

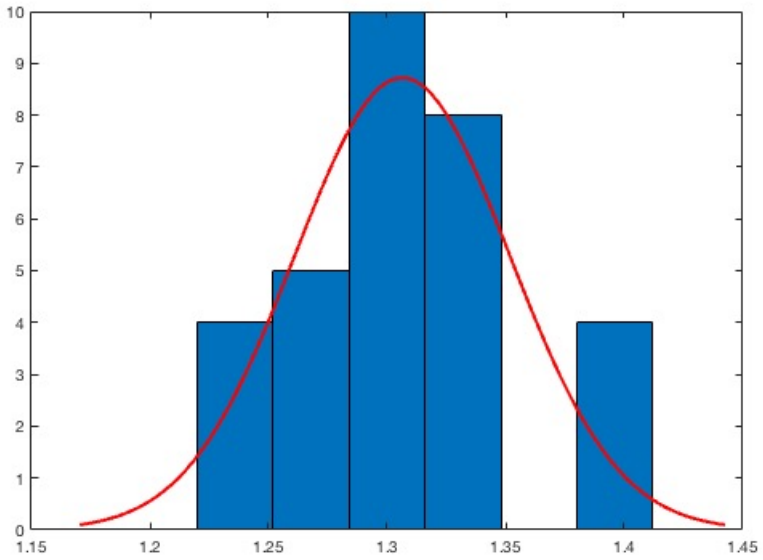

FIGURE 4.15: USD vs. GBP exchange rate and Normal distribution

Another option is the Birnbaum-Saunders distribution. Again, as shown in figure 4.16, this distribution fails to account for the high peak concentrated around the value of 1.3. This distribution is thus not well suited to the data and views. The fit

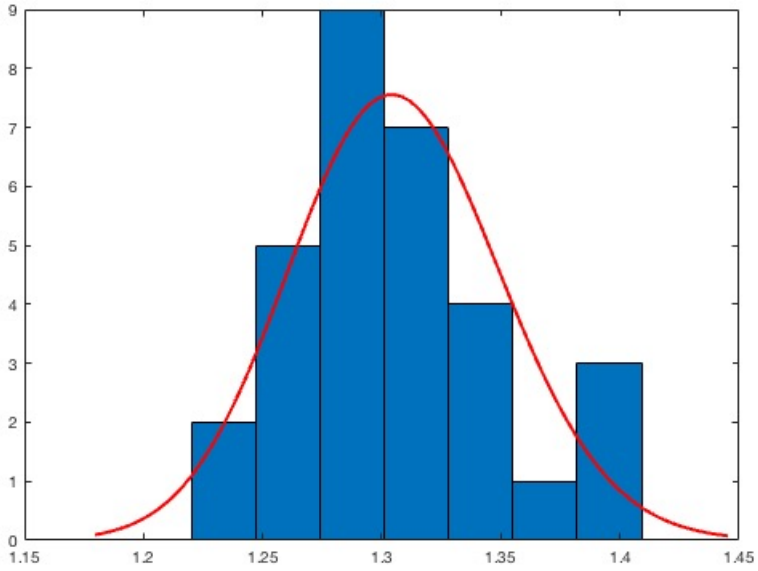

FIGURE 4.16: USD vs. GBP exchange rate and Birnbaum-Saunders distribution

of USD-GBP exchange rate with the Extreme value distribution is depicted in Figure 4.17. Clearly this distribution is not a good choice. First, in aggregate terms, values of the exchange rate in the range $[1, 1.2]$ receive significant mass. Second, even this distribution fails to capture the high peak around the value 1.3.

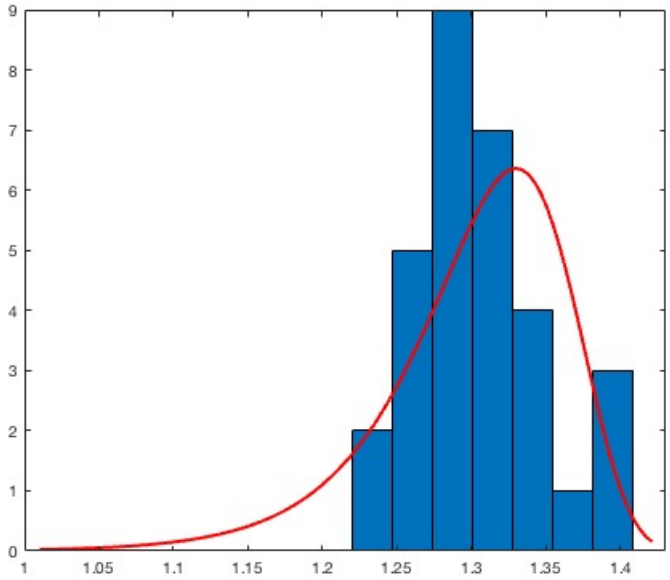

FIGURE 4.17: USD vs. GBP exchange rate and Extreme value distribution

The Gamma distribution (Figure 4.18) is another example in which the peak is not captured.

The fit of the empirical distribution with the Weibull is described in Figure 4.19. Even the Weibull fails to capture the high peak and assigns too much mass to low

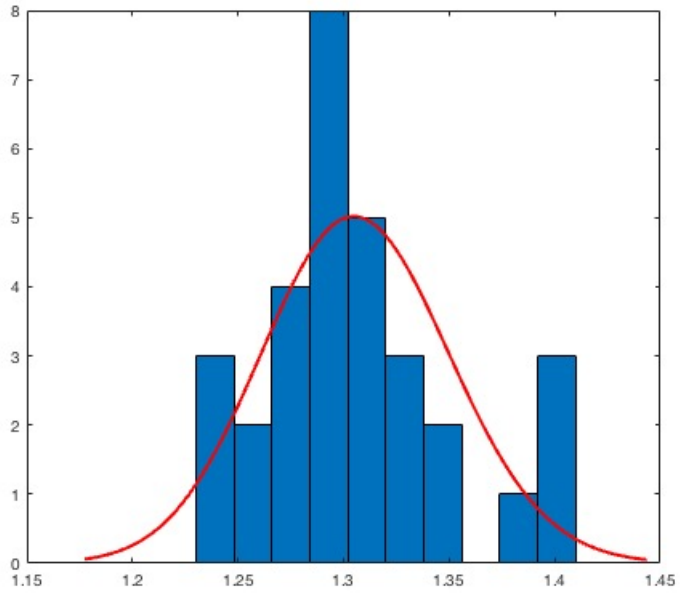

FIGURE 4.18: USD vs. GBP exchange rate and Gamma distribution

values. Finally, in Figures 4.20 and 4.21. The Logistic and the Generalized Extreme

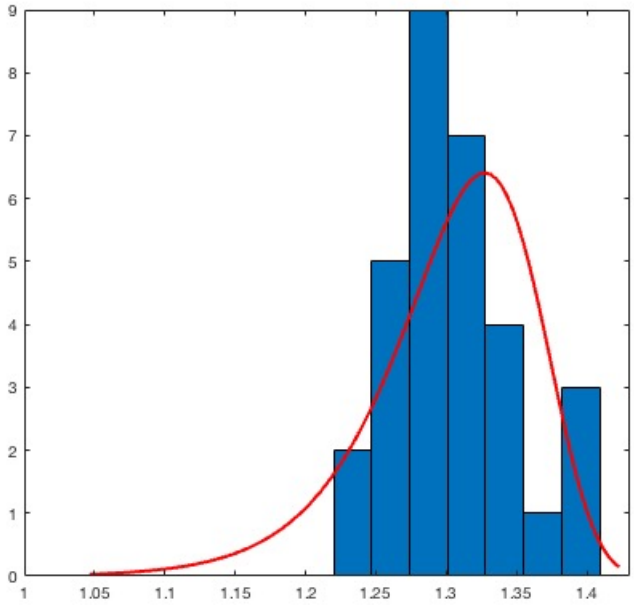

FIGURE 4.19: USD vs. GBP exchange rate and Weibull distribution

Value are described. Both can be considered acceptable because they take into account the high peak and are in line with the analyst's expectation of low mass concentrated around 1.4. We will use the Logistic[11].

The second risk factor is the yield on 3-month US T-bills. Data for US T-bills are monthly, from January 2003 to June 2019. Source is Federal Reserve database.

We will apply the same approach used for the the exchange rate. In order to assess which distribution better approximates the historical behavior of the T-bills, we

[11]Note that the use of the Generalized Extreme Value produces comparable results.

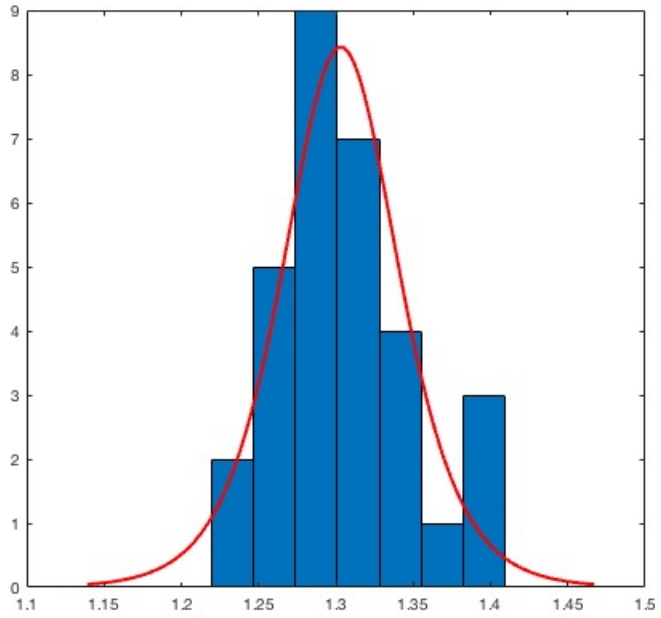

FIGURE 4.20: USD vs. GBP exchange rate and Logistic distribution

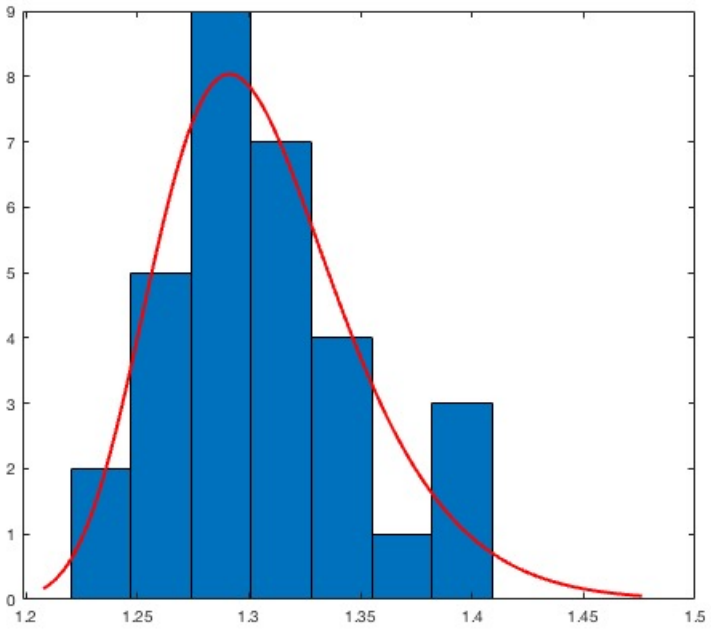

FIGURE 4.21: USD vs. GBP exchange rate and Generalized Extreme Value distribution

will compare the histogram with the pdf. Some distributions were immediately excluded due to total lack of fit with the historical distribution or because they require some assumptions on the data that were not respected (Beta, Burr Type II, Extreme value, Gamma, Generalized extreme value, Generalized Pareto, Inverse Gaussian, Loglogistic, Lognormal, Nakagami, Negative binomial, Poisson, Rician, t-location scale and Weibull).

Start again by considering the fit of the Normal. Figure 4.22 plots the histogram of the historical distribution of the T-bill rate and the pdf of a Normal. The lack of fit is evident. The major problem is that the Normal does not capture the peak concentrated around $[0 - 1\%]$. Another, still smaller, peak that is not taken into account is the one around 2%, which represents the most recent values. Finally the Normal assumes also negative values up to 2%, which are not very reasonable. We can then again conclude that the Normal pdf does not describe well the distribution of T-bills.

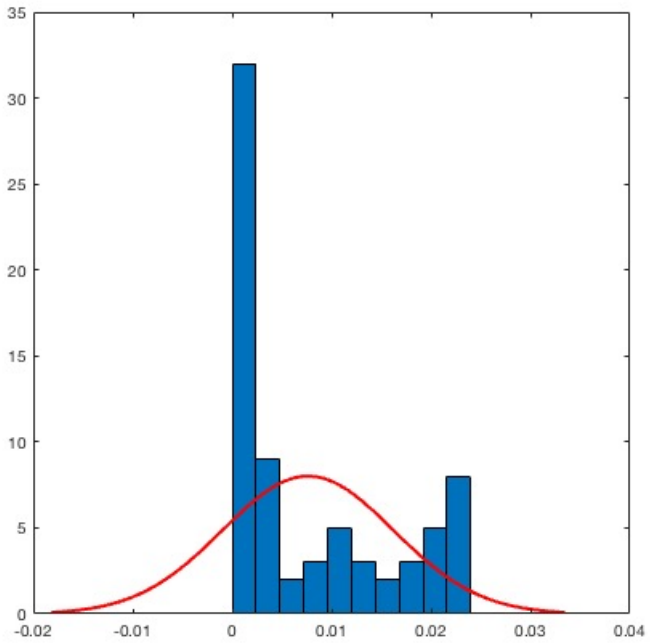

FIGURE 4.22: USD T-bill rates and Normal distribution

Another option is the Birnbaum-Saunders distribution (figure 4.23). This distribution has a shape which fits sufficiently well the empirical distribution. What is not satisfying is the scale. The Birnbaum-Saunders, in fact, reaches values of 10%, which are unrealistically high for a T-bill rate. Such a distributional assumption would probably lead to a misspecification of the VaR measure. It should thus be excluded.

The fit of the T-bill rate with the Rayleigh distribution is depicted in figure 4.24. This is evidently not a right choice. For the inner part of the distribution, in fact, it assumes larger masses than the empirics. Moreover, as the Normal, it fails to capture the heavy peak concentrated at the right tail of the empirical distribution.

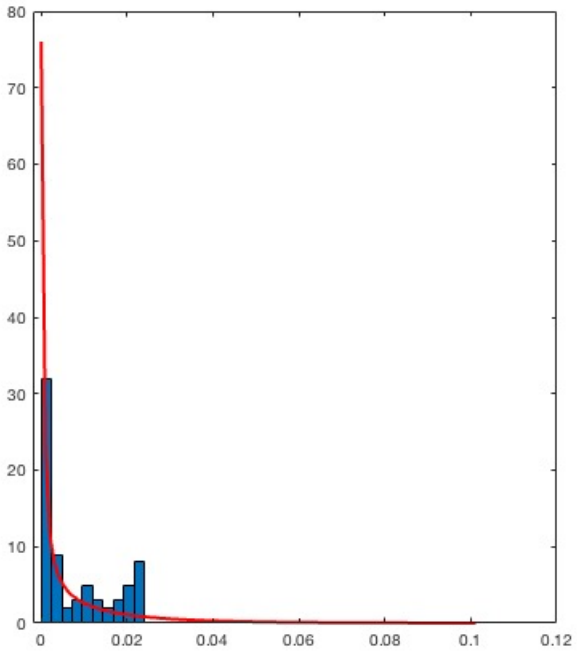

FIGURE 4.23: USD T-bill rates and Birnbaum-Saunders distribution

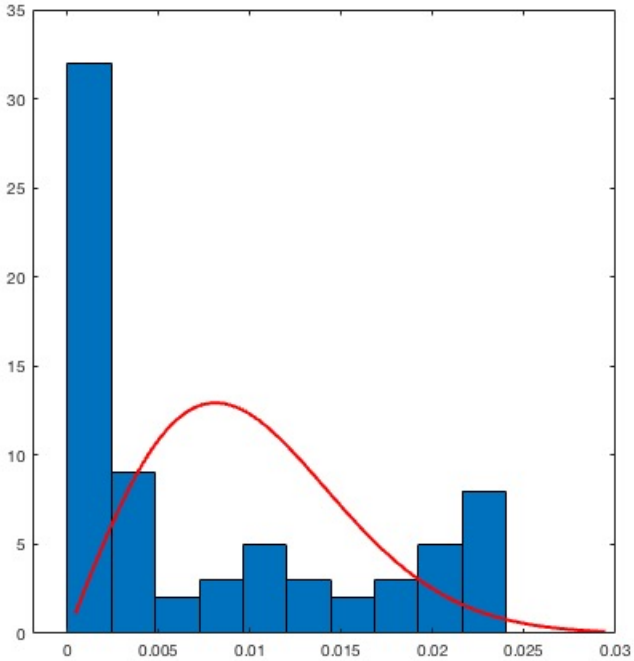

FIGURE 4.24: USD T-bill rates and Rayleigh distribution

Finally, in Figure 4.25, the fit of the historical data with the Exponential is described. This appears to be the most reasonable choice since it is the only distribution which is able to capture the heavy tail concentrated in low values. This is the best choice we can make given this set-up. T-bill rates will then be generated according to the Exponential distribution.

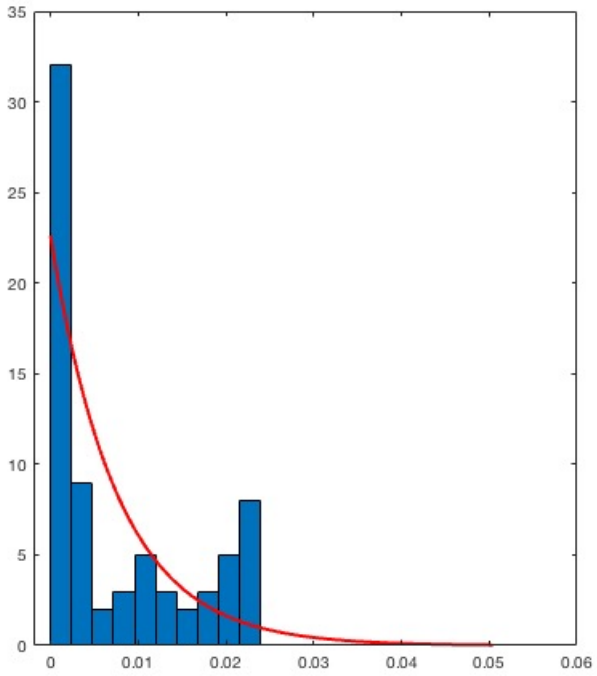

FIGURE 4.25: USD T-bill rates and Exponential distribution

The final risk factor is the yield on the 3-month UK T-bill. Data for UK T-bills are monthly, from October 2010 to February 2019.

Again, some distributions were immediately excluded due to total lack of fit with the historical distribution or because they require some assumptions on the data that were not respected (Beta, Burr Type II, Birnbaum-Saunders, Exponential, Extreme value, Generalized extreme value, Generalized Pareto, Inverse Gaussian, Loglogistic, Lognormal, Nakagami, Negative binomial, Poisson, t-location scale and Weibull).

We will first compare the histogram of this market factor with the Normal (figure 4.26). The fit is not so bad. Among the three market factors, the UK T-Bill rate is by far the closest to the Normal distribution. There still are some peaks that are not captured, but the lack of fit is not so severe. Before taking the decision, we will consider the other remaining cases. Other two options are the Gamma and the Rayleigh distributions depicted respectively in figure 4.27 and 4.28. Even in these cases, the fit is not bad. The only major problem is that the fitted Gamma and Rayleigh reach values of about 3% that are quite high for the UK T-bill and have not been reached in the dataset. Finally, in Figure 4.29, the fit of the historical data with the Rician is described. This is the best choice because it roughly captures all the

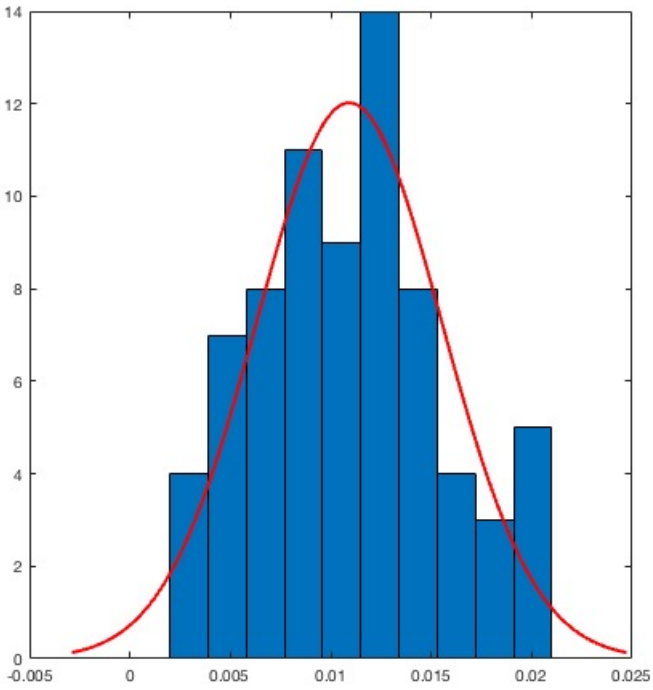

FIGURE 4.26: UK T-bill rates and Normal distribution

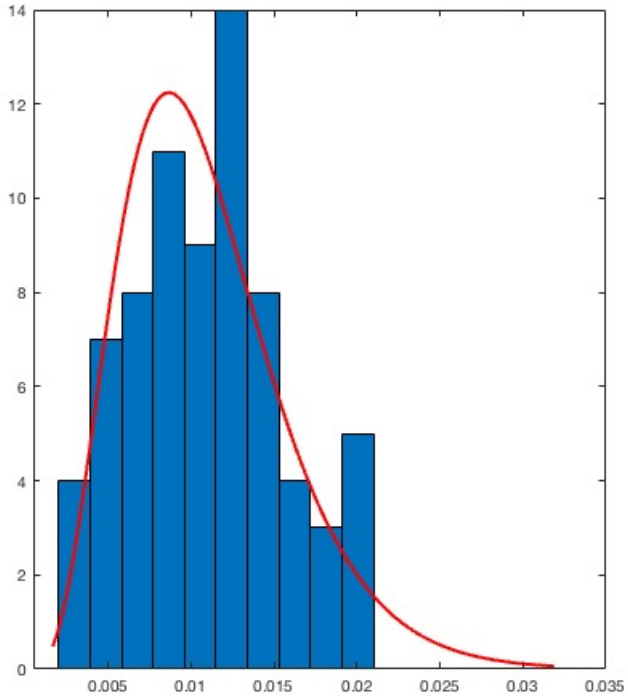

FIGURE 4.27: UK T-bill rates and Gamma distribution

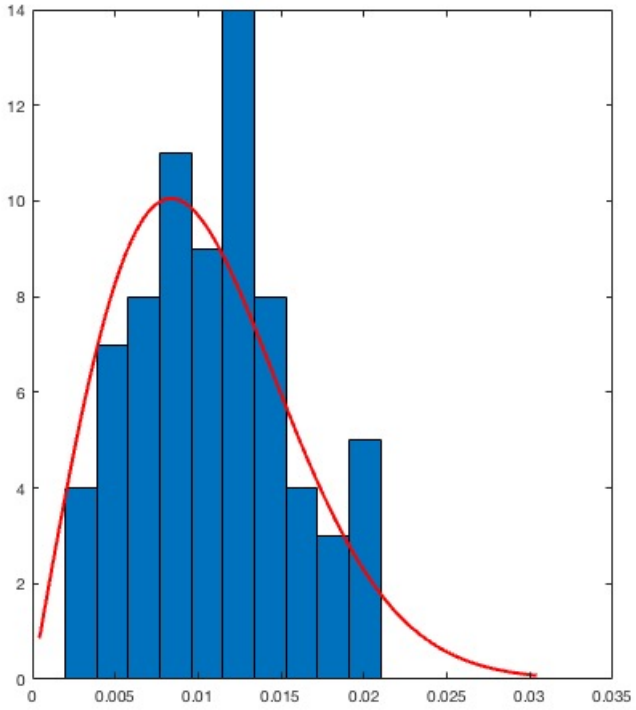

FIGURE 4.28: UK T-bill rates and Rayleigh distribution

peaks, but is also reaches a maximum value of 2.5%, which is more reasonable than 3%. Once the distributions for all market factors have been selected, the evolution of the same must be simulated. In order to generate random numbers drawn from a specific statistic distribution, the relevant parameters need to be specified. All those parameters are estimated based on the available historical time series. For the USD GBP exchange rate, the selected distribution is the Logistic. The two parameters needed are then the mean and standard deviation, which respectively are $\mu_E = 1.30$ and $\sigma_E = 0.24$. For the US T-Bills, the selected distribution is the Exponential, whose only parameter is the mean $\mu_{USA} = 0.76\%$. Finally, the selected distribution for the UK T-Bill is the Rician, whose parameters are $s = 0.94\%$ and $\sigma_{UK} = 0.51\%$.

Once the parameters have been estimated, it is only necessary to generate n random numbers according to the specified distributions. Random number generation has extensively been described in 2 and 3.

As already familiar, now that the evolution of the simulated market factors has been computed, the last step is to re-evaluate the portfolio, based on those factors. For each simulated scenario i the simulated value of the portfolio will be computed as:

$$P_i = S_{s,i}\frac{10GBP}{(1+i_{UKs,i})^{0.25}} - \frac{6USD}{(1+r_{USAs,i})^{0.25}}.$$

Based on this formula, the estimated distribution of the value of the portfolio is described in Figure 4.30.

Finally, it is possible to compute portfolio profits and losses as $P\&L_i = P_i - P_0$. When this distribution is ordered from the highest gain to the greater loss, the

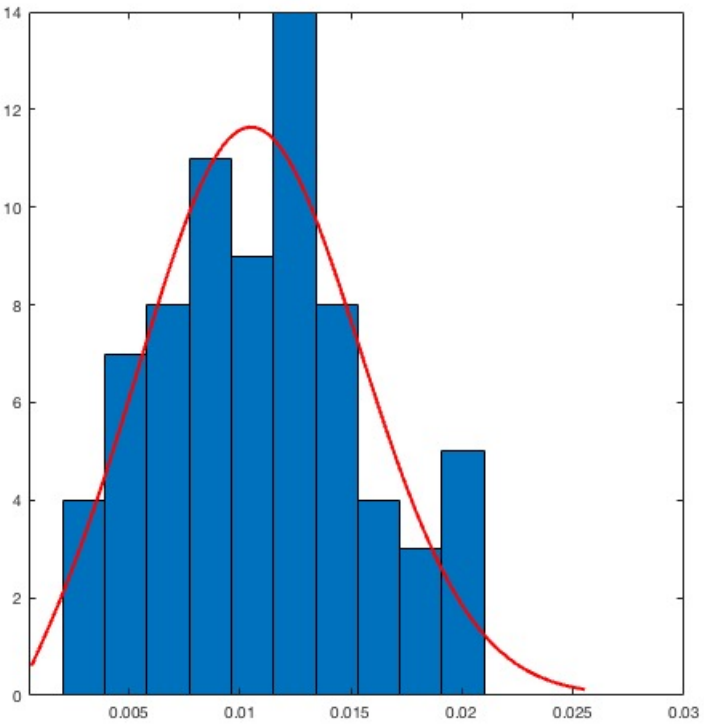

FIGURE 4.29: UK T-bill rates and Rician distribution

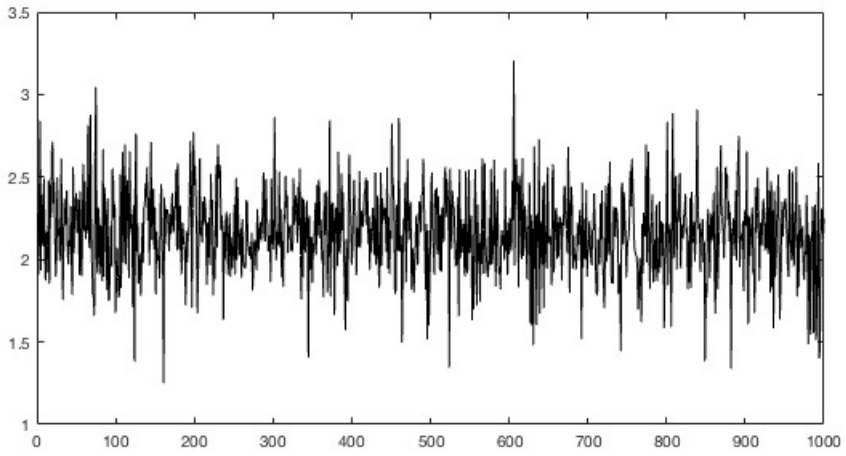

FIGURE 4.30: Simulated evolution of the portfolio

VaR is simply computed as the value correspondent to the 95-th observation. The estimation is then $VaR = 0.53$ mil. USD. This figure is much higher than what predicted by the variance-covariance method but lower than what predicted by the Historical simulation approach.

The code below describes all computations that were performed:

LISTING 4.3: Montecarlo simulation VaR for Portfolio

```
%% VaR Montecarlo simulation

n=1000 %target number of data points
a=0.05 %confidence level
USA_0=0.023 %initial value
UK_0=0.009 %initial value
S_0=1.29 %initial value
Q_us=6
Q_uk=10
T=0.25
%initial portfolio value
P_0=S_0*(Q_uk)/(1+UK_0)^T-Q_us/(1+USA_0)^T

%generate dates for plot
t1=datetime(2003,1,15)
t = t1 + calmonths(1:200)

%% Exchange rate USD vs GBP

S=USDGBP(170:end)
%S_daily=DailyUSDGBP(2:end)

%first look at the distribution

hist(S,5)


%% choose correct distributional assumption

histfit(S) %look at the distribution, normal does not fit well
histfit(S,7,'beta')%no, data is in 0,1
histfit(S,7,'burr')%no
histfit(S,7, 'birnbaumsaunders')%yes
histfit(S,7,'exponential')%no
histfit(S,7,'ev')%not bad
histfit(S,10,'gamma')%not really
histfit(S,7,'gev')%not really
histfit(S,10,'gp')%no
histfit(S,7,'inversegaussian')%not bad
histfit(S,7,'logistic')% not so bad
```

```
histfit(S,10,'loglogistic')%no
histfit(S,10,'lognormal')%no
histfit(S,10,'nakagami')%no
histfit(S,10,'nbin')%no, must be integer
histfit(S,10,'normal')
histfit(S,10,'poisson')%no
histfit(S,10,'rayleigh')%no
histfit(S,10,'rician')%no
histfit(S,10,'tlocationscale')%no, normal is better
histfit(S,7,'wbl')%not bad

% chosen distribution: logistic
% generation of random paths for the factor
pd_S=fitdist(S,'logistic')
S_s=random(pd_S,n,1)
hist(S_s)

%% US t-bills

US=USD20032019(130:end)*0.01

%choose correct distributional assumption

histfit(US)
histfit(US,10,'beta')%no
histfit(US,10,'burr')%no
histfit(US,10, 'birnbaumsaunders')%show
histfit(US,10,'exponential')%not bad
histfit(US,10,'ev')%no
histfit(US,10,'gamma')%no
histfit(US,10,'gev')%no
histfit(US,10,'gp')%no
histfit(US,10,'inversegaussian')%no
histfit(US,10,'logistic')%no
histfit(US,10,'loglogistic')%no
histfit(US,10,'lognormal')%no
histfit(US,10,'nakagami')%no
histfit(US,10,'nbin')%no
histfit(US,10,'normal')%no, show
histfit(US,10,'poisson')%no
histfit(US,10,'rayleigh')%no, show
histfit(US,10,'rician')%no
histfit(US,10,'tlocationscale')%no
histfit(US,10,'wbl')%no

% chosen distribution: exponential
% generation of random paths for the factor
pd_US=fitdist(US,'exponential')
```

```
US_s= random(pd_US,n,1)
hist(US_s)

%% GBP Bills

UK=UK20032019(120:end)*0.01

%choose correct distributional assumption
histfit(UK) %look at the distribution, normal does not fit well
histfit(UK,10,'beta')%no
histfit(UK,10,'burr')%no
histfit(UK,10, 'birnbaumsaunders')%no
histfit(UK,10,'exponential')%no
histfit(UK,10,'ev')%no
histfit(UK,10,'gamma')%quite good
histfit(UK,10,'gev')%no
histfit(UK,10,'gp')%no
histfit(UK,10,'inversegaussian')%no
histfit(UK,10,'logistic')%no
histfit(UK,10,'loglogistic')%no
histfit(UK,10,'lognormal')%no
histfit(UK,10,'nakagami')%no
histfit(UK,10,'nbin')%no, must be integer
histfit(UK,10,'normal')%not so bad, but no
histfit(UK,10,'poisson')%no
histfit(UK,10,'rayleigh')% not really
histfit(UK,10,'rician')% not bad
histfit(UK,10,'tlocationscale')%no
histfit(UK,10,'wbl')%no

% chosen distribution: rician
% generation of random paths for the factor
pd_UK=fitdist(UK,'rician')

UK_s= random(pd_UK,1,n)
hist(UK_s)


%% Porfolio

%initial value
P_0=(Q_uk)/(1+UK_0)^T-Q_us/(1+USA_0)^T*S_0

P=ones(1,n)'*P_0%preallocate space

%compute simulated portfolio value
for i=2:n
```

```
P(i)=(Q_uk)/(1+UK_s(i))^T-Q_us/(1+US_s(i))^T*S_s(i)
end

plot(P)
Pl=zeros(1,n)%preallocate space

%compute profits and losses
for i=2:n
Pl(i)=P(i)-P_0
end

%compute VaR
Pl_ord=sort(Pl)'
hist(Pl_ord)
VaR= Pl_ord(50)
```

4.5.3 Advantages and disadvantages

Main advantages and disadvantages of Montecarlo techniques for VaR have already been described in the previous pages and will be summarized here.

The most crucial advantage of Montecarlo VaR compared to both variance-covariance methods and historical simulation approach is related to distributional assumptions. In Montecarlo VaR, in fact, the analyst is free to choose the distribution he thinks the market factor will have in the future. This is a clear improvement of the variance-covariance approach since this distribution needs not to be Normal and we have shown that unjustified normality assumption leads to clear underestimations of value at risk. This is feature is an advantage also compared to the historical simulation approach. In Montecarlo models, in fact, it is possible to specify a distribution for market factors that is different from the historical one, if deemed necessary. The starting point for the evaluation of the distribution usually continues to be historical data, but the analyst is free to change the value of the parameters or the shape of the distribution itself. This advantage comes however, with a precise treat. While it is undoubtedly useful to be able to choose the desired distribution of market factors and the relative parameters, this may be a dangerous exercise. If, in fact, the choice does not forecast well the future evolution of market factors, then VaR estimations are entirely unreliable.

Another important pro is that, with Montecarlo techniques, it is possible to evaluate the VaR of complex and non-linear assets, while using the other two techniques, it is either impossible or too burdensome.

The major disadvantage of this technique is that it is computationally costly. If, for example, the portfolio is exposed to many risk factors, then it is necessary to estimate their probability distribution one-by-one and then, for each of them, generate random paths. This is time-consuming and costly in terms of computational power. Moreover, if a reliable estimate is wanted, the simulation should be run

several times, and this again increases computational costs. However, recall that Montecarlo methods may be the only available option if the portfolio is non-linearly related to market factors.

Chapter 5

Empirical performance of VaR models

The aim of this final chapter is to analyze the comparison among the three main techniques for the computation of value at risk, in order to understand which one is best suitable to capture the level of exposure of a portfolio to market risk.

It has to be stressed that no one of these techniques has proven to be superior to the others. As already mentioned in 4, in fact, those techniques are different at least with respect to their assumptions, complexity, computations and level of reliance on historical data. By construction, then, they yield different results, and the choice usually depends on the inherent characteristics of the portfolio to be evaluated. It is indeed challenging to set if and how one technique is superior to the others. What is sure is that there is no way to give a comprehensive answer to this question. That is, it is not possible to conclude that one method is superior to the others with respect to all existing portfolios.

In this chapter, the scope of this question is narrowed. The aim is to assess the performance of the three techniques for the computation of VaR with respect to various dimensions.

First, an empirical analysis of the Italian stock market is performed. The analysis is based on the computation of Montecarlo, Variance-Covariance, and Historical VaR of the FTSE MIB index. This study, not available for the Italian stock market to the best of the author's knowledge, is aimed at assessing which method is most suitable for the characteristics of the Italian stock market.

As a second step, the study is replicated for the SP500 Index to draw comparisons with a larger and more developed stock market.

Finally, a deep dive on Montecarlo techniques is performed. Considering again the FTSE MIB, Montecarlo VaR is computed based on different distributional assumptions for the index. This has two main goals. First, establishing which assumption yields the best result (i.e., the closest result to real data). Second, this study is a mean for establishing how distributional assumptions can have an impact on VaR estimates.

5.1 Relevance of the topic and literature review

Financial risk management literature has devoted a significant amount of resources to the topic of Value at Risk. There are numbers of papers that describe how this measure works, its shortcomings and its regulatory treatment. Unfortunately, the procedure is seldom applied to real data, and thus there is no clear evidence on the preferred technique for the computation of VaR. It is generally thought that being Montecarlo by far the most sophisticated approach to the computation of VaR, it is superior to the others. However, it should be clear that complexity is not a guarantee of superior performance.

Even if literature regarding empirical studies of VaR is relatively scarce compared to more holistic approaches to the same risk measures, some examples are worth mentioning.

Lambadiaris et al. (2003) analyze the performance of Montecarlo, Historical and parametric approaches to VaR applied to two Greek portfolios: one made only of stocks and the second composed only of bonds. The findings are mixed, but for the stock portfolio, Montecarlo seems to be superior, for some holding periods and confidence intervals.

Kuester et al. (2005) apply a similar study to the NASDAQ index. Some innovations (mainly related to time-varying volatility) are introduced.

Bams et al. (2002) introduce a new approach to the classification of VaR techniques. They, in fact, suggest dividing VaR models into sophisticated and non-sophisticated tail models. Those models are then applied to exchange rates. They surprisingly find unsophisticated models to be superior.

Hendricks (1996) is a forerunner in empirical applications of VaR. He was, in fact, one of the first scholars to analyze the performance of various VaR models in exchange rate portfolios. Results found in this study are mixed and do not lead to the primacy of any of the methods over the others.

Kanwer et al. (2006) and Kilic (2006) consider the performance of VaR models, respectively on the Pakistani and Turkish stock exchange.

Engle and Gizychi (1999) replicate a similar study on the Austrian banking market, while Bredin and Hyde (2004) considered Irish portfolios.

Asamoah et al. (2016) evaluate the performance of historical and Montecarlo VaR via backtesting. The finding is a clear out-performance of Montecarlo techniques.

Diamandis et al. (2011) evaluate different VaR models for long and short trading positions using some test that are also used in this work.

Gaglianone et al. (2011) review many available tests for VaR backtesting, pointing out that binary tests sacrifice much information.

5.2 A comparison of VaR techniques: the Italian Stock market

In this section, the first set of empirical studies is performed. The ultimate goal is to analyze how the three basic approaches to the computation of VaR perform on a sample of the leading Italian stock market index, the FTSE MIB. Those three approaches will be compared based on different measures. This task is indeed not trivial because once a VaR forecast measure is available, it is not immediately possible to test how it will perform in the future. For this reason, what is usually performed is backtesting, i.e., evaluate how the estimate performs if compared with historical data. For this reason the sample is divided into two parts: the first one is used as "historical", meaning that it will be used to estimate VaR [1]. The second component is used for backtesting, meaning that, based on this time series P&L will be computed and compared with VaR estimates resulting from the application of the three methodologies. Backtesting is typically performed by banks, and it is an essential pillar of banking regulation and supervision. It has to be clear that backtesting does not provide information about the ability of the model to predict future unusual movements in portfolio value. Anyway, it is a useful tool to evaluate portfolio performance in normal market conditions.

Since the object of the analysis will be data related to the FTSE MIB, it is worth spending some words about it.

5.2.1 FTSE MIB

FTSE MIB stands for Financial Times Stock Exchange Milano Indice di Borsa. It is the most relevant index in the Italian stock market and it was introduced on 31 December 1992 with the name of COMIT. The name FTSE MIB was given for the first time on 1 June 2009. Together with FTSE Italia Mid Cap and FTSE Italia Small Cap it forms the FTSE Italia All-Share. This index is composed by the shares of the 40 companies with highest market capitalization traded in the MTA, Mercato Telematico Azionario, and on the MIV, Mercato degli Investment Vehicles. It represents approximately 80% of the Italian stock market. This is a critical point for this analysis. We can, in fact, conclude that this index represents a fairly good approximation of the Italian stock market in general. The composition of the index is rebalanced quarterly. Inclusion decisions are also based on liquidity considerations and on the industry to which the stock belongs. The aim is in fact to create an index which includes all major industries in the Italian reality. The weight of each stock in the index depends on market capitalization (market-cap weighted index), corrected for float.

This index will be used as representative of the Italian stock market throughout the chapter.

[1] The concrete use of the historical time series varies from method to method. In the Variance-Covariance and Montecarlo methods, time series are used to estimate the parameters inputs, while in the Historical approach, time series are directly used to estimate VaR.

5.2.2 Data

Daily adjusted closing prices for FTSE MIB are taken from Yahoo Finance. The relevant period is 25 October 2016 to 1 January 2018[2]. In this time frame, the maximum observable value was 2.3046, the minimum 1.6217. The mean value of the index is 2.0485, with a standard deviation of 17.61%. The median is 2.0961, skewness and kurtosis are respectively -0.7695 and 2.7771. Negative skewness describes a distribution which is asymmetric from the left. The mode is 1.9247. Non-zero kurtosis implies fatter tails compared to the Normal case.

Summary statistics are provided in table 5.1.

TABLE 5.1: Summary statistics of the FTSE MIB 25 October 2016-1 January 2018

maximum	2.3046
minimum	1.6217
mean	2.0485
standard deviation	17.61%
skewness	-0.7695
kurtosis	2.7771
median	2.0961
mode	1.9247

The plot below (figure 5.1) describes the evolution of the index in the period 25 October 2016-1 January 2018.

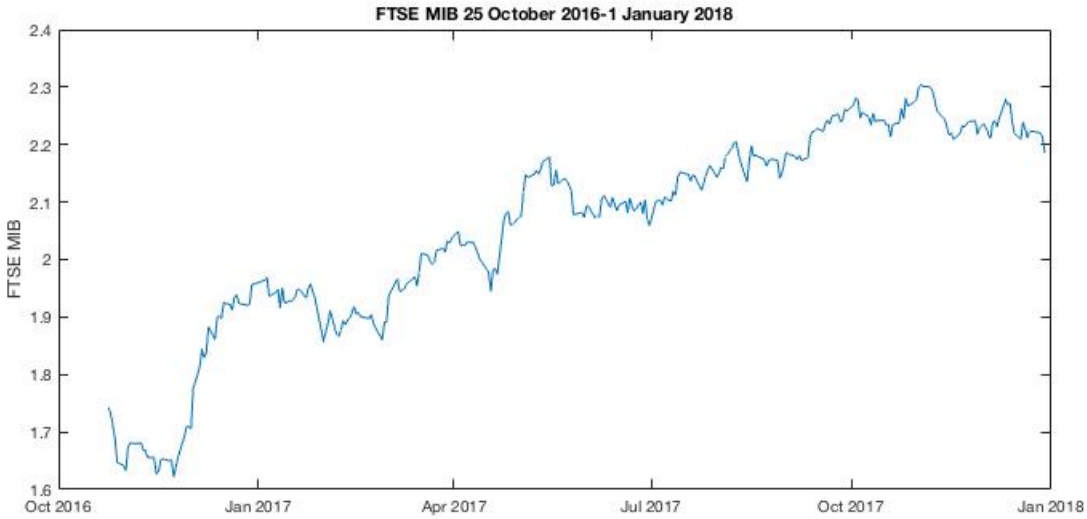

FIGURE 5.1: Evolution of FTSE MIB 25 October 2016-1 January 2018

The trend of the graph is clearly positive but also highly volatile. Peaks and falls are in fact registered often.

[2]Data are normalized by diving for 1000 every data point

As already mentioned, the dataset has been divided into two sections. The first component of the dataset will be used as the basis for the computation of VaR. Specifically, in the Variance-Covariance method, it will be used in order to compute the parameters of the Normal distribution used to compute the VaR [3]. In the Historical simulation approach, this first section of the dataset will be used to compute daily changes of the index, which will then be used for the estimation of the P&L distribution. Finally, in the Montecarlo method, this time series is used for the estimation of the parameters of the distribution that is intended to be representative of future evolutions of the index. The second portion of the dataset will be used for the assessment of model performance. This means that, with the various statistical tests that will be performed, the estimated VaR will be compared with actual losses incurred by the index in this period. We can then say that this second component of the dataset is the backtesting dataset. Summary statistics are described in Table 5.2 below for the first portion of the dataset. Those statistics will be needed as inputs in the following stages of the analysis.

TABLE 5.2: Summary statistics of the FTSE MIB in-sample period

maximum	2.1788
minimum	1.6217
mean	1.9193
standard deviation	14.54%
skewness	-0.4534
kurtosis	2.5442
median	1.9364
mode	1.9247

The histogram in Figure 5.2 describes the distribution of the FTSE MIB index in the first half of the dataset.

5.2.3 Methodology

The main idea behind the methodology of this study is the computation of VaR using the three different techniques, according to the steps described in 4 and then the comparison of the results in order to assess which method does the best job in capturing the risk of the Italian stock market.

For all the methods, α is set to be 5%, and the assumed holding period is one day. The number of VaR estimates per technique is set to $k = 25$.

A short overview of the computational technique used for each method is given, but the heart of this paragraph is the description of the statistics that are used in order to evaluate the performance of each model.

[3] In this section we will not challenge the Normality assumption.

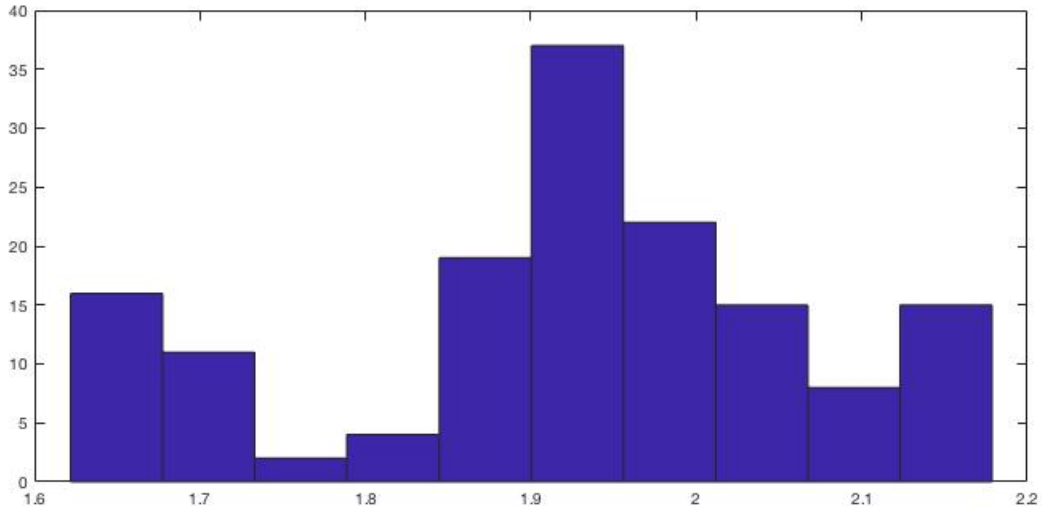

FIGURE 5.2: Empirical distribution of FTSE MIB

First, as already mentioned, the dataset consists of 299 observations (daily price data). Of these 299, one is taken as the current value of the index S_0 and a total of 25 observations are used as out-of-sample observations (i.e., back-testing purposes). This means that we are left with 299-1-25=273 observations that represent the historical time series on which all analysis will be based. As already mentioned, historical data are differently used by the three methods.

When applying the Variance Covariance approach, we assume that the distribution of the index is Normal. We then are already in the position to expect non-satisfying results for this technique. Figure 5.2 has shown that the distribution of the FTSE MIB does not resemble a Normal. As described in 4, VaR is here computed according to the formula:

$$VaR(t, t+\Delta) = -(\mu + z_{1-\alpha}\sigma) \tag{5.1}$$

μ and σ are respectively the mean and standard deviation of the historical FTSE MIB and are computed based on a 25-days horizon with 25-days rolling window.

In the Historical simulation approach, simulated values of the index are computed based on sub-samples of $j = 100$ observations, with 5-days rolling windows. As explained in detail in 4, the simulated distributions are generated based on the application to actual values of the percentage changes of the Index.

Montecarlo simulation deserves some greater attention since assumptions will be changed compared to 4. In order to generate random paths for the FTSE MIB, in fact, we will assume that the process that best describes its future evolution is the Geometric Brownian Motion. Given a sequence of stock prices S_t with $t = 1, 2, ..., T$,

the rate of change of the stock price is defined as follows:

$$\frac{dS_t}{S_t} = \mu_S dt + \sigma_S \sqrt{dt}\epsilon_t \tag{5.2}$$

where μ_S is the mean of the distribution, σ_S is the standard deviation, dt is the time interval (in this case one day) and $\epsilon_t \sim N(0,1)$. For each simulation run, we will generate $n = 1000$ possible values for the Stock price. The Standard Normal element introduces the randomness. Once the random trajectories for the FTSE MIB have been generated, then the procedure overlaps with the one described in 4.

Statistical tests When financial institutions compute value at risk, the aim is to set aside enough capital to cover unexpected losses. It is fundamental to keep this in mind when assessing the quality of a VaR technique. A good VaR model is, in fact, challenging to define. The model doesn't have to be able to capture the exact future amount of the loss, what is crucial is that the model does not underestimate the loss[4]. Such a situation would mean that the bank has not enough capital to cover the unexpected loss. Consequently, when VaR models are evaluated, their forecasts are compared with incurred losses in order to assess whether the application of the model has led to the estimation of an amount of capital which is adequate.

This procedure is necessarily a backtesting, since it compares VaR forecasts with past actual losses data. Backtesting is in fact defined as a statistical procedure whose aim is to determine whether the ex-post observed loss is in line with was predicted by the model. The optimal situation would be to compare VaR estimates with certain future losses. This is of course not possible until the moment in which the loss is eventually incurred. For this reason, Basel accords (1996) have introduced the procedure of backtesting as an essential pillar of model supervision. Even if backtesting does not give information about the future ability of the model to forecast losses, it gives crucial information about the performance of the model when applied to historical data.

Given the importance of VaR measure when assessing the exposure to market risk and given the continuously growing number of VaR models, scholars have proposed various statistics that evaluate the quality of a model. Kupiec started this body of research in 1995.

Most of these statistics are based on the concept of failure rate (also referred to as hit or violation). Define a variable $VaR = \{VaR_1, VaR_2, ..., VaR_n\}$, $n = 1, 2, ..., 25$ that contains the 25 estimates of VaR resulting from the analysis. Define $P\&L = \{P\&L_1, P\&L_2, ..., P\&L_n\}$, $n = 1, 2, ..., 25$ as the out-of-sample profits and losses of the portfolio, i.e. the incurred losses in the backtesting period. A problematic situation occurs if, on a date, the estimated VaR is not able to cover the incurred loss.This happens if the absolute value of VaR is lower than the absolute value of the loss, i.e. if $|VaR_i| < |P\&L_i|$, for some $i \in [1, ..., 25]$. If this situation occurs, then we say that

[4] At least from a supervisory perspective

the model has incurred in a failure. By the definition of confidence level, it is already known that the expected probability of observing a failure, i.e., $|VaR_i| < |P\&L_i|$, is α: $P(|VaR_i| < |P\&L_i|) = \alpha$. If for all the 25 VaR estimates, we compare them with the relevant incurred loss, then it is possible to create a vector that counts the number of failures in the model. This random variable is the most important starting point for every VaR model evaluation. Define then $F(\alpha)$ as a 1xn vector that assumes value 1 is $|VaR_i| < |P\&L_i|$ (i.e. if a failure occurs) and 0 otherwise. This random variable can formally be defined as:

$$F(\alpha) = \{I_{\{}|VaR| < |P\&L|\}\}^{T=1,2,\ldots,25}$$

When constructing this sequence, we should expect to find a 1 with probability α and a 0 with probability $1 - \alpha$. The failures contained in the vector $F(\alpha)$ are i.i.d. Bernoulli random variables with parameter $1 - \alpha$. In the tests that will follow, the assumption that is to be tested is only this one.

Now the tests that have been performed on the three models will be described. They are all based on this concept of failure.

Binomial test The Binomial test is the most communicative. In this test, in fact, we compare the number of failures to the expected number of failures, assuming that failures follow a Binomial distribution. Define x as the number of times in which $|VaR_i| < |P\&L_i|$, N as the length of the vectors VaR and P&L (i.e., the number of times the estimation has been performed) and p as α, where α is the confidence level used in the estimation of VaR (in our case 5%). Assume that the failures are independent. Then the vector that contains the failures follows a Binomial distribution with parameters p and N. Define now the observed failure rate as the ratio between the observed number of failures and the total number of trials $\hat{f} = \frac{x}{N}$. The expected number of failures under the Binomial distribution is simply Np, and the standard deviation of the number of failures is $\sqrt{Np(1-p)}$. We want then to test the null hypothesis that the expected failure rate is equal to the observed failure rate (or, at the same way that the total number of observed failures is equal to the expected number of failures):

$$H_0 : \hat{f} = p = \frac{x}{N}$$

This means that we want to test that the number of failures x follows a Binomial distribution $f(x) = \binom{N}{x} p^x (1-p)^{N-x}$. As N increases and under the null hypothesis, the test statistic

$$Z = \frac{x - Np}{\sqrt{Np(1-p)}} \sim N(0,1) \tag{5.3}$$

is distributed as a Standard Normal. Once the desired level of confidence has been chosen, it is only necessary to compare the observed value of the test statistic with the critical value of the Standard Normal. If the observed value is lower than the critical value, then the null hypothesis cannot be rejected and thus the model should

be considered as well-performing. This test is indeed an elementary test for one-population mean in which the expected number of failures is Np and $Np(1-p)$ is their variance.

Even if we will compare the observed value of the test statistic with the relevant critical value, it is also possible to perform the test using the p-value[5]. In this case, it is necessary to introduce the concept of tail probability. The tail probability is defined as the probability that the standard normal exceeds the value of the observed test statistic: $TP = 1 - \Psi(Z_{obs})$. Since this test is double-sided, the p-value is then equal to $p - value = 2TP$. As usual in hypothesis testing, if $p - value > \alpha$ then there is not enough evidence to reject the null hypothesis and thus the VaR model should be considered as acceptable.

Kupiec's tests Kupiec (1996) proposed two different tests. The first one, the Proportion of Failures (POF) test, basically gives the same information of the Binomial test, since it evaluates the empirical proportion of failures compared to the expected one. The null hypothesis is then the same as the Binomial test. This implies that we are still checking whether the observed number of failures is significantly distant from the expected one. The test takes the form of a likelihood ratio test (and not anymore a test for the mean) where the statistic is:

$$LR_{POF} = -2log(\frac{(1-p)^{N-x}p^x}{(1-\frac{x}{N})^{N-x}(\frac{x}{N})^x}) \tag{5.4}$$

Under H_0, it is distributed as a χ^2 with one degree of freedom. Again, if the observed value of the statistic is lower than the critical value, then there is not enough evidence to reject H_0 and thus the model can be considered as well-performing. The Binomial and POF tests thus carry the same information.

As the Binomial test, Kupiec's POF test provides information only with respect to the number of failures. This implies that the moment in which the failure occurs is not relevant. Since the moment in which the first failures occur is indeed of great interest, Kupiec also proposed a second test, which provides some additional useful information. This test is again a likelihood ratio type test and it is usually called Time until first failure. As the name may suggest, this test evaluates after how many successes, the first failure occurs. Specifically, the test aims to check whether the moment in which the first failure occurs is consistent with the VaR confidence level. The test statistic is this time in the form:

$$LR_{TUFF} = -2log(\frac{p(1-p)^{n-1}}{(\frac{1}{n})(1-\frac{1}{n})^{n-1}}) \tag{5.5}$$

Under H_0, this statistic is distributed according to a χ^2 with one degree of freedom and again, if the observed value of the statistic is lower than the critical value, then

[5]Recall that in statistics the p-value is defined as the probability of observing values that are equal or more extreme of the one observed in the test statistics, assuming that the null hypothesis is true.

there is not enough evidence to reject H_0 and thus the model can be considered as well-performing.

Christoffersen's test The tests above consider the first or the total number of failures in isolation. However, an essential piece of information is the relative distance among failures. In particular, the level of clustering of failures may be an essential piece of information for risk managers. If failures tend to be subsequent to each other, in fact, this means that the bank may not be able to cover losses for many consecutive days. This is a significant concern and some test are needed in order to address it. Christoffersen (1998) was the first to work in this direction by considering the interaction among failures. His test is, again, a likelihood ratio type test. In order to define the test statistic, it is necessary to define several variables. n_{00} is the number of periods with no failures followed by a period with no failure. n_{10} is the number of periods with failures followed by a period with no failure. n_{01} is the number of periods with no failures followed by a period with a failure. n_{11} is the number of periods with failures followed by a period with a failure. $\pi_0 = \frac{n_{01}}{(n_{01}+n_{00})}$ is the probability of having a failure in t conditioned on the fact that no failure happened in $t-1$. $\pi_1 = \frac{n_{11}}{(n_{10}+n_{11})}$ is the probability of having a failure in t conditioned on the fact that a failure happened in $t-1$. Finally, $\pi = \frac{n_{01}+n_{11}}{(n_{00}+n_{01}+n_{10}+n_{11})}$ is the probability of having a failure in period t. The test statistic is defined as

$$LR_{CCI} = -2log(\frac{(1-\pi)^{n_{00}+n_{10}}\pi^{n_{01}+n_{11}}}{(1-\pi_0)^{n_{00}}\pi_0^{n_{01}}(1-\pi_1)^{n_{10}}\pi^{n_{11}}}) \tag{5.6}$$

and it is distributed as a χ^2 with one degree of freedom and again, if the observed value of the statistic is lower than the critical value then there is not enough evidence to reject H_0 and thus the model can be considered as well performing.

Christoffersen's test and the POF test can also be combined obtaining the conditional coverage test, $LR_{CC} = LR_{CCI} + LR_{POF}$, which is distributed according to a χ^2 with two degrees of freedom. In this kind of test, we combine two relevant information: the correct number of failures and their relative distribution.

5.2.4 Results

Figures 5.3,5.4 and 5.5 plot the estimated VaR according to the three techniques and realized losses. As a first observation, we can immediately disregard the Variance Covariance method since it always underestimates the realized loss and thus it is completely misleading. For the other two methods, before going in detail with test results, we can start with a qualitative analysis. First, it can be seen that both methods display some failures, meaning that in both cases, we have observations that underestimate the realized loss. If we look only at the plots, it seems that Montecarlo is superior in estimating VaR since the number of failures is lower. For historical simulation, in fact, underestimation of risk is observed in the middle and at the end

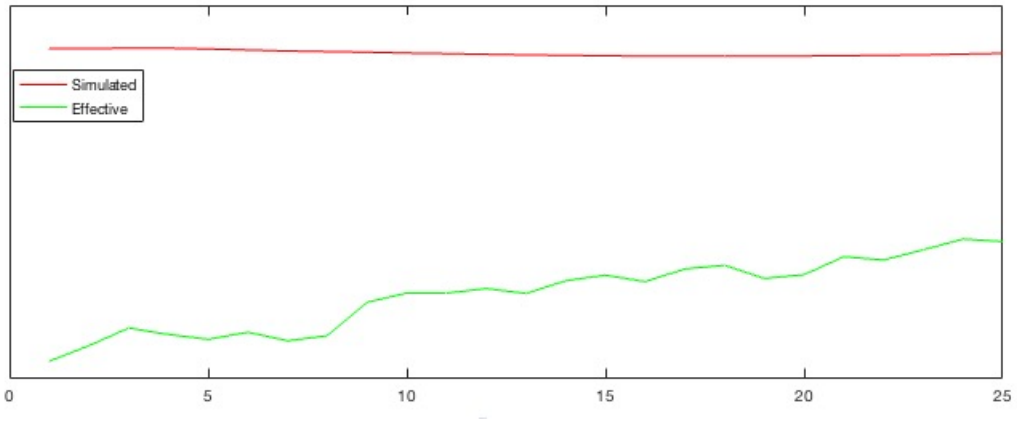

FIGURE 5.3: VaR estimates from Variance-Covariance method and realized profits or losses

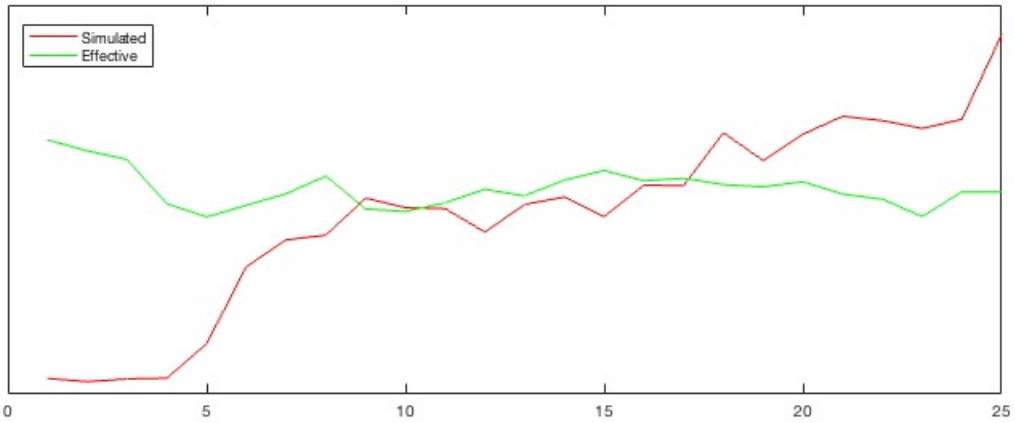

FIGURE 5.4: VaR estimates from Historical Simulation method and realized profits or losses

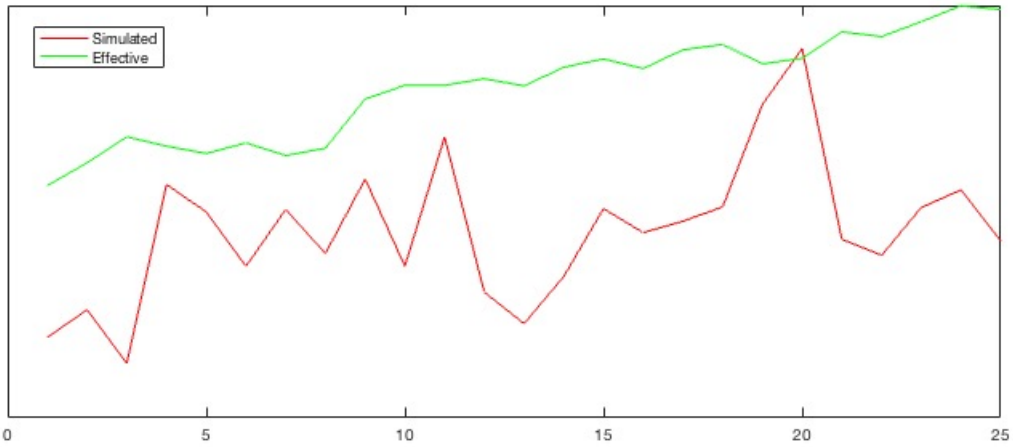

FIGURE 5.5: VaR estimates from Montecarlo Simulation method and realized profits or losses

of the dataset. Specifically, the last part of the sample is highly problematic because it would imply an underestimation that continues for several consecutive days. This may be a serious source of concern. On the other hand, Montecarlo estimates seem to be much more volatile and constantly above the actual losses, but in one case.

We have to verify that test results support this observation.

Binomial test The critical value of this test is 1.6449 and the test statistics for the two techniques are displayed in table 5.3. Historical simulation fails the test, while Montecarlo passes it. This first result is in line with the observations raced above and it is indeed not a surprise: the lower the number of failures, the lower the magnitude of the observed test statistic.

TABLE 5.3: Binomial test statistic values FTSE MIB

Montecarlo	Historical
0.2294	8.0296

Kupiec's tests Observed statistics for these two tests are displayed in table 5.4. The critical value is 3.8415. Montecarlo performs well in both tests, while historical simulation passes only the time until first failure test.

TABLE 5.4: Kupiec's tests statistic values FTSE MIB

	Montecarlo	Historical
LR_{POF}	0.0563	27.8029
LR_{TUFF}	0.0027	0.6812

For Kupiec's POF test, the outcome is not indeed a surprise. This test carries, in fact, the same information of the Binomial test. Non-coherent results between the two tests would have been a concern. Kupiec's second test (Time until first failure) is a bit more informative. This test is about the moment in which the first failure happens. The total number of failures is thus not taken into account. Given this premise, it is possible to infer that both methods pass the test because the first failure is roughly in the middle of the sample in both cases. Usually, in fact, bad performance in this kind of test comes from a failure which happens at the beginning of the sample.

Christoffersen's test The critical values for Christoffersen's and CC are respectively 3.8415 and 5.9915. We can thus conclude that while the Montecarlo method is successful in both tests, Historical simulation fails.

TABLE 5.5: Christoffersen's tests statistic values FTSE MIB

	Montecarlo	Historical
LR_{CCI}	0.0889	13.6970
LR_{CC}	0.1452	41.4999

In this last set of tests, what matters is not only the absolute number of failures but also their distribution in the sample. In the case of historical simulation, in fact, the test is failed since failures tend to be much concentrated at the end of the dataset.

5.2.5 Conclusions

The results of the analysis have clearly pointed out that Montecarlo methods are superior in capturing the level of exposure to market risk compared to historical simulation when the Italian stock market index is taken into account. This implies that in models, reduced complexity comes in every case with a cost. However, it is essential to point out some significant shortcomings of this study. First, models have been evaluated from a supervisory perspective. In this analysis, in fact, a good model is a model which does not lead to underestimation of risk. Supervisors, in fact, are concerned about banks not being able to cover unexpected losses. This implies that model accuracy was not a concern in this analysis. It was not evaluated whether the model is able to predict accurately future losses, but only its ability to cover the loss. However, from a bank's perspective, overestimation of risk is another concern, since, if capital is allocated to cover unexpected losses, then it cannot be invested in more profitable projects.

Second, the universe of VaR models is now incredibly growing. Future research should then focus not only on "traditional" models such as those analyzed in this study but on the recent variations and modifications that have been proposed. Many models, for example, introduce some technical complications that are indeed able to improve model performance significantly. One famous example is the introduction of time-varying volatility both in Variance-Covariance and in Montecarlo models.

Finally, this study is only focused on equities. It would then be of great interest the analysis of the behavior of debt or derivatives portfolios.

5.3 Replication of the study on S&P500 index

In this section, the analysis performed in 5.2 will be replicated for another index, the American S&P500. The methodology and statistics to use are thus the same. The aim is to compare two markets that are different in terms of size in order to spot eventual discrepancies compared to the results that have been described for the Italian Stock market.

5.3.1 S&P500

The S&P500 is well-known market index of the American stock market. This index has been used as a benchmark in many studies since it represents a proxy for the largest stock market in the World. It thus deserves some attention. As the FTSE MIB for the Italian case, the S&P500 is usually considered as a reliable proxy for the state of the American economy. It was introduced for the first time on 1 January 1957 by Standard & Poor's. This index is composed of the 500 largest companies quoted in the US stock exchanges [6]. The weight of each company in the index depends on the market capitalization. It is thus a cap-weighted index, as the FTSE MIB. Changes in the composition of the index are exclusive competence of a specific committee. Inclusion decisions are not only based on market capitalization. As in the FTSE MIB case, in fact, stocks to be included must be liquid. Moreover, considerations related to sectorial representation and float are taken into account too. This index will be used as representative of the American stock market throughout the chapter.

5.3.2 Data

Source of data for the S&P500 is again Yahoo Finance. The data used are daily adjusted closing prices. The time interval considered is again 25 October 2016 to 1 January 2018. In this time frame, the maximum observable value was 2.6902, the minimum 1.8291. The mean value of the index is 2.2711, with a standard deviation of 2.0658%. The median is 2.2652, skewness and kurtosis are respectively 4.61% and 2.1027. The mode is 2.0667. Summary statistics are provided in table 5.6.

TABLE 5.6: Summary statistics of the S&P 500 25 October 2016-1 January 2018

maximum	2.6902
minimum	1.8291
mean	2.2711
standard deviation	2.0658%
skewness	4.61%
kurtosis	2.1027
median	2.2652
mode	2.0667

The plot in Figure 5.6 describes the evolution of the index in the period 25 October 2016 to 1 January 2018. Again, as in the case of FTSE MIB, the trend is generally increasing, but much less volatile.

As already done for the FTSE MIB, the dataset is divided in two sections. The first one is used as a starting point for the application of VaR techniques. Considerations in 5.2 related to the use of this first dataset in the different techniques are still valid.

[6] Note the significant size difference with the FTSE MIB which includes 40 stocks.

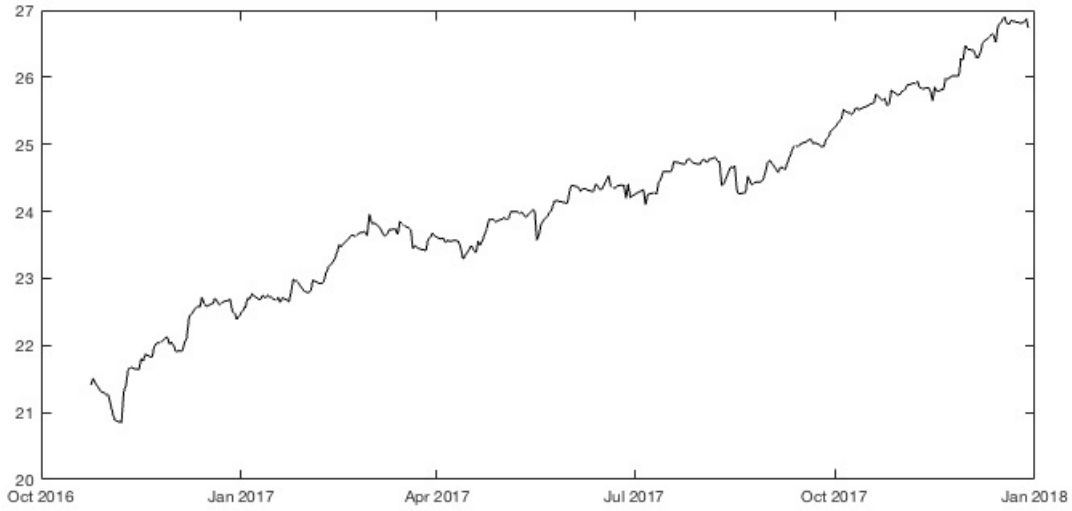

FIGURE 5.6: Historical evolution of SP500 whole sample

The second one is used for backtesting purposes. Summary statistics are described in table 5.7 below for the first portion of the dataset. Those statistics will be needed as inputs in the following stages of the analysis.

TABLE 5.7: Summary statistics of the S&P 500 in-sample period

maximum	2.1750
minimum	1.8291
mean	2.0367
standard deviation	2.88%
skewness	-0.51
kurtosis	2.3516
median	2.0571
mode	2.0667

The histogram in Figure 5.7 describes the distribution of the S&P 500 index in the first half of the dataset.

5.3.3 Methodology

Reference is made to paragraph 5.2.3 since the methodology applied is perfectly overlapping.

5.3.4 Results

Figures 5.8,5.9 and 5.10 plot the estimated VaR according to the three techniques and realized losses. Again, we can immediately disregard the Variance Covariance method since it always underestimates the realized loss and thus it is completely misleading. This is already a good symmetry with the Italian case. Applying the

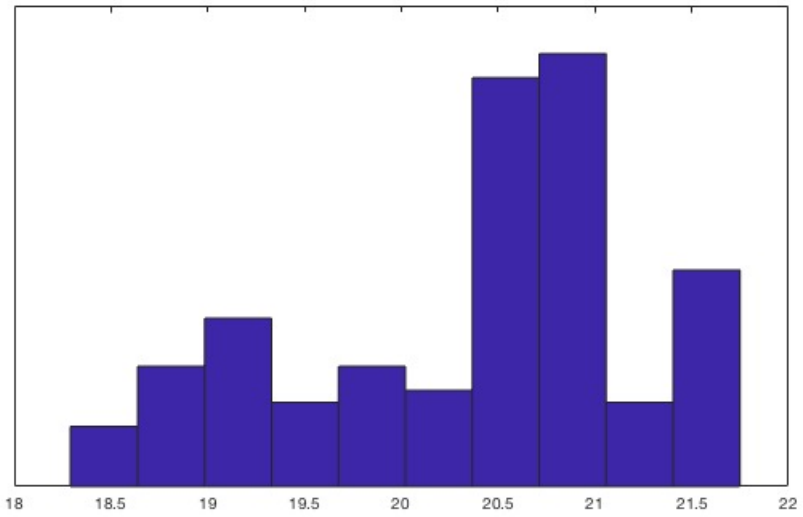

FIGURE 5.7: Distribution of SP500 in-sample period

same structure used in section 5.2, we can again start from a qualitative analysis.

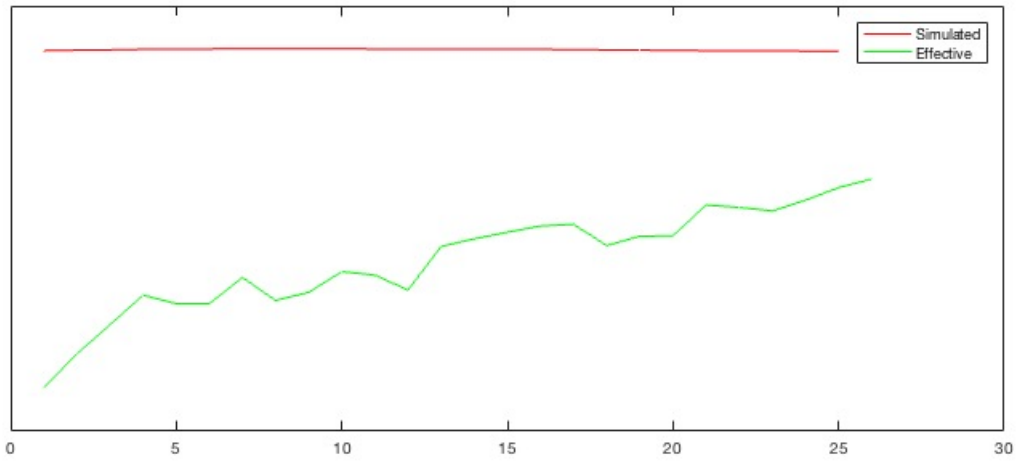

FIGURE 5.8: VaR estimates from Variance-Covariance method and realized profits or losses of SP500

First, it can be seen that both methods display some failures, meaning that in both cases, we have observations that underestimate the realized loss. For Historical simulation, failures seem to be much concentrated at the end of the sample, while for Montecarlo, they are dispersed.

By looking only at the graphs, we can safely state that Montecarlo seems to underestimate losses in a lower number of cases. This is again a symmetry with the Italian stock market. However, compared to the Italian case, the superiority of the Montecarlo method seems to be much less pronounced since failures are much more frequent (4 vs 1). On the other hand, for historical simulation, the quality of the estimation seems to be much deteriorated, with a considerable number of failures.

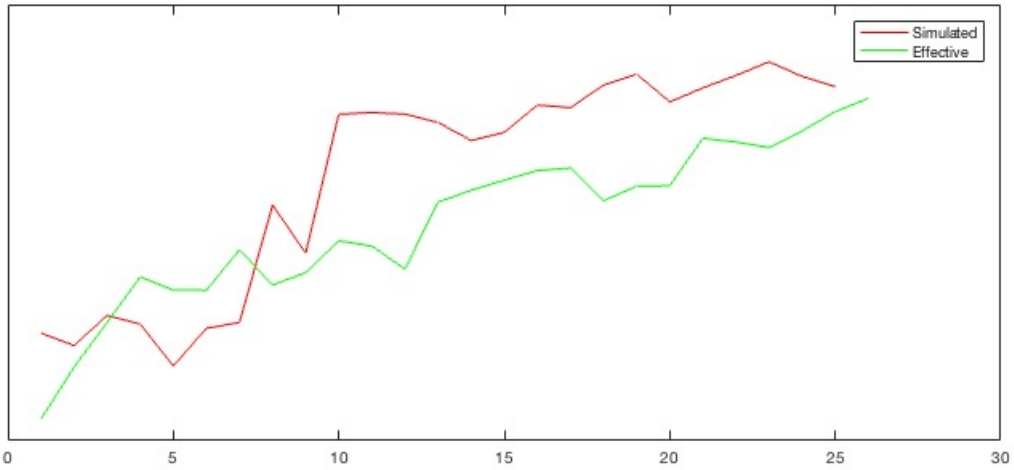

FIGURE 5.9: VaR estimates from Historical Simulation method and realized profits or losses of SP500

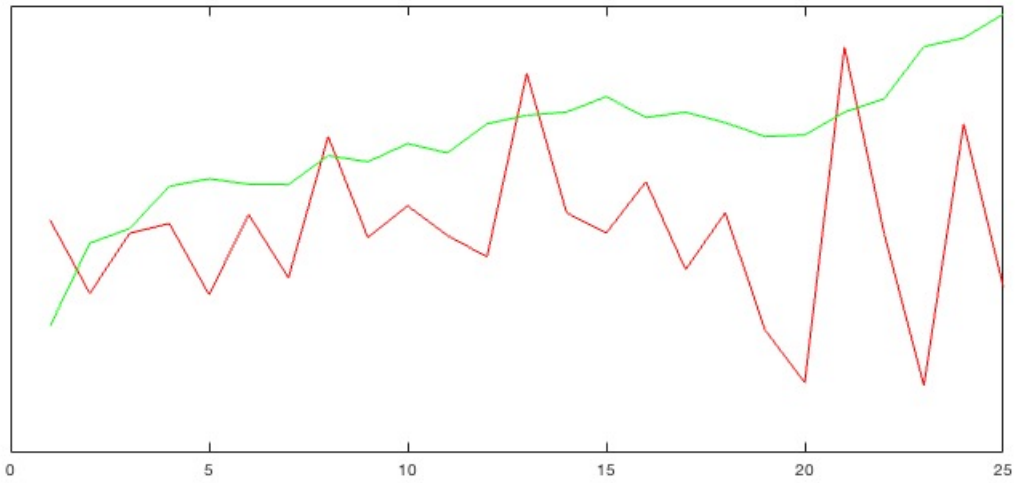

FIGURE 5.10: VaR estimates from Montecarlo Simulation method and realized profits or losses of SP500

Binomial test The critical value of this test is 1.6449 and the test statistic for the two techniques is displayed in table 5.8. Surprisingly, both methods fail the test, meaning that, even if Montecarlo displays less failures, they are still too many compared to what predicted by a binomial distribution. In both cases, test statistics are significantly higher compared to the Italian case.

TABLE 5.8: Binomial test statistic values SP500

Montecarlo	Historical
2.5236	18.1238

Kupiec's tests Observed statistics for these two tests are displayed in table 5.9. The critical value is 3.8415. Both methods fail the two tests. This is indeed not a surprise since Kupiec's test for the proportion of failure gives the same information of the Binomial test. For the time until the first failure test, the results are a consequence of the fact that for both methods, the first failure coincides with the first observation.

TABLE 5.9: Kupiec's tests statistic values SP500

	Montecarlo	Historical
LR_{POF}	4.1367	102.2476
LR_{TUFF}	5.9915	5.9915

Christoffersen's test The critical values for Christoffersen's and CC are respectively 3.8415 and 5.9915. Montecarlo passes both tests, while historical simulation fails both.

TABLE 5.10: Christoffersen's tests statistic values SP500

	Montecarlo	Historical
LR_{CCI}	1.2332	9.4462
LR_{CC}	5.3699	113.6939

5.3.5 Conclusions

The results of this additional study on the US stock market are broadly in line with the ones performed on the Italian stock market. Montecarlo proves again to be the

most accurate method in the ability not to lead to an underestimation of capital. Moreover, the Variance-Covariance (under normality assumption) method proves again to be inadequate.

However, some differences can be spotted. Specifically, even if for the tests proposed in this study the performance of the Montecarlo method is evidently superior if one looks at the plots, it is evident that historical simulation seems to be much more accurate in predicting actual losses. This may come from the fact that history is more informative in the US stock market compared to the Italian case.

Finally, even if the Montecarlo method results again as the best performer, the over performance is not so pronounced as in the Italian case.

5.4 The impact of distributional assumptions on VaR estimates

In 4 it has been stressed that, when a Montecarlo VaR model is implemented, one of the most critical choices to be made is the distribution from which random numbers describing the market factor will be drown. If the assumption does not seem to be supported by data, in fact, the VaR estimate that results is completely unreliable.

It makes then sense to compare empirically how different distributional assumption can impact VaR estimates.

In this final section, VaR will be computed only applying the Montecarlo method, but based on three different distributional assumptions: Geometric Brownian motion, Normal and Log-Normal. The data used will be again from the Italian stock market index FTSE MIB.

5.4.1 Data

Reference is made to 5.2.2 since the dataset used is perfectly overlapping.

5.4.2 Methodology

The methodology used to compute Montecarlo VaR estimates is the same compared to 4. The only difference is that here we apply three different distributional assumptions: Geometric Brownian Motion, Normal and Log-Normal.

For the Geometric Brownian Motion, the procedure maps the one described in 5.2.3. For the Normal, the parameters to be estimated are the mean and the standard deviation, which are estimated based on historical data. For the Log-Normal, the parameters to be estimated are again mean and standard deviation. Finally, the tests performed on the three estimates are the ones described in Section 5.2.3.

5.4.3 Results

Figures 5.11, 5.12 and 5.13 plot the VaR distributions compared to actual losses for the three distributional assumptions (respectively Geometric Brownian Motion, Normal and Log-Normal). We will again start from a qualitative analysis.

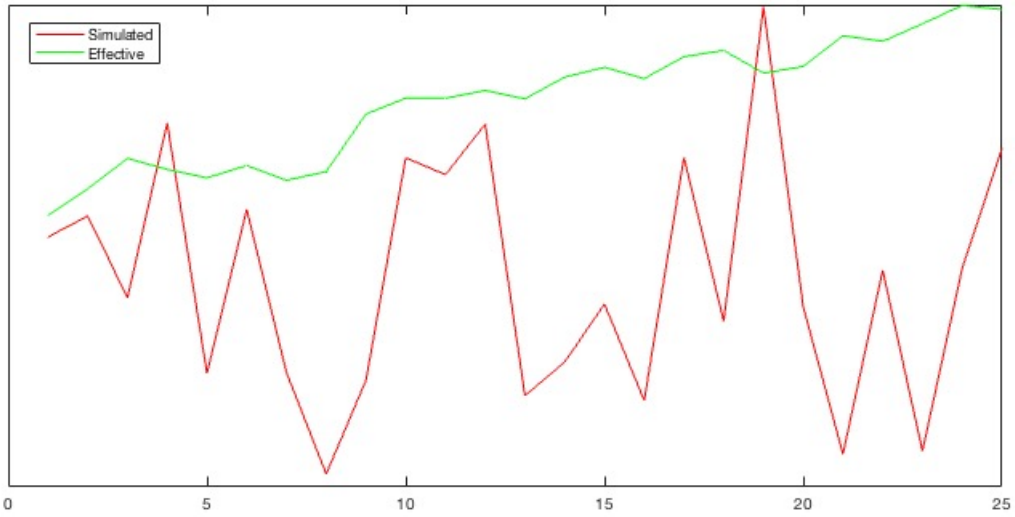

FIGURE 5.11: VaR estimates from Geometric Brownian Motion assumption and realized profits or losses

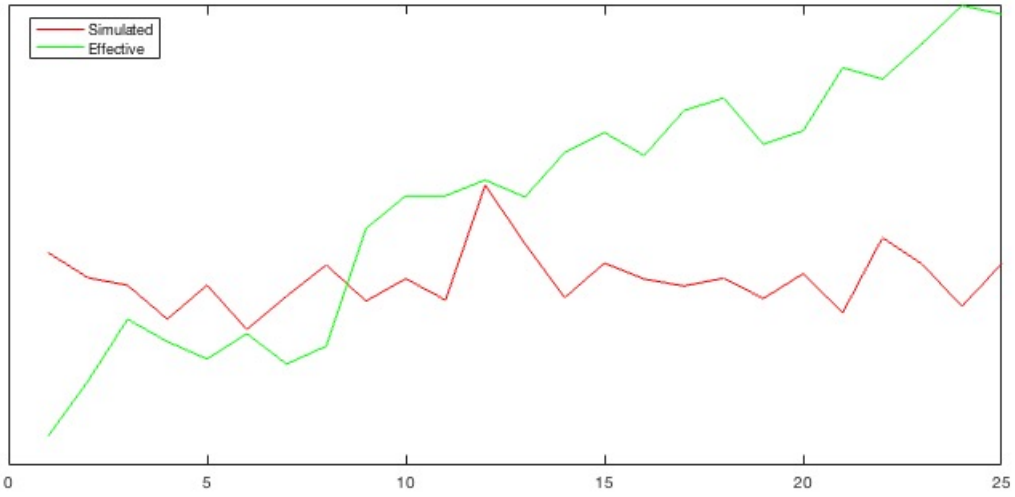

FIGURE 5.12: VaR estimates from Normal assumption and realized profits or losses

The Geometric Brownian Motion (GBM) assumption seems to be the best performer from a supervisory perspective since underestimations of risk (failures) are less frequent compared to the other two methods. Normal and Log-Normal assumptions, on the other hand, display roughly the same behavior since failures are concentrated at the beginning of the distributions. Estimates based on GBM seem to be much more volatile and disperse. Anyway, failures are only two and the magnitude

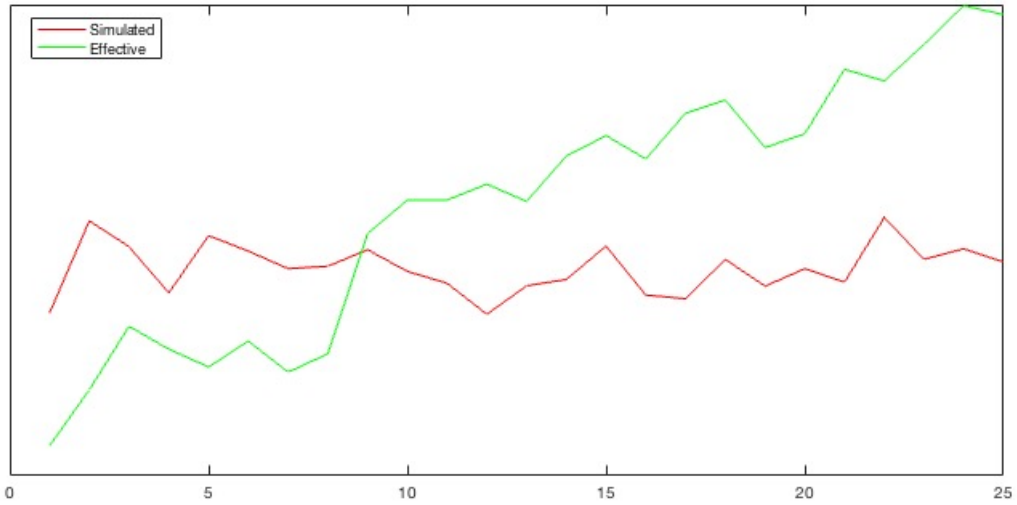

FIGURE 5.13: VaR estimates from Log-Normal assumption and realized profits or losses

of the failures is tiny compared to the other two assumptions. We will now go into test results.

Binomial test The critical value of this test is 1.6449 and the test statistic for the three is displayed in the table below. Montecarlo method under the Normal and Log-Normal assumption fails this test, meaning that the number of failures is significantly higher than what predicted by the Binomial distribution. Under Geometric Brownian motion, the test is passed. This is a consequence of the fact that in the GBM case failures are only two.

TABLE 5.11: Binomial test statistic values: distributional study

GBM	Normal	Log-Normal
0.6882	6.1942	6.1942

Kupiec's tests Observed statistics for these two tests are displayed in the table below (5.12). The critical value is 3.8415. Again, Normal and Log-Normal fail the two tests, while GBM passes them. This is in line with what observed in the Binomial test. For the TUFF test, the failure of Normal and Log-Normal is due to the fact that the first rejection happens immediately.

Christoffersen's test The critical values for Christoffersen's and CC are respectively 3.8415 and 5.9915. GBM Montecarlo passes the two tests. The other two techniques fail both tests.

TABLE 5.12: Kupiec's tests statistic values: distributional study

	GBM	Normal	Log-Normal
LR_{POF}	0.4040	18.3322	18.3322
LR_{TUFF}	2.3776	5.9915	5.9915

TABLE 5.13: Christoffersen's tests statistic values: distributional study

	GBM	Normal	Log-Normal
LR_{CCI}	0.3639	22.2593	22.2593
LR_{CC}	0.7679	40.5915	40.5915

5.4.4 Conclusions

The results of this study have clearly pointed out that Montecarlo VaR computed according to the GBM assumption seems to be superior compared both to the Normal and to the Log-Normal case. This over-performance is extremely pronounced, but it is indeed not a surprise. In financial literature, in fact, it is well known that stock returns hardly display a Normal distribution. A process which is in general considered to be able to better approximate stock returns is the GBM. If the modeling of the process is superior, then the VaR estimate that results is consequently more appropriate.

Another take away of the analysis is that the Log-Normal assumption seems not to have significant advantages over the Normal. In financial literature, in fact, Log-Normal assumption is usually preferred to the Normal but, in this case, results are perfectly overlapping.

5.5 Conclusions

The analyses that have been performed in this final chapter have explained why it was worth deeply discussing random number generators and the Montecarlo method. In the cases that have been analyzed, in fact, Montecarlo seems to be the best performer, i.e., the method that underestimates less frequently the realized loss and thus capital requirements for market risk.

This result is somehow positive because it implies that the model increased sophistication has positive effects on performance. However, some limitations of this study have already been underlined. First, the analysis was performed considering only the stock market. Specifically, a cap-weighted stock portfolio (FTSE-MIB and S&P500 Indexes) was used as test asset. The reason for the use of an index is computational ease since it allows to draw conclusions at portfolio level with no need

to perform portfolio construction and evaluation. It was critically assumed that the FTSE MIB represented a good approximation of the Italian stock market (and the S&P500 of the US one). If we accept this assumption, then results can be generalized to be applicable to the two stock markets. A first interesting topic to be further investigated is the consistency of results for different portfolios, for example, debt or derivatives. In fact, there is no reason to expect those results to be the same.

Another possible evolution of the analysis would work at the model level. As already pointed out VaR models are continually changing and growing compared to the three basic versions analyzed in this thesis. It would be highly interesting to analyze how those results change as models are slightly modified.

Finally, the perspective that has been endorsed here is a "supervisory perspective", meaning that the sole focus of attention was the underestimation of actual losses. Of course, other aspects deserve attention, such as the accuracy of the models, i.e., its ability to closely predict the exact amount of the loss.

Conclusions

With the last chapter of this work (5), the reader understands why a large body of this thesis was devoted to a detailed description of Montecarlo implementation. This approach, in fact, proves to be superior to the other two, at least for what concerns the studies that have been presented. Limitations of these studies have already been pointed out, and future research should follow this direction in order to assess if this result is robust also for other categories of assets, in different markets and under other assumptions (for example by considering accuracy measures rather than failure rates).

Despite those necessary and severe limitations, the results of the studies that have been performed are still of some utility. The Montecarlo method proves to be superior to the other two both in the Italian and in the US stock markets, from a supervisory perspective. It results, then, that the best performing model is the most complex one, at least under the point of view that has been considered.

Montecarlo simulation is, in fact, much more time consuming and computationally intensive, but this burden may come with the benefit of fewer underestimations of capital requirements.

Provided that the Montecarlo method has proven to be superior to the other two approaches, considerations and examples are made in relation to random number generation. The generation of random numbers is, in fact, the critical and distinctive feature of Montecarlo methods and it has to be acknowledged that even if the underlying VaR model is relatively sophisticated and well suited, the VaR estimate that results may be inaccurate because random number generator that is employed is of poor quality. Efficiency and period length considerations should be then a basis for the design of every Montecarlo VaR model.

Finally, the third empirical analysis presented in 5, has shown that distributional assumptions can have a severe impact on VaR estimates that result from the implementation of the Montecarlo method. The consequence of this is that a considerable amount of resources should be invested in the analysis of the suitability of a distributional assumption for each relevant instrument. Again, mistakes in these assumptions may lead the VaR estimates to be unreliable, even if based on good quality models.

Appendix A

Proofs

A.1 Proof of 3.1

Let X be a random variable with continuous distribution function F. Define now a new random variable Y as $F(X)$, i.e. $Y = F(X)$. It follows that:

$$F(Y \leq x) = F(F(X) \leq x) = F(X \leq F^{-1}(x)) = F^{-1}(F(x)) = x \tag{A.1}$$

Then the random variable Y must have a uniform distribution in the interval $(0,1)$, $Y = F(X) \sim U(0,1)$.

A.2 Proof of Acceptance-Rejection method

Define Z as the random variable generated by the implementation of the method. Note that the proposal random variable Y and the Uniform U are independent,then:

$$P(Z \leq z) = P(Y \geq x | U \leq \frac{f(Y)}{cg(Y)}) = \frac{\int_{-\infty}^{x} \int_{0}^{f(t)/cg(t)} g(t) ds dt}{\int_{-\infty}^{\infty} \int_{0}^{f(t)/cg(t)} g(t) ds dt} = \int_{-\infty}^{x} f(t) dt. \tag{A.2}$$

Then the claim follows.

A.3 Proof of Ratio of uniforms relation

Given the two random variables (U,V) and the area of definition c, their joint density, $f(u,v)$, is defined as $f(u,v) = I(u,v)/c$. Define two new random variables $X = U$ and $Y = V/U$. The joint density of these two new random variables $f(x,y)$ is simply given by:

$$f(x,y) = xI(x,y)/c = \frac{x}{c} I_{[0,\sqrt{h(y)}]}(x) \tag{A.3}$$

Then it follows that:

$$f(y) = f(v/u) = \int_{0}^{\sqrt{h(y)}} \frac{x}{c} dx = \frac{1}{2c} h(y) \tag{A.4}$$

This quantity is evidently proportional to h.

Appendix B

Codes

B.1 Chapter 1

B.1.1 Buffon needle experiment

```
%% Buffon needle experiment: estimation of pi
% Author: Martina Aquila
%% simulation
% set d=2 and L=1
clc
clear all

n=20000;
D=rand(1,n);% vector of n pseudorandom numbers in the range [0,1)
theta=rand(1,n)*pi/2;% vector of n pseudorandom numbers in the range [0,pi/2)
succ=D<=sin(theta)/2; % vector of 0 if false and 1 if true
sum(succ) %show the number of hits
pi_hat=n/sum(succ) %our estimated value of pi!
```

B.1.2 Histogram of Buffon needle experiment

```
%% histograms
% Author: Martina Aquila
n=100;
m=2000;
for i=1:m
D=rand(1,n)
theta=pi/2*rand(1,n)
succ=D<=sin(theta)/2
pi_hat(i)=n/sum(succ) %vector of estimated values of pi
end
%descriptive analysis of results
hist(pi_hat) %histogram of distribution
mean_pi_hat=n/sum(pi_hat) %sample mean
dev=(pi_hat-mean_pi_hat).*(pi_hat-mean_pi_hat) %squared deviations from the mean
var_pi_hat=sum(dev)/(n-1) %sample variance
```

```
sdev_pi_hat=sqrt(var_pi_hat) %sample standard deviation
```

B.1.3 Monte Carlo double

```
% Source: Stochastic Simulation and Applications in Finance with MATLAB Progra
By Huu Tue Huynh, Van Son Lai, Issouf Soumare
function result = MonteCarlo_double(f, g, x0, x1, y0, y1, n)
%
% Monte Carlo integration of f over a domain g>=0, embedded
% in a rectangle [x0,x1]x[y0,y1]. n^2 is the number of
% random points.

% Draw n^2 random points in the rectangle
x = x0 + (x1 - x0)*rand(n,1);
y = y0 + (y1 - y0)*rand(n,1);
% Compute sum of f values inside the integration domain
f_mean = 0;
num_inside = 0; % number of x,y points inside domain (g>=0)
for i = 1:length(x)
for j = 1:length(y)
if g(x(i), y(j)) >= 0
num_inside = num_inside + 1;
f_mean = f_mean + f(x(i), y(j));
end
end
end
f_mean = f_mean/num_inside;
area = num_inside/(n^2)*(x1 - x0)*(y1 - y0);
result = area*f_mean;
end
```

B.2 Chapter 2

B.2.1 LCG

```
%% LCG
% Author: Martina Aquila
clc
clear all
n=100
m=32;
a=5;
b=3;

x_t=11*ones(n);
```

```
for i=2:n
x_t(i)=mod(a*x_t(i-1)+b,m)
end

x_t=x_t(:,1)

u_t=x_t/m

hist(u_t)
```

B.2.2 LCG non full

```
%% LCG non full
% Author: Martina Aquila
clc
clear all

n=100
m=32;
a=6;
b=11;

x_t=10*ones(n);

for i=2:n
x_t(i)=mod(a*x_t(i-1)+b,m)
end

x_t=x_t(:,1)
```

B.2.3 LCG LGM

```
%% LCG LGM
% Author: Martina Aquila
clc
clear all

n=100
m=2147483647;
a=16807;
b=0;

x_t=ones(n);

for i=2:n
x_t(i)=mod(a*x_t(i-1),m)
```

```
15 end
```

B.2.4 LCG with histogram

```
1  %% LCG
   % Author: Martina Aquila
3  clc
   clear all
5  n=100
   m=32;
7  a=5;
   b=3;
9
   x_t=11*ones(n);
11
   for i=2:n
13 x_t(i)=mod(a*x_t(i-1)+b,m)
   end
15
   x_t=x_t(:,1)
17
   u_t=x_t/m
19
   hist(u_t)
```

B.2.5 LCG uniform

```
   %% LCG Uniform
2  % Author: Martina Aquila
   clc
4  clear all
   n=100
6  m=1;
   a=7;
8
   u_t=0.2*ones(n);
10
   for i=2:n
12 u_t(i)=mod(a*u_t(i-1),m)
   end
14
   u_t=u_t(:,1)
16
   hist(u_t)
```

B.2.6 MCG

```
%% MCG
% Author: Martina Aquila
clc
clear all

m=32;
a_1=5;
a_2=6

x_t=11*ones(m);

for i=3:m
x_t(i)=mod(a_1*x_t(i-1)+a_2*x_t(i-2),m)
end
%uniformly distributed random numbers
u_t=x_t/m
```

B.2.7 MCG Matrix

```
%% MCG matrix
% Author: Martina Aquila
clc
clear all

m=32;
A=[0,1,0,0
0,0,1,0
0,0,0,1
3,4,5,6];
det(A)% check that A is invertible

X_t=11*ones(m);

for i=2:m
X_t(i)=mod(X_t(i-1),m)
end
%uniformly distributed random numbers
u_t=x_t/m
```

B.2.8 W-H

```
%% WH
% Author: Martina Aquila
clc
```

```
clear all

m_1=30269;
a_1=171;
m_2=30307;
a_2=172;
m_3=30323;
a_3=170;

x_t=11*ones(m_1);
y_t=11*ones(m_2);
z_t=11*ones(m_3);



for i=2:m_1
x_t(i)=mod(a_1*x_t(i-1),m_1)
end

for i=2:m_2
y_t(i)=mod(a_2*y_t(i-1),m_2)
end

for i=2:m_3
z_t(i)=mod(a_3*z_t(i-1),m_3)
end

u_t=mod(x_t./m_1+y_t./m_2+z_t./m_3,1)
```

B.2.9 MRG32k3a

```
%% MRG32k3a.m
%Adapted from D.P.Kroese
m1=2^32-209; m2=2^32-22853;
ax2p=1403580; ax3n=810728;
ay1p=527612; ay3n=1370589;

x=[12345 12345 12345]; % Initial x at -1, -2, -3
y=[12345 12345 12345]; % Initial y at -1, -2, -3

n=100; % Compute the sequence for N steps
u=zeros(1,n);
for t=1:n
x_t=mod(ax2p*x(2)-ax3n*x(3),m1);
y_t=mod(ay1p*y(1)-ay3n*y(3),m2);
if x_t <= y_t
u(t)=(x_t - y_t + m1)/(m1+1);
```

```
else
u(t)=(x_t - y_t)/(m1+1);
end
x(2:3)=x(1:2); x(1)=x_t; y(2:3)=y(1:2); y(1)=y_t;
end
```

B.2.10 KISS99

```
% KISS99
%Adapted from D.P.Kroese
% Seeds: Correct variable types crucial

A=uint32(12345); B=uint32(65435); Y=12345; Z=uint32(34221);
n=100; % Compute the sequence for N steps
u=zeros(1,n);
for t=1:n
% Two Multiply with Carry Generators
A=36969*bitand(A,uint32(65535))+bitshift(A,-16);
B=18000*bitand(B,uint32(65535))+bitshift(B,-16);
% MWC: Low and High 16 bits are A and B
X=bitshift(A,16)+B;
% CONG: Linear Congruential Generator
y = mod(69069*y+1234567,4294967296);
% SHR3: 3-Shift Register Generator
z=bitxor(z,bitshift(z,17));
z=bitxor(z,bitshift(z,-13));
z=bitxor(z,bitshift(z,5));
% Combine them to form the KISS99 generator
KISS=mod(double(bitxor(x,uint32(y)))+double(z),4294967296);
u(t)=KISS/4294967296; % u[0,1] output
end
```

B.3 Chapter 3

B.3.1 Inverse transform-Continuous variable

```
%% Inverse transform-Continuous variable
% Author: Martina Aquila
clc
clear all

n=1000; %set number of repetitions

U=rand(n,1);%generate uniform

X=sqrt(U);%generate desired random variable
```

```
11
%plot histogram of generated distribution
13 hist(U)
hist(X)
```

B.3.2 Inverse transform-Beroulli

```
%% Inverse transform method Bernulli
2 % Author: Martina Aquila
clc
4 clear all

6 p=0.2%choose value for the parameter
n=100%set number of repetitions
8
u=rand(n,1);%generate uniform
10
%set up loop for generation of Bernulli
12
for i=1:n
14 if u(i)<p
u(i)=0
16 else u(i)=1
end
18 end
```

B.3.3 Inverse transform-Discrete 1

```
%% Inverse transform- Discrete 1
2 %Adapted from D.P.Kroese
clc
4 clear all

6 n=1000 %set number of repetitions
p=[0.1,0.2,0.3,0.4] %vector of probabilities
8
x=zeros(n,1) %preallocate the space
10
for i=1:n
12 x(i)=min(find(rand<cumsum(p)))
end
```

B.3.4 Inverse transform-Discrete 2

```
%% Inverse transform- Discrete 2
%Adapted from D.P. Kroese
clc
clear all

n=1000 %set number of repetitions
p=[0.1,0.2,0.3,0.4] %vector of probabilities

[dummy,x]=histc(rand(1,n),[0,cumsum(p)])
```

B.3.5 Alias method

```
%% Alias method A
%Adapted from Dirp K. Kroese

clc
clear all

p=rand(1,200);
p=p/sum(p);
n=size(p,2); %sample size
a=1:n;
q=zeros(1,n);%initialyze the space
q=n*p;
B=find(q>=1);
A=find(q<1);
while (~isempty(A) && ~isempty(B))
i = B(1);
j = A(1);
a(i) = j;
q(j) = q(j) -(1- q(i));
if (q(j) < 1)
A = setdiff(A,j);
B = union(B,j);
end
B = setdiff(B,i);
end
pp = q/n
for i = 1:n
ind = find(a == i);
pp(i) = pp(i) + sum((1 - q(ind)))/n;
end
max(abs(pp - p))
N = 10^6; % generate sample of  size N
X = zeros(1,N);
for i = 1:N
```

```
K = ceil(rand*n);
if (rand > q(K));
X(i) = a(K);
else
X(i) = K;
end
end
```

B.3.6 Acceptance-Rejection Positive Normal

```
%% Acceptance-Rejection Positive Normal Distribution
% Author: Martina Aquila
clc
clear all

n=100 %set number of repetitions
y=zeros(n,1) %preallocate the space
x=exprnd(1,n,1) % generate proposal
u=rand(n,1)% generate uniform

%implement AR

for i=1:n
if u(i)<=sqrt(2/pi)*exp(-x(i)^2/2)/sqrt(2*exp(1)/pi)
y(i)=x(i)
else y(i)=NaN
end
end

%% Efficiency
eff=sqrt(pi/2*exp(1))
```

B.3.7 Bernoulli

```
%% Bernulli
% Author: Martina Aquila
clc
clear all

n=100;
p=0.25;
u=rand(n,1);
x=zeros(n,1);

for i=1:n
if u(i)<=p
```

```
x(i)=1
else x(i)=0
end
end

bar(x)
```

B.3.8 Binomial via Normal

```
%% Binomial via Normal
% Author: Martina Aquila
clc
clear all

n=100;
p=0.1;
y=randn(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=max(0,floor(n*p+0.5+y(i)*sqrt(n*p*(1-p))))
end
```

B.3.9 Binomial Recursive 1

```
%% recursive binomial generator 1
%Adapted from D.P. Kroese
function x=binomialrnd(n,p)
% recursive binomial generator
if n<=10
x=sum(rand(1,n)<p);
else
k=ceil(n*p);Y=nbinrnd(k,p);% generate NegBin(k,p)
T=k+Y;
if T<=n
x=k+binomialrnd(n-T,p);
else
x=k-binomialrnd(T-n,p);
end
end
```

B.3.10 Binomial Recursive 2

```
%% recursive binomial generator 2
%Adapted from D.P. Kroese
```

```
3
function x=binomrnd_beta(n,p)
5
if n<=10
7 x=sum(rand(1,n)<p);
else
9 k=ceil(n*p);Uk=betarnd(k,n+1-k);% generate beta r.v.
if Uk<p
11 x=k+binomrnd_beta(n-k,(p-Uk)/(1-Uk));
else
13 x=k-binomrnd_beta(k-1,(Uk-p)/Uk);
end
15 end
```

B.3.11 Geometric via Exponential

```
%% Geometric 1
2 % Author: Martina Aquila
clc
4 clear all

6 n=100;
p=0.2;
8 y=exprand(-ln(1-p),1,n)';
x=zeros(1,n)'
10 ;
for i=1:n
12 x(i)=ceil(y(i))
end
```

B.3.12 Geometric via Uniform

```
%% Geometric 2
2 % Author: Martina Aquila
clc
4 clear all
n=100;
6 p=0.2;
u=rand(1,n)';
8 x=zeros(1,n)';
for i=1:n
10 x(i)=ceil(ln(u(i)/ln(1-p))
end
```

B.3.13 Hypergeometric

```
%% Hypergeometric
%Adapted from D.P. Kroese
N = 100; %total number of balls
n = 20; % take n balls
r = 30; % number of red balls
w = zeros(1,N);
w(1:r) = 1;
K = 10^5;
x = zeros(1,K);
for i=1:K
[s,ix] = sort(rand(1,N));
x(i) = sum(w(ix(1:n)));
end

xx = [0:n];
count = hist(x,xx);
ex = hygepdf(xx,N,r,n)*K;
clf
hold on
plot(xx,count,'.r')
plot(xx,ex,'ob')
hold off
```

B.3.14 Negative Binomial

```
%% Negative Binomial
% Author: Martina Aquila

clc
clear all
n=10;
r=10;
p=0.2;
u=rand(1,n);
y=zeros(1,n);
c=ln(1-p);

for i=1:n
y(i)=floor(ln(u(i)/c)
end

x=cumsum(y)
```

B.3.15 Poisson

```
%% Poisson
% Author: Martina Aquila
clc
clear all

n=100;
l=0.2;
m=floor(7/8*l);
y=gamrnd(m,1,1,n)';
z=zeros(1,n)';
x=zeros(1,n)';

for i=1:n
if y(i)<=l
z(i)=poissrnd(l-y(i))
x(i)=m+z(i)
else x(i)=binornd(m-1,l/y(i))
end
end
```

B.3.16 Beta(a,1)

```
%% Beta (alpha,1)
% Author: Martina Aquila
clc
clear all

n=100;
a=0.8;

u=rand(1,n)';
x=zeros(1,n)';
for i=1:n
x(i)=u(i)^(1/a)
end
```

B.3.17 Beta(1,b)

```
%% Beta (1,beta)
% Author: Martina Aquila
clc
clear all

n=100;
b=0.8;
```

```
u=rand(1,n)';
x=zeros(1,n)';
for i=1:n
x(i)=1-u(i)^(1/b)
end
```

B.3.18 Beta(0.5,0.5)

```
%% Beta (0.5,0.5)
% Author: Martina Aquila
clc
clear all

n=100;

u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=cos(pi*u(i)/2)^2
end
```

B.3.19 Beta(a,a) 1

```
%% Beta (alpha,alpha) 1
% Author: Martina Aquila
clc
clear all

n=100;
a=0.7;

u_1=rand(1,n)';
u_2=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=0.5*(1+sqrt(1-u_1(i)^(2/(2*a-1)))*cos(2*pi*u_2(i)))
end
```

B.3.20 Beta(a,a) 2

```
%% Beta (alpha,alpha) 2
% Author: Martina Aquila
```

```
clc
clear all

n=100;
a=0.7;

u=rand(1,n)';
v=rand(1,n)'*2-1;
x=zeros(1,n)';

s=u.^2+v.^2

for i=1:n
if s(i)>1
x(i)=NaN
else x(i)=0.5+u(i)*v(i)/s(i)*sqrt(1-s(i)^(2/(2*a-1)))
end
end
```

B.3.21 Beta Johnk algorithm

```
%% Beta Johnk algorithm
% Author: Martina Aquila
clc
clear all

n=100;
a=0.8;
b=0.6;

u_1=rand(1,n)';
u_2=rand(1,n)';
v_1=ones(1,n)';
v_2=ones(1,n)';

for i=1:n
v_1(i)=u_1(i)^(1/a)
end

for i=1:n
v_2(i)=u_2(i)^(1/b)
end

w=v_1+v_2
x=zeros(1,n)';

for i=1:n
```

```
if w(i)>1
x(i)=NaN
else
x(i)=v_1(i)/w(i)
end
end
```

B.3.22 Cauchy(0,1)

```
%% Cauchy(0,1)
% Author: Martina Aquila
clc
clear all

n=100;
m=0;
s=1;

u=rand(1,n);
c=zeros(1,n);
for i=1:n
c(i)=tan(pi*u(i))
end
```

B.3.23 Cauchy(0,1) Ratio of Uniforms

```
%% Cauchy(0,1) Ratio of Uniform
% Author: Martina Aquila
clc
clear all

n=100;
u=rand(1,n)';
v=rand(1,n)';
v=v-0.5;
x=zeros(1,n)';

for i=1:n
if u(i)^2+v(i)^2<=1
x(i)=v(i)/u(i)
else x(i)=NaN
end
end
```

B.3.24 Cauchy(0,1) Ratio of Normals

```
%% Cauchy(0,1) Ratio of Normal
% Author: Martina Aquila
clc
clear all

n=100;
y=rand(1,n)';
v=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=y(i)/v(i)
end
```

B.3.25 Exponential

```
%% Exponential
% Author: Martina Aquila
clc
clear all

l=1.2;
n=100;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=-log(u(i))/l
end
```

B.3.26 F 1

```
%% F 1
% Author: Martina Aquila
clc
clear all

m=5;
n=6;
k=100;
x=chi2rnd (m,k,1);
y=chi2rnd (n,k,1);
z=zeros(1,k);
```

```
for i=1:k
z(i)=(x(i)/m)/(y(i)/n)
end
```

B.3.27 F 2

```
%% F 2
clc
clear all

m=5;
n=6;
k=100;

b=betarnd (m/2,n/2,k,1)';
x=zeros(1,k)';

for i=1:k
x(i)=(b(i)*n)/(m*(1-b(i)))
end
```

B.3.28 Frechet

```
%% Frechet(alpha,0,1)
% Author: Martina Aquila
clc
clear all

n=100;
a=3;

u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=(-log(u(i)))^(-1/a)
end
```

B.3.29 Gamma Best

```
%% Gamma best
% adapted from D.P. Kroese
N = 10^5; alpha = 0.3;
d= 0.07 + 0.75*sqrt(1-alpha); b = 1 + exp(-d)*alpha/d;
x = zeros(N,1);
```

```
for i = 1:N
cont = true;
while cont
U1 = rand;
U2 = rand;
V = b*U1;
if V <= 1
X = d*V^(1/alpha);
if U2 <= (2-X)/(2+X)
cont = false; break;
else
if U2 <= exp(-X)
cont = false; break;
end
end
else
X = -log(d*(b-V)/alpha);
y = X/d;
if U2*(alpha + y*(1-alpha)) < 1
cont= false; break;
else
if U2 <= y^(alpha - 1)
cont= false;break;
end
end
end
end
x(i) = X;
end

clf
hold on
x = sort(x);
ecdf(x); % empirical cdf
y = gamcdf(x,alpha);  % cdf of gamma distribution
plot(x,y,'r')
hold off
```

B.3.30 Gamma Cheng Feast

```
%% Gamma Cheng Feast algorithm
% Author: Martina Aquila
clc
clear all

a=1.2;
n=100;
```

```
u_1=rand(1,n)';
u_2=rand(1,n)';

for i=1:n
v(i)=((a-(6*a)^(-1))*u_1(i))/((a-1)*u_2(i))
end

x=ones(1,n)';

for i=1:n
if 2*(u_2(i)-1)/(a-1)+v(i)+1/v(i)<=2
x(i)=(a-1)*v(i)
if 2*log(u_2(i))/(a-1)-log(v(i))+v(i)<=1
x(i)=(a-1)*v(i)
else x(i)=NaN
end
end
end
```

B.3.31 Gamma Chi-square

```
%% Gamma(1/2,1/2) (chi-square)
% Author: Martina Aquila
clc
clear all

n=100;
z=randn(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=z(i)^2
end
```

B.3.32 Gumbel

```
%% Gumbel
% Author: Martina Aquila
clc
clear all

n=100;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
```

```
x(i)=-log(-log(u(i)))
end
```

B.3.33 Laplace 1

```
%% Laplace 1
% Author: Martina Aquila
clc
clear all

n=100;
y=exprnd(1,1,n)';
x=zeros(1,n)';

p=0.5;
u=rand(1,n)';
b=zeros(1,n)';

for i=1:n
if u(i)<=p
b(i)=1
else b(i)=0
end
end
for i=1:n
x(i)=(2*b(i)-1)*y(i)
end
```

B.3.34 Laplace 2

```
%% Laplace 2
% Author: Martina Aquila
clc
clear all

n=100;
u=rand(1,n)'-0.5;
x=zeros(1,n)';

for i=1:n
x(i)=sign(u(i))*log(1-2*abs(u(i)))
end
```

B.3.35 Laplace 3

```
%% Laplace 3
% Author: Martina Aquila
clc
clear all

n=100;
v=exprnd(1,1,n)';
w=exprnd(1,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=v(i)-w(i)
end
```

B.3.36 Laplace 4

```
%% Laplace 4
% Author: Martina Aquila
clc
clear all

n=100;
e=exprnd(1,1,n)';
y=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=y(i)*sqrt(2*e(i))
end
```

B.3.37 Logistic

```
%% Logistic
% Author: Martina Aquila
clc
clear all

n=100;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=log(u(i)/(1-u(i)))
end
```

B.3.38 Log-Normal

```
%% Log-Normal
% Author: Martina Aquila
clc
clear all

n=100;
m=0;
s2=1;

y=randn(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=exp(y(i))
end
```

B.3.39 Normal Box-Muller

```
%% Normal Box-Muller
% Author: Martina Aquila
clc
clear all

n=100;
u_1=rand(1,n)';
u_2=rand(1,n)';
x_1=zeros(1,n)';
x_2=zeros(1,n)';

for i=1:n
x_1(i)=sqrt(-2*log(u_1(i)))*cos(2*pi*u_2(i))
x_2(i)=sqrt(-2*log(u_1(i)))*sin(2*pi*u_2(i))
end
plot(x_1)
```

B.3.40 Normal Rejection Polar method

```
%% Normal Rejection Polar method
% Author: Martina Aquila
clc
clear all

n=100;
v_1=rand(1,n)'*2+1;
```

```
v_2=rand(1,n)'*2+1;
r=v_1.^2+v_2.^2;
x_1=zeros(1,n)';
x_2=zeros(1,n)';

for i=1:n
if r(i)>=1
x_1(i)=NaN
x_2(i)=NaN
else
x_1(i)=v_1(i)*sqrt(-2*log(r(i)^2)/r(i)^2)
x_2(i)=v_2(i)*sqrt(-2*log(r(i)^2)/r(i)^2)
end
end
```

B.3.41 Pareto 1

```
%% Pareto 1
% Author: Martina Aquila
clc
clear all

n=100;
a=2;

u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=u(i)^(-1/a)-1
end
```

B.3.42 Pareto 2

```
%% Pareto 2
% Author: Martina Aquila
clc
clear all

n=100;
a=2;
y=exprnd(1,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=exp(y(i)/a)- 1
```

```
end
```

B.3.43 Student-t via Chi Square and Standard Normal

```
%% Student-t  Chi-square-NormaleSt
% Author: Martina Aquila
clc
clear all

n=100;
v=5;

z=randn(1,n)';
y=gamrnd(v/2,0.5,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=z(i)/sqrt(y(i)/v)
end
```

B.3.44 Student-t via Beta

```
%% Student-t  Beta
% Author: Martina Aquila
clc
clear all

n=100;
v=5;

y=betarnd(v/2,v/2,1,n)';
x=zeros(1,n)';

for i=1:n
x(i)= sqrt(v)*(y(i)-0.5)/(sqrt(y(i)*(1-y(i))))
end
```

B.3.45 Student-t via Ratio of Uniform

```
%% Student-t  Ratio of uniform
% Author: Martina Aquila
clc
clear all
n=100;
v=5;
```

```
z=rand(1,n)';
u=rand(1,n)'*2*sqrt(v)+sqrt(v);
x=zeros(1,n)';
w=z.^(1/v);

for i=1:n
if w(i)^2+u(i)^2/v<=1
x(i)=u(i)/w(i)
else x(i)=NaN
end
end
```

B.3.46 Student-t via Polar 1

```
%% Student-t  Polar 1
% Author: Martina Aquila
clc
clear all

n=100;
w=5;
u=rand(1,n)';
v=rand(1,n)';

t=u.*2*pi;
r=sqrt(w*(v.^(-2/w)-1));
x=r.*cos(t);
y=r.*sin(t);
```

B.3.47 Student-t via Polar 2

```
%% Student-t  Polar 2
% Author: Martina Aquila
clc
clear all

n=100;
q=5;
x=zeros(1,n)';
u=rand(1,n)'*2+1;
v=rand(1,n)'*2+1;
w=u.^2+v.^2;

for i=1:n
if w(i)>1
```

```
x(i)=NaN
else x(i)=sgn(u(i))*sqrt((u(i)^2/w(i))*q*(w(i)^(-2/q)-1))
end
end
```

B.3.48 Student-t via Polar Bailey

```
%% Student-t Rejection Polar method
% Author: Martina Aquila
clc
clear all

n=100;
ni=4;
v_1=rand(1,n)'*2+1;
v_2=rand(1,n)'*2+1;
r=v_1.^2+v_2.^2;
x=zeros(1,n)';

for i=1:n
if r>=1
x(i)=NaN
else x(i)=v_1(i)*sqrt((ni*(r(i)^(-8/ni)-1)/r(i)))
end
end
```

B.3.49 Uniform(a,b)

```
%% Uniform(a,b)
% Author: Martina Aquila
clc
clear all

n=100;
a=2;
b=3;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=a+(b-a)*u(i)
end
```

B.3.50 Wald

```
%% Inverse Gaussian MSGH method
% Author: Martina Aquila
clc
clear all

m=4;
l=2;
n=100;
z=rand(1,n)';
y=z.^2;
x_1=m+(m^2*y)/(2*l)-m/(2*l)*sqrt(4*m*l*y+m^2*y.^2);
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
if u(i)<=m/(m+x_1(i))
x(i)=x_1(i)
else x(i)=m^2/(x_1(i))
end
end
```

B.3.51 Weibull

```
%% Weibull
% Author: Martina Aquila
clc
clear all

n=100;
a=1.2;
b=1;
u=rand(1,n)';
x=zeros(1,n)';

for i=1:n
x(i)=(-log(u(i)))^(1/a)
end
```

B.3.52 Raileigh

```
%% Rayleigh Inverse CDF
% Author: Martina Aquila
clc
clear all
```

```
n=100;
b=1;
a=2;
s=sqrt(2/b);
u=rand(1,n)';
r=zeros(1,n)';

for i=1:n
r(i)=s*sqrt(-log(u(i)))
end
```

B.4 Chapter 4

B.4.1 Parametric VaR for Portfolio

```
%% VaR Forward Var/Cov
% Author: Martina Aquila
r_usa_0=0.023
r_uk_0=0.009
S_0=1.29
Q_us=6
Q_uk=10
T=0.25
P_0=S_0*(Q_uk)/(1+r_uk_0)^T-Q_us/(1+r_usa_0)^T

t1=datetime(2010,1,15)
t = t1 + calmonths(1:108)
%% Exchange rate analysis

S=GBPUSDmontly(3:end)
m_S=mean(S)
s_S=std(S)

plot(t,S)
hist(S)
%% US t-bills

r_usa=USDBillsmonthly(3:end)*0.01

m_usa=mean(r_usa)
s_usa=std(r_usa)*12^(-1/2)
plot(t,r_usa)
hist(r_usa)
%% GBP Bills

r_uk=Ukbillsmonthly(3:end)*0.01
```

```

m_uk=mean(r_uk)
s_uk=std(r_uk)
plot(t,r_uk)
hist(r_uk)
%% Portfolio

covS_uk=corrcoef(S,r_uk)
corS_uk=covS_uk(1,2)
covS_us=corrcoef(S,r_usa)
corS_usa=covS_us(1,2)
covus_uk=corrcoef(r_usa,r_uk)
coruk_usa=covus_uk(1,2)

m_port=Q_us*m_usa+Q_uk*m_uk*m_S
s_port=sqrt(Q_us^2*s_usa^2+Q_uk^2*s_uk^2+s_S^2+2*Q_us*Q_uk*coruk_usa*s_usa*s_uk+2*

%alpha=5%

VaR=m_port-1.65*s_port
```

B.4.2 Historical simulation VaR for Portfolio

```
%% VaR Historical simulation
% Author: Martina Aquila
n=100 %target number of data points
a=0.05 %confidence level
i_usa_0=0.023
i_uk_0=0.009
S_0=1.29
Q_us=6
Q_uk=10
T=0.25
P_0=S_0*(Q_uk)/(1+i_uk_0)^T-Q_us/(1+i_usa_0)^T

t1=datetime(2010,10,15)
t = t1 + calmonths(1:100)

%% Exchange rate USD vs GBP

S=GBPUSDmontly(10:end)'
r_S=zeros(1,n)' %percentage changes

for i=1:n
r_S(i)=S(i+1)/S(i)-1
end

```

```
S_1=S_0*(1+r_S(1))
S_s=ones(1,n)'*S_1

for i=2:n
S_s(i)= S_s(i-1)*(1+r_S(i))
end

hist(S_s)
figure;
plot( S(2:end), "r");
hold on;
plot(S_s, "g");
hold off;

%% US t-bills

i_usa=USDBillsmonthly(10:end)*0.01

r_usa=zeros(1,n)' %percentage changes

for i=1:n
r_usa(i)=i_usa(i+1)/i_usa(i)-1
end

i_usa_1=i_usa_0*(1+r_usa(1))
i_usa_s=ones(1,n)'*i_usa_1

for i=2:n
i_usa_s(i)= i_usa_s(i-1)*(1+r_usa(i))
end

figure;
plot( i_usa(2:end), "r");
hold on;
plot(i_usa_s, "g");
hold off;

hist(i_usa_s)

%% GBP Bills

i_uk=Ukbillsmonthly(10:end)*0.01

r_uk=zeros(1,n)' %percentage changes

for i=1:n
r_uk(i)=i_uk(i+1)/i_uk(i)-1
end
```

```
i_uk_1=i_uk_0*(1+r_uk(1))
i_uk_s=ones(1,n)'*i_uk_1

for i=2:n
i_uk_s(i)= i_uk_s(i-1)*(1+r_uk(i))
end

figure;
plot( i_uk(2:end), "r");
hold on;
plot(i_uk_s, "g");
hold off;
hist(i_uk_s)

%% Porfolio

P_0=(Q_uk)/(1+i_uk_0)^T-Q_us/(1+i_usa_0)^T*S_0

P=ones(1,n)'*P_0
for i=2:n
P(i)=(Q_uk)/(1+i_uk_s(i))^T-Q_us/(1+i_usa_s(i))^T*S_s(i)
end
plot(P)
Pl=zeros(1,n)

for i=2:n
Pl(i)=P(i)-P(i-1)
end
Pl_ord=sort(Pl)
hist(Pl_ord)
VaR= Pl_ord(5)
```

B.4.3 Montecarlo simulation VaR for Portfolio

```
%% VaR Montecarlo simulation

n=1000 %target number of data points
a=0.05 %confidence level
USA_0=0.023 %initial value
UK_0=0.009 %initial value
S_0=1.29 %initial value
Q_us=6
Q_uk=10
T=0.25
%initial portfolio value
P_0=S_0*(Q_uk)/(1+UK_0)^T-Q_us/(1+USA_0)^T
```

```
%generate dates for plot
t1=datetime(2003,1,15)
t = t1 + calmonths(1:200)

%% Exchange rate USD vs GBP

S=USDGBP(170:end)
%S_daily=DailyUSDGBP(2:end)

%first look at the distribution

hist(S,5)


%% choose correct distributional assumption

histfit(S) %look at the distribution, normal does not fit well
histfit(S,7,'beta')%no, data is in 0,1
histfit(S,7,'burr')%no
histfit(S,7, 'birnbaumsaunders')%yes
histfit(S,7,'exponential')%no
histfit(S,7,'ev')%not bad
histfit(S,10,'gamma')%not really
histfit(S,7,'gev')%not really
histfit(S,10,'gp')%no
histfit(S,7,'inversegaussian')%not bad
histfit(S,7,'logistic')% not so bad
histfit(S,10,'loglogistic')%no
histfit(S,10,'lognormal')%no
histfit(S,10,'nakagami')%no
histfit(S,10,'nbin')%no, must be integer
histfit(S,10,'normal')
histfit(S,10,'poisson')%no
histfit(S,10,'rayleigh')%no
histfit(S,10,'rician')%no
histfit(S,10,'tlocationscale')%no, normal is better
histfit(S,7,'wbl')%not bad

% chosen distribution: logistic
% generation of random paths for the factor
pd_S=fitdist(S,'logistic')
S_s=random(pd_S,n,1)
hist(S_s)

%% US t-bills

US=USD20032019(130:end)*0.01
```

```
%choose correct distributional assumption

histfit(US)
histfit(US,10,'beta')%no
histfit(US,10,'burr')%no
histfit(US,10, 'birnbaumsaunders')%show
histfit(US,10,'exponential')%not bad
histfit(US,10,'ev')%no
histfit(US,10,'gamma')%no
histfit(US,10,'gev')%no
histfit(US,10,'gp')%no
histfit(US,10,'inversegaussian')%no
histfit(US,10,'logistic')%no
histfit(US,10,'loglogistic')%no
histfit(US,10,'lognormal')%no
histfit(US,10,'nakagami')%no
histfit(US,10,'nbin')%no
histfit(US,10,'normal')%no, show
histfit(US,10,'poisson')%no
histfit(US,10,'rayleigh')%no, show
histfit(US,10,'rician')%no
histfit(US,10,'tlocationscale')%no
histfit(US,10,'wbl')%no

% chosen distribution: exponential
% generation of random paths for the factor
pd_US=fitdist(US,'exponential')
US_s= random(pd_US,n,1)
hist(US_s)


%% GBP Bills

UK=UK20032019(120:end)*0.01

%choose correct distributional assumption
histfit(UK) %look at the distribution, normal does not fit well
histfit(UK,10,'beta')%no
histfit(UK,10,'burr')%no
histfit(UK,10, 'birnbaumsaunders')%no
histfit(UK,10,'exponential')%no
histfit(UK,10,'ev')%no
histfit(UK,10,'gamma')%quite good
histfit(UK,10,'gev')%no
histfit(UK,10,'gp')%no
histfit(UK,10,'inversegaussian')%no
histfit(UK,10,'logistic')%no
```

```
histfit(UK,10,'loglogistic')%no
histfit(UK,10,'lognormal')%no
histfit(UK,10,'nakagami')%no
histfit(UK,10,'nbin')%no, must be integer
histfit(UK,10,'normal')%not so bad, but no
histfit(UK,10,'poisson')%no
histfit(UK,10,'rayleigh')% not really
histfit(UK,10,'rician')% not bad
histfit(UK,10,'tlocationscale')%no
histfit(UK,10,'wbl')%no

% chosen distribution: rician
% generation of random paths for the factor
pd_UK=fitdist(UK,'rician')

UK_s= random(pd_UK,1,n)
hist(UK_s)


%% Porfolio

%initial value
P_0=(Q_uk)/(1+UK_0)^T-Q_us/(1+USA_0)^T*S_0

P=ones(1,n)'*P_0%preallocate space

%compute simulated portfolio value
for i=2:n
P(i)=(Q_uk)/(1+UK_s(i))^T-Q_us/(1+US_s(i))^T*S_s(i)
end

plot(P)
Pl=zeros(1,n)%preallocate space

%compute profits and losses
for i=2:n
Pl(i)=P(i)-P_0
end

%compute VaR
Pl_ord=sort(Pl)'
hist(Pl_ord)
VaR= Pl_ord(50)
```

B.5 Chapter 5

B.5.1 FTSE MIB Study

```
%% VaR FTSE MIB  Variance Covariance

S=FTSEMIB(2:end)
S(242,:)=[]
S_0=FTSEMIB(151)


%% Initialization

B=FTSEMIB(152:300)% backtesting dataset
a=0.05


%% VaR
VaR_1=mean(S(1:25))/250-1.65*std(S(1:25))*sqrt(1/250)
VaR_2=mean(S(2:26))/250-1.65*std(S(2:26))*sqrt(1/250)
VaR_3=mean(S(3:27))/250-1.65*std(S(3:27))*sqrt(1/250)
VaR_4=mean(S(4:28))/250-1.65*std(S(4:28))*sqrt(1/250)
VaR_5=mean(S(5:29))/250-1.65*std(S(5:29))*sqrt(1/250)
VaR_6=mean(S(6:30))/250-1.65*std(S(6:30))*sqrt(1/250)
VaR_7=mean(S(7:31))/250-1.65*std(S(7:31))*sqrt(1/250)
VaR_8=mean(S(8:32))/250-1.65*std(S(8:32))*sqrt(1/250)
VaR_9=mean(S(9:33))/250-1.65*std(S(9:33))*sqrt(1/250)
VaR_10=mean(S(10:34))/250-1.65*std(S(10:34))*sqrt(1/250)
VaR_11=mean(S(11:35))/250-1.65*std(S(11:35))*sqrt(1/250)
VaR_12=mean(S(12:36))/250-1.65*std(S(12:36))*sqrt(1/250)
VaR_13=mean(S(13:37))/250-1.65*std(S(13:37))*sqrt(1/250)
VaR_14=mean(S(14:38))/250-1.65*std(S(14:38))*sqrt(1/250)
VaR_15=mean(S(15:39))/250-1.65*std(S(15:39))*sqrt(1/250)
VaR_16=mean(S(16:40))/250-1.65*std(S(16:40))*sqrt(1/250)
VaR_17=mean(S(17:41))/250-1.65*std(S(17:41))*sqrt(1/250)
VaR_18=mean(S(18:42))/250-1.65*std(S(18:42))*sqrt(1/250)
VaR_19=mean(S(19:43))/250-1.65*std(S(19:43))*sqrt(1/250)
VaR_20=mean(S(20:44))/250-1.65*std(S(20:44))*sqrt(1/250)
VaR_21=mean(S(21:45))/250-1.65*std(S(21:45))*sqrt(1/250)
VaR_22=mean(S(22:46))/250-1.65*std(S(22:46))*sqrt(1/250)
VaR_23=mean(S(23:47))/250-1.65*std(S(23:47))*sqrt(1/250)
VaR_24=mean(S(24:48))/250-1.65*std(S(24:48))*sqrt(1/250)
VaR_25=mean(S(25:49))/250-1.65*std(S(25:49))*sqrt(1/250)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,VaR_1


```

```
%% Backtesting

PL_B=B(65:89)-S_0


figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Acceptability
%not acceptable
```

```
%% VaR FTSE MIB Historical simulation


FM=FTSEMIB(2:end)
FM(242,:)=[]
B=FTSEMIB(65:89)% backtesting dataset
S_0=FTSEMIB(151)

%% Initialization
%5-day rolling window
S_1=FM(1:100)%first dataset
S_2=FM(5:104)%first dataset
S_3=FM(10:109)%first dataset
S_4=FM(15:114)%first dataset
S_5=FM(20:119)%first dataset
S_6=FM(25:124)%first dataset
S_7=FM(30:129)%first dataset
S_8=FM(35:134)%first dataset
S_9=FM(40:139)%first dataset
S_10=FM(45:144)%first dataset
S_11=FM(50:149)%first dataset
S_12=FM(55:154)%first dataset
S_13=FM(60:159)%first dataset
S_14=FM(65:164)%first dataset
S_15=FM(70:169)%first dataset
S_16=FM(75:174)%first dataset
S_17=FM(80:179)%first dataset
S_18=FM(85:184)%first dataset
S_19=FM(90:189)%first dataset
S_20=FM(95:194)%first dataset
S_21=FM(100:199)%first dataset
S_22=FM(105:204)%first dataset
S_23=FM(110:209)%first dataset
S_24=FM(115:214)%first dataset
```

```
35 S_25=FM(120:219)%first dataset

37

   S_0=FTSEMIB(151)
39 n=100%100
   k=25%149 %target number of VaR estimations=lenght backtesting dataset
41 a=0.05

43

   %% Historical simulation
45

   r_S_1=zeros(1,n)' %percentage changes
47 r_S_2=zeros(1,n)'
   r_S_3=zeros(1,n)'
49 r_S_4=zeros(1,n)'
   r_S_5=zeros(1,n)'
51 r_S_6=zeros(1,n)'
   r_S_7=zeros(1,n)'
53 r_S_8=zeros(1,n)'
   r_S_9=zeros(1,n)'
55 r_S_10=zeros(1,n)'
   r_S_11=zeros(1,n)'
57 r_S_12=zeros(1,n)'
   r_S_13=zeros(1,n)'
59 r_S_14=zeros(1,n)'
   r_S_15=zeros(1,n)'
61 r_S_16=zeros(1,n)'
   r_S_17=zeros(1,n)'
63 r_S_18=zeros(1,n)'
   r_S_19=zeros(1,n)'
65 r_S_20=zeros(1,n)'
   r_S_21=zeros(1,n)'
67 r_S_22=zeros(1,n)'
   r_S_23=zeros(1,n)'
69 r_S_24=zeros(1,n)'
   r_S_25=zeros(1,n)'
71

   for i=1:n-1
73 r_S_1(i)=S_1(i+1)/S_1(i)-1
   end
75 for i=1:n-1
   r_S_2(i)=S_2(i+1)/S_2(i)-1
77 end
   for i=1:n-1
79 r_S_3(i)=S_3(i+1)/S_3(i)-1
   end
81 for i=1:n-1
   r_S_4(i)=S_4(i+1)/S_4(i)-1
```

```
83 end
   for i=1:n-1
85 r_S_5(i)=S_5(i+1)/S_5(i)-1
   end
87 for i=1:n-1
   r_S_6(i)=S_6(i+1)/S_6(i)-1
89 end
   for i=1:n-1
91 r_S_7(i)=S_7(i+1)/S_7(i)-1
   end
93 for i=1:n-1
   r_S_8(i)=S_8(i+1)/S_8(i)-1
95 end
   for i=1:n-1
97 r_S_9(i)=S_9(i+1)/S_9(i)-1
   end
99 for i=1:n-1
   r_S_10(i)=S_10(i+1)/S_10(i)-1
101 end
   for i=1:n-1
103 r_S_11(i)=S_11(i+1)/S_11(i)-1
   end
105 for i=1:n-1
   r_S_12(i)=S_12(i+1)/S_12(i)-1
107 end
   for i=1:n-1
109 r_S_13(i)=S_13(i+1)/S_13(i)-1
   end
111 for i=1:n-1
   r_S_14(i)=S_14(i+1)/S_14(i)-1
113 end
   for i=1:n-1
115 r_S_15(i)=S_15(i+1)/S_15(i)-1
   end
117 for i=1:n-1
   r_S_16(i)=S_16(i+1)/S_16(i)-1
119 end
   for i=1:n-1
121 r_S_17(i)=S_17(i+1)/S_17(i)-1
   end
123 for i=1:n-1
   r_S_18(i)=S_18(i+1)/S_18(i)-1
125 end
   for i=1:n-1
127 r_S_19(i)=S_19(i+1)/S_19(i)-1
   end
129 for i=1:n-1
   r_S_20(i)=S_20(i+1)/S_20(i)-1
```

```
end
for i=1:n-1
r_S_21(i)=S_21(i+1)/S_21(i)-1
end
for i=1:n-1
r_S_22(i)=S_22(i+1)/S_22(i)-1
end
for i=1:n-1
r_S_23(i)=S_23(i+1)/S_23(i)-1
end
for i=1:n-1
r_S_24(i)=S_24(i+1)/S_24(i)-1
end
for i=1:n-1
r_S_25(i)=S_25(i+1)/S_25(i)-1
end


S_s_1=ones(1,n)'*S_1(1)
S_s_2=ones(1,n)'*S_2(1)
S_s_3=ones(1,n)'*S_3(1)
S_s_4=ones(1,n)'*S_4(1)
S_s_5=ones(1,n)'*S_5(1)
S_s_6=ones(1,n)'*S_6(1)
S_s_7=ones(1,n)'*S_7(1)
S_s_8=ones(1,n)'*S_8(1)
S_s_9=ones(1,n)'*S_9(1)
S_s_10=ones(1,n)'*S_10(1)
S_s_11=ones(1,n)'*S_11(1)
S_s_12=ones(1,n)'*S_12(1)
S_s_13=ones(1,n)'*S_13(1)
S_s_14=ones(1,n)'*S_14(1)
S_s_15=ones(1,n)'*S_15(1)
S_s_16=ones(1,n)'*S_16(1)
S_s_17=ones(1,n)'*S_17(1)
S_s_18=ones(1,n)'*S_18(1)
S_s_19=ones(1,n)'*S_19(1)
S_s_20=ones(1,n)'*S_20(1)
S_s_21=ones(1,n)'*S_21(1)
S_s_22=ones(1,n)'*S_22(1)
S_s_23=ones(1,n)'*S_23(1)
S_s_24=ones(1,n)'*S_24(1)
S_s_25=ones(1,n)'*S_25(1)


for i=2:n
S_s_1(i)= S_s_1(i-1)*(1+r_S_1(i))
end
```

```
179
    for i=2:n
181 S_s_1(i)= S_s_1(i-1)*(1+r_S_1(i))
    end
183 for i=2:n
    S_s_2(i)= S_s_2(i-1)*(1+r_S_2(i))
185 end
    for i=2:n
187 S_s_3(i)= S_s_3(i-1)*(1+r_S_3(i))
    end
189 for i=2:n
    S_s_4(i)= S_s_4(i-1)*(1+r_S_4(i))
191 end
    for i=2:n
193 S_s_5(i)= S_s_5(i-1)*(1+r_S_5(i))
    end
195 for i=2:n
    S_s_6(i)= S_s_6(i-1)*(1+r_S_6(i))
197 end
    for i=2:n
199 S_s_7(i)= S_s_7(i-1)*(1+r_S_7(i))
    end
201 for i=2:n
    S_s_8(i)= S_s_8(i-1)*(1+r_S_8(i))
203 end
    for i=2:n
205 S_s_9(i)= S_s_9(i-1)*(1+r_S_9(i))
    end
207 for i=2:n
    S_s_10(i)= S_s_10(i-1)*(1+r_S_10(i))
209 end
    for i=2:n
211 S_s_11(i)= S_s_11(i-1)*(1+r_S_11(i))
    end
213 for i=2:n
    S_s_12(i)= S_s_12(i-1)*(1+r_S_12(i))
215 end
    for i=2:n
217 S_s_13(i)= S_s_13(i-1)*(1+r_S_13(i))
    end
219 for i=2:n
    S_s_14(i)= S_s_14(i-1)*(1+r_S_14(i))
221 end
    for i=2:n
223 S_s_15(i)= S_s_15(i-1)*(1+r_S_15(i))
    end
225 for i=2:n
    S_s_16(i)= S_s_16(i-1)*(1+r_S_16(i))
```

```
227 end
    for i=2:n
229 S_s_17(i)= S_s_17(i-1)*(1+r_S_17(i))
    end
231 for i=2:n
    S_s_18(i)= S_s_18(i-1)*(1+r_S_18(i))
233 end
    for i=2:n
235 S_s_19(i)= S_s_19(i-1)*(1+r_S_19(i))
    end
237 for i=2:n
    S_s_20(i)= S_s_20(i-1)*(1+r_S_20(i))
239 end
    for i=2:n
241 S_s_21(i)= S_s_21(i-1)*(1+r_S_21(i))
    end
243 for i=2:n
    S_s_22(i)= S_s_22(i-1)*(1+r_S_22(i))
245 end
    for i=2:n
247 S_s_23(i)= S_s_23(i-1)*(1+r_S_23(i))
    end
249 for i=2:n
    S_s_24(i)= S_s_24(i-1)*(1+r_S_24(i))
251 end
    for i=2:n
253 S_s_25(i)= S_s_25(i-1)*(1+r_S_25(i))
    end
255
    Pl_1=sort(S_s_1-S_0)
257 Pl_2=sort(S_s_2-S_0)
    Pl_3=sort(S_s_3-S_0)
259 Pl_4=sort(S_s_4-S_0)
    Pl_5=sort(S_s_5-S_0)
261 Pl_6=sort(S_s_6-S_0)
    Pl_7=sort(S_s_7-S_0)
263 Pl_8=sort(S_s_8-S_0)
    Pl_9=sort(S_s_9-S_0)
265 Pl_10=sort(S_s_10-S_0)
    Pl_11=sort(S_s_11-S_0)
267 Pl_12=sort(S_s_12-S_0)
    Pl_13=sort(S_s_13-S_0)
269 Pl_14=sort(S_s_14-S_0)
    Pl_15=sort(S_s_15-S_0)
271 Pl_16=sort(S_s_16-S_0)
    Pl_17=sort(S_s_17-S_0)
273 Pl_18=sort(S_s_18-S_0)
    Pl_19=sort(S_s_19-S_0)
```

```
Pl_20=sort(S_s_20-S_0)
Pl_21=sort(S_s_21-S_0)
Pl_22=sort(S_s_22-S_0)
Pl_23=sort(S_s_23-S_0)
Pl_24=sort(S_s_24-S_0)
Pl_25=sort(S_s_25-S_0)


VaR_1= Pl_1(n*a)
VaR_2= Pl_2(n*a)
VaR_3= Pl_3(n*a)
VaR_4= Pl_4(n*a)
VaR_5= Pl_5(n*a)
VaR_6= Pl_6(n*a)
VaR_7= Pl_7(n*a)
VaR_8= Pl_8(n*a)
VaR_9= Pl_9(n*a)
VaR_10= Pl_10(n*a)
VaR_11= Pl_11(n*a)
VaR_12= Pl_12(n*a)
VaR_13= Pl_13(n*a)
VaR_14= Pl_14(n*a)
VaR_15= Pl_15(n*a)
VaR_16= Pl_16(n*a)
VaR_17= Pl_17(n*a)
VaR_18= Pl_18(n*a)
VaR_19= Pl_19(n*a)
VaR_20= Pl_20(n*a)
VaR_21= Pl_21(n*a)
VaR_22= Pl_22(n*a)
VaR_23= Pl_23(n*a)
VaR_24= Pl_24(n*a)
VaR_25= Pl_25(n*a)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,V

%% Backtesting

PL_B=B-S_0

figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Quality checks for the model
```

```
%compute the difference between VaR estimate and actual losses
error_HS=zeros(1,25)

for i=1:25
error_HS(i)=Var_sim(i)-PL_B(i)
end

% number of times VaR fails to cover losses
failure_HS=zeros(1,25)
for i=1:25
if error_HS(i)>0
failure_HS(i)=1
end
end

x_HS=sum(failure_HS)%9
p=0.05 %level of confidence
N=25 %number of estimates

%% Binomial test, compare actual and expected number of vailures

exp_val_bin=p*N
test_bin_HS=(x_HS-N*p)/(sqrt(N*p*(1-p))) % test rejected
crit_val_bin=norminv(1-p)
%% Kupiec proportion of failures test

LR_pof_HS=-2*log(((1-p)^(N-x_HS)*p^x_HS)/((1-x_HS/N)^(N-x_HS)*(x_HS/N)^x_HS))
crit=chi2inv((1-p),1) %test failed

%% Kupiec time untill first failure test
%number of days before first rejection
k_HS=8

LR_TUFF_HS=-2*log((p*(1-p)^(k_HS-1))/((1/k_HS)*(1-(1/k_HS))^(k_HS-1)))
crit_TUFF=chi2inv((1-p),1)
% test passed

%% Christoffersen Interval forecast test

n_00_HS=14
n_10_HS=1
n_01_HS=2
n_11_HS=7

p_0_HS=n_01_HS/(n_00_HS+n_01_HS)
p_1_HS=n_11_HS/(n_10_HS+n_11_HS)
p_HS=(n_01_HS+n_11_HS)/(n_00_HS+n_01_HS+n_11_HS)
```

```
LRCCI_HS=-2*log(((1-p_HS)^(n_00_HS+n_10_HS)*p_HS^(n_01_HS+n_11_HS))/((1-p_0_HS
%check failed
LRCC_HS=LRCCI_HS+LR_pof_HS
crit2=chi2inv((1-p),2)
%check failed

%% Mixed Kupiec test, not working

days_HS=[8,8,0,0,0,0,0,0,0]

sumTB_HS=zeros(1,9)

for i=1:9
sumTB_HS=log((p*(1-p)^(days_HS(i)-1))/((1/days_HS(i))*(1-1/days_HS(i))^(days_H
end

LRTBFI_HS=-2*sum(sumTB_HS)
crit_TBFI_HS=chi2inv((1-p),9)
```

```
%% Montecarlo VaR FTSE MIB

%% Descriptive statistics full sample

FM=FTSEMIB(2:end)
FM(242,:)=[]
%max=max(FM)
%min=min(FM)
avg=sum(FM)/300
%std=std(FM)
kurt=kurtosis(FM)
skew=skewness(FM)
%median=median(FM)
%mode=mode(FM)

%% Time series FTSE MIB full sample
bdates=busdays(datetime('24-October-2016','Locale','en_US'),'1-January-2018','
plot(bdates,FM)
%% Initialization

S=FTSEMIB(2:150)% data set for estimation of parameters
B=FTSEMIB(152:300)% backtesting dataset
S_0=FTSEMIB(151)
n=1000%100
k=25%149 %target number of VaR estimations=lenght backtesting dataset
a=0.05

hist(S)
%% Parameter estimation
```

```
m=mean(S)
s=std(S)

%% MC
%initialyze space
S_s_1=S(1)*ones(1,n)
S_s_2=S(1)*ones(1,n)
S_s_3=S(1)*ones(1,n)
S_s_4=S(1)*ones(1,n)
S_s_5=S(1)*ones(1,n)
S_s_6=S(1)*ones(1,n)
S_s_7=S(1)*ones(1,n)
S_s_8=S(1)*ones(1,n)
S_s_9=S(1)*ones(1,n)
S_s_10=S(1)*ones(1,n)
S_s_11=S(1)*ones(1,n)
S_s_12=S(1)*ones(1,n)
S_s_13=S(1)*ones(1,n)
S_s_14=S(1)*ones(1,n)
S_s_15=S(1)*ones(1,n)
S_s_16=S(1)*ones(1,n)
S_s_17=S(1)*ones(1,n)
S_s_18=S(1)*ones(1,n)
S_s_19=S(1)*ones(1,n)
S_s_20=S(1)*ones(1,n)
S_s_21=S(1)*ones(1,n)
S_s_22=S(1)*ones(1,n)
S_s_23=S(1)*ones(1,n)
S_s_24=S(1)*ones(1,n)
S_s_25=S(1)*ones(1,n)

% generation of simulated path
for i=2:n
S_s_1(i)=S_s_1(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end

for i=2:n
S_s_2(i)=S_s_2(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_3(i)=S_s_3(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_4(i)=S_s_4(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_5(i)=S_s_5(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
```

```
end
for i=2:n
S_s_6(i)=S_s_6(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_7(i)=S_s_7(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_8(i)=S_s_8(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_9(i)=S_s_9(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_10(i)=S_s_10(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_11(i)=S_s_11(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_12(i)=S_s_12(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_13(i)=S_s_13(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_14(i)=S_s_14(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_15(i)=S_s_15(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_16(i)=S_s_16(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_17(i)=S_s_17(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_18(i)=S_s_18(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_19(i)=S_s_19(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_20(i)=S_s_20(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_21(i)=S_s_21(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
```

```
end
for i=2:n
S_s_22(i)=S_s_22(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_23(i)=S_s_23(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_24(i)=S_s_24(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_25(i)=S_s_25(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end

%% P&L
%initialyze space
PL_1=zeros(1,n)
PL_2=zeros(1,n)
PL_3=zeros(1,n)
PL_4=zeros(1,n)
PL_5=zeros(1,n)
PL_6=zeros(1,n)
PL_7=zeros(1,n)
PL_8=zeros(1,n)
PL_9=zeros(1,n)
PL_10=zeros(1,n)
PL_11=zeros(1,n)
PL_12=zeros(1,n)
PL_13=zeros(1,n)
PL_14=zeros(1,n)
PL_15=zeros(1,n)
PL_16=zeros(1,n)
PL_17=zeros(1,n)
PL_18=zeros(1,n)
PL_19=zeros(1,n)
PL_20=zeros(1,n)
PL_21=zeros(1,n)
PL_22=zeros(1,n)
PL_23=zeros(1,n)
PL_24=zeros(1,n)
PL_25=zeros(1,n)

for i=1:n
PL_1(i)=S_s_1(i)-S_0
end
for i=1:n
PL_2(i)=S_s_2(i)-S_0
end
```

```
for i=1:n
PL_3(i)=S_s_3(i)-S_0
end
for i=1:n
PL_4(i)=S_s_4(i)-S_0
end
for i=1:n
PL_5(i)=S_s_5(i)-S_0
end
for i=1:n
PL_6(i)=S_s_6(i)-S_0
end
for i=1:n
PL_7(i)=S_s_7(i)-S_0
end
for i=1:n
PL_8(i)=S_s_8(i)-S_0
end
for i=1:n
PL_9(i)=S_s_9(i)-S_0
end
for i=1:n
PL_10(i)=S_s_10(i)-S_0
end
for i=1:n
PL_11(i)=S_s_11(i)-S_0
end
for i=1:n
PL_12(i)=S_s_12(i)-S_0
end
for i=1:n
PL_13(i)=S_s_13(i)-S_0
end
for i=1:n
PL_14(i)=S_s_14(i)-S_0
end
for i=1:n
PL_15(i)=S_s_15(i)-S_0
end
for i=1:n
PL_16(i)=S_s_16(i)-S_0
end
for i=1:n
PL_17(i)=S_s_17(i)-S_0
end
for i=1:n
PL_18(i)=S_s_18(i)-S_0
end
```

```
for i=1:n
PL_19(i)=S_s_19(i)-S_0
end
for i=1:n
PL_20(i)=S_s_20(i)-S_0
end
for i=1:n
PL_21(i)=S_s_21(i)-S_0
end
for i=1:n
PL_22(i)=S_s_22(i)-S_0
end
for i=1:n
PL_23(i)=S_s_23(i)-S_0
end
for i=1:n
PL_24(i)=S_s_24(i)-S_0
end
for i=1:n
PL_25(i)=S_s_25(i)-S_0
end

%% VaR

PL_ord_1=sort(PL_1)
PL_ord_2=sort(PL_2)
PL_ord_3=sort(PL_3)
PL_ord_4=sort(PL_4)
PL_ord_5=sort(PL_5)
PL_ord_6=sort(PL_6)
PL_ord_7=sort(PL_7)
PL_ord_8=sort(PL_8)
PL_ord_9=sort(PL_9)
PL_ord_10=sort(PL_10)
PL_ord_11=sort(PL_11)
PL_ord_12=sort(PL_12)
PL_ord_13=sort(PL_13)
PL_ord_14=sort(PL_14)
PL_ord_15=sort(PL_15)
PL_ord_16=sort(PL_16)
PL_ord_17=sort(PL_17)
PL_ord_18=sort(PL_18)
PL_ord_19=sort(PL_19)
PL_ord_20=sort(PL_20)
PL_ord_21=sort(PL_21)
PL_ord_22=sort(PL_22)
PL_ord_23=sort(PL_23)
PL_ord_24=sort(PL_24)
```

```
PL_ord_25=sort(PL_25)

VaR_1=PL_ord_1(n*a)
VaR_2=PL_ord_2(n*a)
VaR_3=PL_ord_3(n*a)
VaR_4=PL_ord_4(n*a)
VaR_5=PL_ord_5(n*a)
VaR_6=PL_ord_6(n*a)
VaR_7=PL_ord_7(n*a)
VaR_8=PL_ord_8(n*a)
VaR_9=PL_ord_9(n*a)
VaR_10=PL_ord_10(n*a)
VaR_11=PL_ord_11(n*a)
VaR_12=PL_ord_12(n*a)
VaR_13=PL_ord_13(n*a)
VaR_14=PL_ord_14(n*a)
VaR_15=PL_ord_15(n*a)
VaR_16=PL_ord_16(n*a)
VaR_17=PL_ord_17(n*a)
VaR_18=PL_ord_18(n*a)
VaR_19=PL_ord_19(n*a)
VaR_20=PL_ord_20(n*a)
VaR_21=PL_ord_21(n*a)
VaR_22=PL_ord_22(n*a)
VaR_23=PL_ord_23(n*a)
VaR_24=PL_ord_24(n*a)
VaR_25=PL_ord_25(n*a)

Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,V

%% Backtesting

PL_B=B(65:89)-S_0

figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Quality checks for the model
%compute the difference between VaR estimate and actual losses
error_MC=zeros(1,25)

for i=1:25
```

```
error_MC(i)=Var_sim(i)-PL_B(i)
end

% number of times VaR fails to cover losses
failure_MC=zeros(1,25)
for i=1:25
if error_MC(i)>0
failure_MC(i)=1
end
end

x_MC=sum(failure_MC)
p=0.05 %level of confidence
N=25 %number of estimates

%% Binomial test, compare actual and expected number of vailures

exp_val_bin=p*N
test_bin_MC=(x_MC-N*p)/(sqrt(N*p*(1-p)))
crit_val_bin=norminv(1-p) %test passed
%% Kupiec proportion of failures test

LR_pof_MC=-2*log(((1-p)^(N-x_MC)*p^x_MC)/((1-x_MC/N)^(N-x_MC)*(x_MC/N)^x_MC))
crit=chi2inv((1-p),1) %test passed

%% Kupiec time untill first failure test
%number of days before first rejection
k_MC=19

LR_TUFF_MC=-2*log((p*(1-p)^(k_MC-1))/((1/k_MC)*(1-(1/k_MC))^(k_MC-1)))
crit_TUFF=chi2inv((1-p),1)
% test passed

%% Christoffersen Interval forecast test

n_00_MC=22
n_10_MC=1
n_01_MC=1
n_11_MC=0

p_0_MC=n_01_MC/(n_00_MC+n_01_MC)
p_1_MC=n_11_MC/(n_10_MC+n_11_MC)
p_MC=(n_01_MC+n_11_MC)/(n_00_MC+n_01_MC+n_11_MC)

LRCCI_MC=-2*log(((1-p_MC)^(n_00_MC+n_10_MC)*p_MC^(n_01_MC+n_11_MC))/((1-p_0_MC)^(n
%check passed
LRCC_MC=LRCCI_MC+LR_pof_MC
crit2=chi2inv((1-p),2)
```

```
%check passed

%% Mixed Kupiec test, not working

days_MC=[0,0]

sumTB_MC=zeros(1,2)

for i=1:2
sumTB_MC=log((p*(1-p)^(days_MC(i)-1))/((1/days_MC(i))*(1-1/days_MC(i))^(days_M
end

LRTBFI_MC=-2*sum(sumTB_MC)
crit_TBFI_MC=chi2inv((1-p),2)
```

B.5.2 S&P500 Study

```
%% VaR SP500 Historical Variance Covariance

S=SP500(2:end)
S(242,:)=[]
S_0=S(151)


%% Initialization

B=SP500(30:55)% backtesting dataset
a=0.05


%% VaR
VaR_1=mean(S(1:25))/250-1.65*std(S(1:25))*sqrt(1/250)
VaR_2=mean(S(2:26))/250-1.65*std(S(2:26))*sqrt(1/250)
VaR_3=mean(S(3:27))/250-1.65*std(S(3:27))*sqrt(1/250)
VaR_4=mean(S(4:28))/250-1.65*std(S(4:28))*sqrt(1/250)
VaR_5=mean(S(5:29))/250-1.65*std(S(5:29))*sqrt(1/250)
VaR_6=mean(S(6:30))/250-1.65*std(S(6:30))*sqrt(1/250)
VaR_7=mean(S(7:31))/250-1.65*std(S(7:31))*sqrt(1/250)
VaR_8=mean(S(8:32))/250-1.65*std(S(8:32))*sqrt(1/250)
VaR_9=mean(S(9:33))/250-1.65*std(S(9:33))*sqrt(1/250)
VaR_10=mean(S(10:34))/250-1.65*std(S(10:34))*sqrt(1/250)
VaR_11=mean(S(11:35))/250-1.65*std(S(11:35))*sqrt(1/250)
VaR_12=mean(S(12:36))/250-1.65*std(S(12:36))*sqrt(1/250)
VaR_13=mean(S(13:37))/250-1.65*std(S(13:37))*sqrt(1/250)
VaR_14=mean(S(14:38))/250-1.65*std(S(14:38))*sqrt(1/250)
VaR_15=mean(S(15:39))/250-1.65*std(S(15:39))*sqrt(1/250)
VaR_16=mean(S(16:40))/250-1.65*std(S(16:40))*sqrt(1/250)
```

```
VaR_17=mean(S(17:41))/250-1.65*std(S(17:41))*sqrt(1/250)
VaR_18=mean(S(18:42))/250-1.65*std(S(18:42))*sqrt(1/250)
VaR_19=mean(S(19:43))/250-1.65*std(S(19:43))*sqrt(1/250)
VaR_20=mean(S(20:44))/250-1.65*std(S(20:44))*sqrt(1/250)
VaR_21=mean(S(21:45))/250-1.65*std(S(21:45))*sqrt(1/250)
VaR_22=mean(S(22:46))/250-1.65*std(S(22:46))*sqrt(1/250)
VaR_23=mean(S(23:47))/250-1.65*std(S(23:47))*sqrt(1/250)
VaR_24=mean(S(24:48))/250-1.65*std(S(24:48))*sqrt(1/250)
VaR_25=mean(S(25:49))/250-1.65*std(S(25:49))*sqrt(1/250)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,VaR_1

%% Backtesting

PL_B=B-S_0


figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Acceptability
%not acceptable
```

```
%% VaR SP500 Historical simulation


SP=SP500(2:end)
SP(242,:)=[]
B=SP500(30:55)% backtesting dataset
S_0=SP500(151)

%% Initialization
%5-day rolling window
S_1=SP(1:100)%first dataset
S_2=SP(5:104)%first dataset
S_3=SP(10:109)%first dataset
S_4=SP(15:114)%first dataset
S_5=SP(20:119)%first dataset
S_6=SP(25:124)%first dataset
S_7=SP(30:129)%first dataset
S_8=SP(35:134)%first dataset
S_9=SP(40:139)%first dataset
S_10=SP(45:144)%first dataset
S_11=SP(50:149)%first dataset
```

```
S_12=SP(55:154)%first dataset
S_13=SP(60:159)%first dataset
S_14=SP(65:164)%first dataset
S_15=SP(70:169)%first dataset
S_16=SP(75:174)%first dataset
S_17=SP(80:179)%first dataset
S_18=SP(85:184)%first dataset
S_19=SP(90:189)%first dataset
S_20=SP(95:194)%first dataset
S_21=SP(100:199)%first dataset
S_22=SP(105:204)%first dataset
S_23=SP(110:209)%first dataset
S_24=SP(115:214)%first dataset
S_25=SP(120:219)%first dataset

S_0=SP500(151)
n=100%100
k=25%149 %target number of VaR estimations=lenght backtesting dataset
a=0.05

%% Historical simulation

r_S_1=zeros(1,n)' %percentage changes
r_S_2=zeros(1,n)'
r_S_3=zeros(1,n)'
r_S_4=zeros(1,n)'
r_S_5=zeros(1,n)'
r_S_6=zeros(1,n)'
r_S_7=zeros(1,n)'
r_S_8=zeros(1,n)'
r_S_9=zeros(1,n)'
r_S_10=zeros(1,n)'
r_S_11=zeros(1,n)'
r_S_12=zeros(1,n)'
r_S_13=zeros(1,n)'
r_S_14=zeros(1,n)'
r_S_15=zeros(1,n)'
r_S_16=zeros(1,n)'
r_S_17=zeros(1,n)'
r_S_18=zeros(1,n)'
r_S_19=zeros(1,n)'
r_S_20=zeros(1,n)'
r_S_21=zeros(1,n)'
r_S_22=zeros(1,n)'
r_S_23=zeros(1,n)'
r_S_24=zeros(1,n)'
```

```
r_S_25=zeros(1,n)'

for i=1:n-1
r_S_1(i)=S_1(i+1)/S_1(i)-1
end
for i=1:n-1
r_S_2(i)=S_2(i+1)/S_2(i)-1
end
for i=1:n-1
r_S_3(i)=S_3(i+1)/S_3(i)-1
end
for i=1:n-1
r_S_4(i)=S_4(i+1)/S_4(i)-1
end
for i=1:n-1
r_S_5(i)=S_5(i+1)/S_5(i)-1
end
for i=1:n-1
r_S_6(i)=S_6(i+1)/S_6(i)-1
end
for i=1:n-1
r_S_7(i)=S_7(i+1)/S_7(i)-1
end
for i=1:n-1
r_S_8(i)=S_8(i+1)/S_8(i)-1
end
for i=1:n-1
r_S_9(i)=S_9(i+1)/S_9(i)-1
end
for i=1:n-1
r_S_10(i)=S_10(i+1)/S_10(i)-1
end
for i=1:n-1
r_S_11(i)=S_11(i+1)/S_11(i)-1
end
for i=1:n-1
r_S_12(i)=S_12(i+1)/S_12(i)-1
end
for i=1:n-1
r_S_13(i)=S_13(i+1)/S_13(i)-1
end
for i=1:n-1
r_S_14(i)=S_14(i+1)/S_14(i)-1
end
for i=1:n-1
r_S_15(i)=S_15(i+1)/S_15(i)-1
end
for i=1:n-1
```

```
r_S_16(i)=S_16(i+1)/S_16(i)-1
end
for i=1:n-1
r_S_17(i)=S_17(i+1)/S_17(i)-1
end
for i=1:n-1
r_S_18(i)=S_18(i+1)/S_18(i)-1
end
for i=1:n-1
r_S_19(i)=S_19(i+1)/S_19(i)-1
end
for i=1:n-1
r_S_20(i)=S_20(i+1)/S_20(i)-1
end
for i=1:n-1
r_S_21(i)=S_21(i+1)/S_21(i)-1
end
for i=1:n-1
r_S_22(i)=S_22(i+1)/S_22(i)-1
end
for i=1:n-1
r_S_23(i)=S_23(i+1)/S_23(i)-1
end
for i=1:n-1
r_S_24(i)=S_24(i+1)/S_24(i)-1
end
for i=1:n-1
r_S_25(i)=S_25(i+1)/S_25(i)-1
end


S_s_1=ones(1,n)'*S_1(1)
S_s_2=ones(1,n)'*S_2(1)
S_s_3=ones(1,n)'*S_3(1)
S_s_4=ones(1,n)'*S_4(1)
S_s_5=ones(1,n)'*S_5(1)
S_s_6=ones(1,n)'*S_6(1)
S_s_7=ones(1,n)'*S_7(1)
S_s_8=ones(1,n)'*S_8(1)
S_s_9=ones(1,n)'*S_9(1)
S_s_10=ones(1,n)'*S_10(1)
S_s_11=ones(1,n)'*S_11(1)
S_s_12=ones(1,n)'*S_12(1)
S_s_13=ones(1,n)'*S_13(1)
S_s_14=ones(1,n)'*S_14(1)
S_s_15=ones(1,n)'*S_15(1)
S_s_16=ones(1,n)'*S_16(1)
S_s_17=ones(1,n)'*S_17(1)
```

```
S_s_18=ones(1,n)'*S_18(1)
S_s_19=ones(1,n)'*S_19(1)
S_s_20=ones(1,n)'*S_20(1)
S_s_21=ones(1,n)'*S_21(1)
S_s_22=ones(1,n)'*S_22(1)
S_s_23=ones(1,n)'*S_23(1)
S_s_24=ones(1,n)'*S_24(1)
S_s_25=ones(1,n)'*S_25(1)


for i=2:n
S_s_1(i)= S_s_1(i-1)*(1+r_S_1(i))
end

for i=2:n
S_s_1(i)= S_s_1(i-1)*(1+r_S_1(i))
end
for i=2:n
S_s_2(i)= S_s_2(i-1)*(1+r_S_2(i))
end
for i=2:n
S_s_3(i)= S_s_3(i-1)*(1+r_S_3(i))
end
for i=2:n
S_s_4(i)= S_s_4(i-1)*(1+r_S_4(i))
end
for i=2:n
S_s_5(i)= S_s_5(i-1)*(1+r_S_5(i))
end
for i=2:n
S_s_6(i)= S_s_6(i-1)*(1+r_S_6(i))
end
for i=2:n
S_s_7(i)= S_s_7(i-1)*(1+r_S_7(i))
end
for i=2:n
S_s_8(i)= S_s_8(i-1)*(1+r_S_8(i))
end
for i=2:n
S_s_9(i)= S_s_9(i-1)*(1+r_S_9(i))
end
for i=2:n
S_s_10(i)= S_s_10(i-1)*(1+r_S_10(i))
end
for i=2:n
S_s_11(i)= S_s_11(i-1)*(1+r_S_11(i))
end
for i=2:n
```

```
214 S_s_12(i)= S_s_12(i-1)*(1+r_S_12(i))
    end
216 for i=2:n
    S_s_13(i)= S_s_13(i-1)*(1+r_S_13(i))
218 end
    for i=2:n
220 S_s_14(i)= S_s_14(i-1)*(1+r_S_14(i))
    end
222 for i=2:n
    S_s_15(i)= S_s_15(i-1)*(1+r_S_15(i))
224 end
    for i=2:n
226 S_s_16(i)= S_s_16(i-1)*(1+r_S_16(i))
    end
228 for i=2:n
    S_s_17(i)= S_s_17(i-1)*(1+r_S_17(i))
230 end
    for i=2:n
232 S_s_18(i)= S_s_18(i-1)*(1+r_S_18(i))
    end
234 for i=2:n
    S_s_19(i)= S_s_19(i-1)*(1+r_S_19(i))
236 end
    for i=2:n
238 S_s_20(i)= S_s_20(i-1)*(1+r_S_20(i))
    end
240 for i=2:n
    S_s_21(i)= S_s_21(i-1)*(1+r_S_21(i))
242 end
    for i=2:n
244 S_s_22(i)= S_s_22(i-1)*(1+r_S_22(i))
    end
246 for i=2:n
    S_s_23(i)= S_s_23(i-1)*(1+r_S_23(i))
248 end
    for i=2:n
250 S_s_24(i)= S_s_24(i-1)*(1+r_S_24(i))
    end
252 for i=2:n
    S_s_25(i)= S_s_25(i-1)*(1+r_S_25(i))
254 end

256 Pl_1=sort(S_s_1-S_0)
    Pl_2=sort(S_s_2-S_0)
258 Pl_3=sort(S_s_3-S_0)
    Pl_4=sort(S_s_4-S_0)
260 Pl_5=sort(S_s_5-S_0)
    Pl_6=sort(S_s_6-S_0)
```

```
Pl_7=sort(S_s_7-S_0)
Pl_8=sort(S_s_8-S_0)
Pl_9=sort(S_s_9-S_0)
Pl_10=sort(S_s_10-S_0)
Pl_11=sort(S_s_11-S_0)
Pl_12=sort(S_s_12-S_0)
Pl_13=sort(S_s_13-S_0)
Pl_14=sort(S_s_14-S_0)
Pl_15=sort(S_s_15-S_0)
Pl_16=sort(S_s_16-S_0)
Pl_17=sort(S_s_17-S_0)
Pl_18=sort(S_s_18-S_0)
Pl_19=sort(S_s_19-S_0)
Pl_20=sort(S_s_20-S_0)
Pl_21=sort(S_s_21-S_0)
Pl_22=sort(S_s_22-S_0)
Pl_23=sort(S_s_23-S_0)
Pl_24=sort(S_s_24-S_0)
Pl_25=sort(S_s_25-S_0)


VaR_1= Pl_1(n*a)
VaR_2= Pl_2(n*a)
VaR_3= Pl_3(n*a)
VaR_4= Pl_4(n*a)
VaR_5= Pl_5(n*a)
VaR_6= Pl_6(n*a)
VaR_7= Pl_7(n*a)
VaR_8= Pl_8(n*a)
VaR_9= Pl_9(n*a)
VaR_10= Pl_10(n*a)
VaR_11= Pl_11(n*a)
VaR_12= Pl_12(n*a)
VaR_13= Pl_13(n*a)
VaR_14= Pl_14(n*a)
VaR_15= Pl_15(n*a)
VaR_16= Pl_16(n*a)
VaR_17= Pl_17(n*a)
VaR_18= Pl_18(n*a)
VaR_19= Pl_19(n*a)
VaR_20= Pl_20(n*a)
VaR_21= Pl_21(n*a)
VaR_22= Pl_22(n*a)
VaR_23= Pl_23(n*a)
VaR_24= Pl_24(n*a)
VaR_25= Pl_25(n*a)

```

```
Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,V

%% Backtesting

PL_B=B-S_0

figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Quality checks for the model
%compute the difference between VaR estimate and actual losses
error_HS=zeros(1,25)

for i=1:25
error_HS(i)=Var_sim(i)-PL_B(i)
end

% number of times VaR fails to cover losses
failure_HS=zeros(1,25)
for i=1:25
if error_HS(i)>0
failure_HS(i)=1
end
end

x_HS=sum(failure_HS)%9
p=0.05 %level of confidence
N=25 %number of estimates

%% Binomial test, compare actual and expected number of vailures

exp_val_bin=p*N
test_bin_HS=(x_HS-N*p)/(sqrt(N*p*(1-p))) % test rejected
crit_val_bin=norminv(1-p)
%% Kupiec proportion of failures test

LR_pof_HS=-2*log(((1-p)^(N-x_HS)*p^x_HS)/((1-x_HS/N)^(N-x_HS)*(x_HS/N)^x_HS))
crit=chi2inv((1-p),1) %test failed

%% Kupiec time untill first failure test
%number of days before first rejection
k_HS=0

LR_TUFF_HS=-2*log((p*(1-p)^(k_HS-1))/((1/k_HS)*(1-(1/k_HS))^(k_HS-1)))
crit_TUFF=chi2inv((1-p),1)
```

```
% test passed

%% Christoffersen Interval forecast test

n_00_HS=3
n_10_HS=1
n_01_HS=1
n_11_HS=19

p_0_HS=n_01_HS/(n_00_HS+n_01_HS)
p_1_HS=n_11_HS/(n_10_HS+n_11_HS)
p_HS=(n_01_HS+n_11_HS)/(n_00_HS+n_01_HS+n_11_HS)

LRCCI_HS=-2*log(((1-p_HS)^(n_00_HS+n_10_HS)*p_HS^(n_01_HS+n_11_HS))/((1-p_0_HS)^(n
%check failed
LRCC_HS=LRCCI_HS+LR_pof_HS
crit2=chi2inv((1-p),2)
%check failed

%% Mixed Kupiec test, not working

days_HS=[8,8,0,0,0,0,0,0,0]

sumTB_HS=zeros(1,9)

for i=1:9
sumTB_HS=log((p*(1-p)^(days_HS(i)-1))/((1/days_HS(i))*(1-1/days_HS(i))^(days_HS(i)
end

LRTBFI_HS=-2*sum(sumTB_HS)
crit_TBFI_HS=chi2inv((1-p),9)
```

```
%% Descriptive statistics full sample

SP=SP500(2:end)/100
SP(242,:)=[]
%max=max(S)
%min=min(S)
%avg=mean(SP)
%std=std(S)
%kurt=kurtosis(SP)
%skew=skewness(SP)
%median=median(S)
%mode=mode(S)

%% Time series FTSE MIB full sample
bdates=busdays(datetime('24-October-2016','Locale','en_US'),'1-January-2018','dai
plot(bdates,SP(205:end))
```

```
%% Initialization
B=SP500(30:55)% backtesting dataset

S=SP(2:150)% data set for estimation of parameters
%B=SP(152:300)% backtesting dataset
S_0=SP(151)
n=1000%100
k=25%149 %target number of VaR estimations=lenght backtesting dataset
a=0.05

hist(SP)
%% Parameter estimation

m=mean(S)
s=std(S)

%% MC
%initialyze space
S_s_1=S(1)*ones(1,n)
S_s_2=S(1)*ones(1,n)
S_s_3=S(1)*ones(1,n)
S_s_4=S(1)*ones(1,n)
S_s_5=S(1)*ones(1,n)
S_s_6=S(1)*ones(1,n)
S_s_7=S(1)*ones(1,n)
S_s_8=S(1)*ones(1,n)
S_s_9=S(1)*ones(1,n)
S_s_10=S(1)*ones(1,n)
S_s_11=S(1)*ones(1,n)
S_s_12=S(1)*ones(1,n)
S_s_13=S(1)*ones(1,n)
S_s_14=S(1)*ones(1,n)
S_s_15=S(1)*ones(1,n)
S_s_16=S(1)*ones(1,n)
S_s_17=S(1)*ones(1,n)
S_s_18=S(1)*ones(1,n)
S_s_19=S(1)*ones(1,n)
S_s_20=S(1)*ones(1,n)
S_s_21=S(1)*ones(1,n)
S_s_22=S(1)*ones(1,n)
S_s_23=S(1)*ones(1,n)
S_s_24=S(1)*ones(1,n)
S_s_25=S(1)*ones(1,n)

% generation of simulated path
for i=2:n
S_s_1(i)=S_s_1(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
```

```
65
for i=2:n
67 S_s_2(i)=S_s_2(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
69 for i=2:n
S_s_3(i)=S_s_3(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
71 end
for i=2:n
73 S_s_4(i)=S_s_4(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
75 for i=2:n
S_s_5(i)=S_s_5(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
77 end
for i=2:n
79 S_s_6(i)=S_s_6(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
81 for i=2:n
S_s_7(i)=S_s_7(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
83 end
for i=2:n
85 S_s_8(i)=S_s_8(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
87 for i=2:n
S_s_9(i)=S_s_9(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
89 end
for i=2:n
91 S_s_10(i)=S_s_10(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
93 for i=2:n
S_s_11(i)=S_s_11(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
95 end
for i=2:n
97 S_s_12(i)=S_s_12(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
99 for i=2:n
S_s_13(i)=S_s_13(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
101 end
for i=2:n
103 S_s_14(i)=S_s_14(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
105 for i=2:n
S_s_15(i)=S_s_15(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
107 end
for i=2:n
109 S_s_16(i)=S_s_16(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
111 for i=2:n
S_s_17(i)=S_s_17(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
```

```
end
for i=2:n
S_s_18(i)=S_s_18(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_19(i)=S_s_19(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_20(i)=S_s_20(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_21(i)=S_s_21(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_22(i)=S_s_22(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_23(i)=S_s_23(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_24(i)=S_s_24(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_25(i)=S_s_25(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end

%% P&L
%initialyze space
PL_1=zeros(1,n)
PL_2=zeros(1,n)
PL_3=zeros(1,n)
PL_4=zeros(1,n)
PL_5=zeros(1,n)
PL_6=zeros(1,n)
PL_7=zeros(1,n)
PL_8=zeros(1,n)
PL_9=zeros(1,n)
PL_10=zeros(1,n)
PL_11=zeros(1,n)
PL_12=zeros(1,n)
PL_13=zeros(1,n)
PL_14=zeros(1,n)
PL_15=zeros(1,n)
PL_16=zeros(1,n)
PL_17=zeros(1,n)
PL_18=zeros(1,n)
PL_19=zeros(1,n)
PL_20=zeros(1,n)
```

```
PL_21=zeros(1,n)
PL_22=zeros(1,n)
PL_23=zeros(1,n)
PL_24=zeros(1,n)
PL_25=zeros(1,n)

for i=1:n
PL_1(i)=S_s_1(i)-S_0
end
for i=1:n
PL_2(i)=S_s_2(i)-S_0
end
for i=1:n
PL_3(i)=S_s_3(i)-S_0
end
for i=1:n
PL_4(i)=S_s_4(i)-S_0
end
for i=1:n
PL_5(i)=S_s_5(i)-S_0
end
for i=1:n
PL_6(i)=S_s_6(i)-S_0
end
for i=1:n
PL_7(i)=S_s_7(i)-S_0
end
for i=1:n
PL_8(i)=S_s_8(i)-S_0
end
for i=1:n
PL_9(i)=S_s_9(i)-S_0
end
for i=1:n
PL_10(i)=S_s_10(i)-S_0
end
for i=1:n
PL_11(i)=S_s_11(i)-S_0
end
for i=1:n
PL_12(i)=S_s_12(i)-S_0
end
for i=1:n
PL_13(i)=S_s_13(i)-S_0
end
for i=1:n
PL_14(i)=S_s_14(i)-S_0
end
```

```
for i=1:n
PL_15(i)=S_s_15(i)-S_0
end
for i=1:n
PL_16(i)=S_s_16(i)-S_0
end
for i=1:n
PL_17(i)=S_s_17(i)-S_0
end
for i=1:n
PL_18(i)=S_s_18(i)-S_0
end
for i=1:n
PL_19(i)=S_s_19(i)-S_0
end
for i=1:n
PL_20(i)=S_s_20(i)-S_0
end
for i=1:n
PL_21(i)=S_s_21(i)-S_0
end
for i=1:n
PL_22(i)=S_s_22(i)-S_0
end
for i=1:n
PL_23(i)=S_s_23(i)-S_0
end
for i=1:n
PL_24(i)=S_s_24(i)-S_0
end
for i=1:n
PL_25(i)=S_s_25(i)-S_0
end

%% VaR

PL_ord_1=sort(PL_1)
PL_ord_2=sort(PL_2)
PL_ord_3=sort(PL_3)
PL_ord_4=sort(PL_4)
PL_ord_5=sort(PL_5)
PL_ord_6=sort(PL_6)
PL_ord_7=sort(PL_7)
PL_ord_8=sort(PL_8)
PL_ord_9=sort(PL_9)
PL_ord_10=sort(PL_10)
PL_ord_11=sort(PL_11)
PL_ord_12=sort(PL_12)
```

```
PL_ord_13=sort(PL_13)
PL_ord_14=sort(PL_14)
PL_ord_15=sort(PL_15)
PL_ord_16=sort(PL_16)
PL_ord_17=sort(PL_17)
PL_ord_18=sort(PL_18)
PL_ord_19=sort(PL_19)
PL_ord_20=sort(PL_20)
PL_ord_21=sort(PL_21)
PL_ord_22=sort(PL_22)
PL_ord_23=sort(PL_23)
PL_ord_24=sort(PL_24)
PL_ord_25=sort(PL_25)


VaR_1=PL_ord_1(n*a)
VaR_2=PL_ord_2(n*a)
VaR_3=PL_ord_3(n*a)
VaR_4=PL_ord_4(n*a)
VaR_5=PL_ord_5(n*a)
VaR_6=PL_ord_6(n*a)
VaR_7=PL_ord_7(n*a)
VaR_8=PL_ord_8(n*a)
VaR_9=PL_ord_9(n*a)
VaR_10=PL_ord_10(n*a)
VaR_11=PL_ord_11(n*a)
VaR_12=PL_ord_12(n*a)
VaR_13=PL_ord_13(n*a)
VaR_14=PL_ord_14(n*a)
VaR_15=PL_ord_15(n*a)
VaR_16=PL_ord_16(n*a)
VaR_17=PL_ord_17(n*a)
VaR_18=PL_ord_18(n*a)
VaR_19=PL_ord_19(n*a)
VaR_20=PL_ord_20(n*a)
VaR_21=PL_ord_21(n*a)
VaR_22=PL_ord_22(n*a)
VaR_23=PL_ord_23(n*a)
VaR_24=PL_ord_24(n*a)
VaR_25=PL_ord_25(n*a)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,VaR_1

%% Backtesting

PL_B=B-S_0
```

```
305
figure;
307 plot( Var_sim, "r");
hold on;
309 plot(PL_B, "g");
hold off;
311
%% Quality checks for the model
313 %compute the difference between VaR estimate and actual losses
error_MC=zeros(1,25)
315
for i=1:25
317 error_MC(i)=Var_sim(i)-PL_B(i)
end
319
% number of times VaR fails to cover losses
321 failure_MC=zeros(1,25)
for i=1:25
323 if error_MC(i)>0
failure_MC(i)=1
325 end
end
327
x_MC=sum(failure_MC)
329 p=0.05 %level of confidence
N=25 %number of estimates
331
%% Binomial test, compare actual and expected number of vailures
333
exp_val_bin=p*N
335 test_bin_MC=(x_MC-N*p)/(sqrt(N*p*(1-p)))
crit_val_bin=norminv(1-p) %test failed
337 %% Kupiec proportion of failures test

339 LR_pof_MC=-2*log(((1-p)^(N-x_MC)*p^x_MC)/((1-x_MC/N)^(N-x_MC)*(x_MC/N)^x_MC))
crit=chi2inv((1-p),1) %test failed
341
%% Kupiec time untill first failure test
343 %number of days before first rejection
k_MC=0
345
LR_TUFF_MC=-2*log((p*(1-p)^(k_MC-1))/((1/k_MC)*(1-(1/k_MC))^(k_MC-1)))
347 crit_TUFF=chi2inv((1-p),1)
% test passed
349
%% Christoffersen Interval forecast test
351
n_00_MC=16
```

```
n_10_MC=3
n_01_MC=2
n_11_MC=3

p_0_MC=n_01_MC/(n_00_MC+n_01_MC)
p_1_MC=n_11_MC/(n_10_MC+n_11_MC)
p_MC=(n_01_MC+n_11_MC)/(n_00_MC+n_01_MC+n_11_MC)

LRCCI_MC=-2*log(((1-p_MC)^(n_00_MC+n_10_MC)*p_MC^(n_01_MC+n_11_MC))/((1-p_0_MC)^(n
%check passed
LRCC_MC=LRCCI_MC+LR_pof_MC
crit2=chi2inv((1-p),2)
%check failed

%% Mixed Kupiec test, not working

days_MC=[0,0]

sumTB_MC=zeros(1,2)

for i=1:2
sumTB_MC=log((p*(1-p)^(days_MC(i)-1))/((1/days_MC(i))*(1-1/days_MC(i))^(days_MC(i)
end

LRTBFI_MC=-2*sum(sumTB_MC)
crit_TBFI_MC=chi2inv((1-p),2)
```

B.5.3 Distributional assumption Study

```
%% Montecarlo VaR Normal

%% Initialization

S=FTSEMIB(2:150)% data set for estimation of parameters
B=FTSEMIB(152:300)% backtesting dataset
S_0=FTSEMIB(151)
n=1000%100
k=25%149 %target number of VaR estimations=lenght backtesting dataset
a=0.05

hist(S)
%% Parameter estimation

[muhat,sigmahat] = normfit(S)


%% MC
```

```
%initialyze space
S_s_1=normrnd(muhat,sigmahat,1,n)
S_s_2=normrnd(muhat,sigmahat,1,n)
S_s_3=normrnd(muhat,sigmahat,1,n)
S_s_4=normrnd(muhat,sigmahat,1,n)
S_s_5=normrnd(muhat,sigmahat,1,n)
S_s_6=normrnd(muhat,sigmahat,1,n)
S_s_7=normrnd(muhat,sigmahat,1,n)
S_s_8=normrnd(muhat,sigmahat,1,n)
S_s_9=normrnd(muhat,sigmahat,1,n)
S_s_10=normrnd(muhat,sigmahat,1,n)
S_s_11=normrnd(muhat,sigmahat,1,n)
S_s_12=normrnd(muhat,sigmahat,1,n)
S_s_13=normrnd(muhat,sigmahat,1,n)
S_s_14=normrnd(muhat,sigmahat,1,n)
S_s_15=normrnd(muhat,sigmahat,1,n)
S_s_16=normrnd(muhat,sigmahat,1,n)
S_s_17=normrnd(muhat,sigmahat,1,n)
S_s_18=normrnd(muhat,sigmahat,1,n)
S_s_19=normrnd(muhat,sigmahat,1,n)
S_s_20=normrnd(muhat,sigmahat,1,n)
S_s_21=normrnd(muhat,sigmahat,1,n)
S_s_22=normrnd(muhat,sigmahat,1,n)
S_s_23=normrnd(muhat,sigmahat,1,n)
S_s_24=normrnd(muhat,sigmahat,1,n)
S_s_25=normrnd(muhat,sigmahat,1,n)



%% P&L
%initialyze space
PL_1=S_s_1-S_0
PL_2=S_s_2-S_0
PL_3=S_s_3-S_0
PL_4=S_s_4-S_0
PL_5=S_s_5-S_0
PL_6=S_s_6-S_0
PL_7=S_s_7-S_0
PL_8=S_s_8-S_0
PL_9=S_s_9-S_0
PL_10=S_s_10-S_0
PL_11=S_s_11-S_0
PL_12=S_s_12-S_0
PL_13=S_s_13-S_0
PL_14=S_s_14-S_0
PL_15=S_s_15-S_0
PL_16=S_s_16-S_0
PL_17=S_s_17-S_0
```

```
PL_18=S_s_18-S_0
PL_19=S_s_19-S_0
PL_20=S_s_20-S_0
PL_21=S_s_21-S_0
PL_22=S_s_22-S_0
PL_23=S_s_23-S_0
PL_24=S_s_24-S_0
PL_25=S_s_25-S_0

%% VaR

PL_ord_1=sort(PL_1)
PL_ord_2=sort(PL_2)
PL_ord_3=sort(PL_3)
PL_ord_4=sort(PL_4)
PL_ord_5=sort(PL_5)
PL_ord_6=sort(PL_6)
PL_ord_7=sort(PL_7)
PL_ord_8=sort(PL_8)
PL_ord_9=sort(PL_9)
PL_ord_10=sort(PL_10)
PL_ord_11=sort(PL_11)
PL_ord_12=sort(PL_12)
PL_ord_13=sort(PL_13)
PL_ord_14=sort(PL_14)
PL_ord_15=sort(PL_15)
PL_ord_16=sort(PL_16)
PL_ord_17=sort(PL_17)
PL_ord_18=sort(PL_18)
PL_ord_19=sort(PL_19)
PL_ord_20=sort(PL_20)
PL_ord_21=sort(PL_21)
PL_ord_22=sort(PL_22)
PL_ord_23=sort(PL_23)
PL_ord_24=sort(PL_24)
PL_ord_25=sort(PL_25)


VaR_1=PL_ord_1(n*a)
VaR_2=PL_ord_2(n*a)
VaR_3=PL_ord_3(n*a)
VaR_4=PL_ord_4(n*a)
VaR_5=PL_ord_5(n*a)
VaR_6=PL_ord_6(n*a)
VaR_7=PL_ord_7(n*a)
VaR_8=PL_ord_8(n*a)
VaR_9=PL_ord_9(n*a)
VaR_10=PL_ord_10(n*a)
```

```
VaR_11=PL_ord_11(n*a)
VaR_12=PL_ord_12(n*a)
VaR_13=PL_ord_13(n*a)
VaR_14=PL_ord_14(n*a)
VaR_15=PL_ord_15(n*a)
VaR_16=PL_ord_16(n*a)
VaR_17=PL_ord_17(n*a)
VaR_18=PL_ord_18(n*a)
VaR_19=PL_ord_19(n*a)
VaR_20=PL_ord_20(n*a)
VaR_21=PL_ord_21(n*a)
VaR_22=PL_ord_22(n*a)
VaR_23=PL_ord_23(n*a)
VaR_24=PL_ord_24(n*a)
VaR_25=PL_ord_25(n*a)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,V

%% Backtesting

PL_B=B(65:89)-S_0


figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Quality checks for the model
%compute the difference between VaR estimate and actual losses
error_MC=zeros(1,25)

for i=1:25
error_MC(i)=Var_sim(i)-PL_B(i)
end

% number of times VaR fails to cover losses
failure_MC=zeros(1,25)
for i=1:25
if error_MC(i)>0
failure_MC(i)=1
end
end

x_MC=sum(failure_MC)
p=0.05 %level of confidence
```

```
163 N=25 %number of estimates

165 %% Binomial test, compare actual and expected number of vailures

167 exp_val_bin=p*N
    test_bin_MC=(x_MC-N*p)/(sqrt(N*p*(1-p)))
169 crit_val_bin=norminv(1-p) %test passed
    %% Kupiec proportion of failures test
171
    LR_pof_MC=-2*log(((1-p)^(N-x_MC)*p^x_MC)/((1-x_MC/N)^(N-x_MC)*(x_MC/N)^x_MC))
173 crit=chi2inv((1-p),1) %test passed

175 %% Kupiec time untill first failure test
    %number of days before first rejection
177 k_MC=1

179 LR_TUFF_MC=-2*log((p*(1-p)^(k_MC-1))/((1/k_MC)*(1-(1/k_MC))^(k_MC-1)))
    crit_TUFF=chi2inv((1-p),1)
181 % test passed

183 %% Christoffersen Interval forecast test

185 n_00_MC=15
    n_10_MC=1
187 n_01_MC=0
    n_11_MC=7
189
    p_0_MC=n_01_MC/(n_00_MC+n_01_MC)
191 p_1_MC=n_11_MC/(n_10_MC+n_11_MC)
    p_MC=(n_01_MC+n_11_MC)/(n_00_MC+n_01_MC+n_11_MC)
193
    LRCCI_MC=-2*log(((1-p_MC)^(n_00_MC+n_10_MC)*p_MC^(n_01_MC+n_11_MC))/((1-p_0_MC)^(n
195 %check passed
    LRCC_MC=LRCCI_MC+LR_pof_MC
197 crit2=chi2inv((1-p),2)
    %check passed
199
    %% Mixed Kupiec test, not working
201
    days_MC=[0,0]
203
    sumTB_MC=zeros(1,2)
205
    for i=1:2
207 sumTB_MC=log((p*(1-p)^(days_MC(i)-1))/((1/days_MC(i))*(1-1/days_MC(i))^(days_MC(i)
    end
209
    LRTBFI_MC=-2*sum(sumTB_MC)
```

```
211 crit_TBFI_MC=chi2inv((1-p),2)
```

```
%% Montecarlo VaR Log-Normal
2
%% Initialization
4
S=FTSEMIB(2:150)% data set for estimation of parameters
6 B=FTSEMIB(152:300)% backtesting dataset
S_0=FTSEMIB(151)
8 n=1000%100
k=25%149 %target number of VaR estimations=lenght backtesting dataset
10 a=0.05

12 hist(S)
%% Parameter estimation
14
parmhat = lognfit(S)
16

18 %% MC
%initialyze space
20 S_s_1=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_2=lognrnd(parmhat(1),parmhat(2),1,n)
22 S_s_3=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_4=lognrnd(parmhat(1),parmhat(2),1,n)
24 S_s_5=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_6=lognrnd(parmhat(1),parmhat(2),1,n)
26 S_s_7=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_8=lognrnd(parmhat(1),parmhat(2),1,n)
28 S_s_9=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_10=lognrnd(parmhat(1),parmhat(2),1,n)
30 S_s_11=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_12=lognrnd(parmhat(1),parmhat(2),1,n)
32 S_s_13=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_14=lognrnd(parmhat(1),parmhat(2),1,n)
34 S_s_15=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_16=lognrnd(parmhat(1),parmhat(2),1,n)
36 S_s_17=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_18=lognrnd(parmhat(1),parmhat(2),1,n)
38 S_s_19=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_20=lognrnd(parmhat(1),parmhat(2),1,n)
40 S_s_21=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_22=lognrnd(parmhat(1),parmhat(2),1,n)
42 S_s_23=lognrnd(parmhat(1),parmhat(2),1,n)
S_s_24=lognrnd(parmhat(1),parmhat(2),1,n)
44 S_s_25=lognrnd(parmhat(1),parmhat(2),1,n)

46
```

```
%% P&L
%initialyze space
PL_1=S_s_1-S_0
PL_2=S_s_2-S_0
PL_3=S_s_3-S_0
PL_4=S_s_4-S_0
PL_5=S_s_5-S_0
PL_6=S_s_6-S_0
PL_7=S_s_7-S_0
PL_8=S_s_8-S_0
PL_9=S_s_9-S_0
PL_10=S_s_10-S_0
PL_11=S_s_11-S_0
PL_12=S_s_12-S_0
PL_13=S_s_13-S_0
PL_14=S_s_14-S_0
PL_15=S_s_15-S_0
PL_16=S_s_16-S_0
PL_17=S_s_17-S_0
PL_18=S_s_18-S_0
PL_19=S_s_19-S_0
PL_20=S_s_20-S_0
PL_21=S_s_21-S_0
PL_22=S_s_22-S_0
PL_23=S_s_23-S_0
PL_24=S_s_24-S_0
PL_25=S_s_25-S_0

%% VaR

PL_ord_1=sort(PL_1)
PL_ord_2=sort(PL_2)
PL_ord_3=sort(PL_3)
PL_ord_4=sort(PL_4)
PL_ord_5=sort(PL_5)
PL_ord_6=sort(PL_6)
PL_ord_7=sort(PL_7)
PL_ord_8=sort(PL_8)
PL_ord_9=sort(PL_9)
PL_ord_10=sort(PL_10)
PL_ord_11=sort(PL_11)
PL_ord_12=sort(PL_12)
PL_ord_13=sort(PL_13)
PL_ord_14=sort(PL_14)
PL_ord_15=sort(PL_15)
PL_ord_16=sort(PL_16)
PL_ord_17=sort(PL_17)
```

```
PL_ord_18=sort(PL_18)
PL_ord_19=sort(PL_19)
PL_ord_20=sort(PL_20)
PL_ord_21=sort(PL_21)
PL_ord_22=sort(PL_22)
PL_ord_23=sort(PL_23)
PL_ord_24=sort(PL_24)
PL_ord_25=sort(PL_25)


VaR_1=PL_ord_1(n*a)
VaR_2=PL_ord_2(n*a)
VaR_3=PL_ord_3(n*a)
VaR_4=PL_ord_4(n*a)
VaR_5=PL_ord_5(n*a)
VaR_6=PL_ord_6(n*a)
VaR_7=PL_ord_7(n*a)
VaR_8=PL_ord_8(n*a)
VaR_9=PL_ord_9(n*a)
VaR_10=PL_ord_10(n*a)
VaR_11=PL_ord_11(n*a)
VaR_12=PL_ord_12(n*a)
VaR_13=PL_ord_13(n*a)
VaR_14=PL_ord_14(n*a)
VaR_15=PL_ord_15(n*a)
VaR_16=PL_ord_16(n*a)
VaR_17=PL_ord_17(n*a)
VaR_18=PL_ord_18(n*a)
VaR_19=PL_ord_19(n*a)
VaR_20=PL_ord_20(n*a)
VaR_21=PL_ord_21(n*a)
VaR_22=PL_ord_22(n*a)
VaR_23=PL_ord_23(n*a)
VaR_24=PL_ord_24(n*a)
VaR_25=PL_ord_25(n*a)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,V

%% Backtesting

PL_B=B(65:89)-S_0


figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
```

```
hold off;

%% Quality checks for the model
%compute the difference between VaR estimate and actual losses
error_MC=zeros(1,25)

for i=1:25
error_MC(i)=Var_sim(i)-PL_B(i)
end

% number of times VaR fails to cover losses
failure_MC=zeros(1,25)
for i=1:25
if error_MC(i)>0
failure_MC(i)=1
end
end

x_MC=sum(failure_MC)
p=0.05 %level of confidence
N=25 %number of estimates

%% Binomial test, compare actual and expected number of vailures

exp_val_bin=p*N
test_bin_MC=(x_MC-N*p)/(sqrt(N*p*(1-p)))
crit_val_bin=norminv(1-p) %test passed
%% Kupiec proportion of failures test

LR_pof_MC=-2*log(((1-p)^(N-x_MC)*p^x_MC)/((1-x_MC/N)^(N-x_MC)*(x_MC/N)^x_MC))
crit=chi2inv((1-p),1) %test passed

%% Kupiec time untill first failure test
%number of days before first rejection
k_MC=1

LR_TUFF_MC=-2*log((p*(1-p)^(k_MC-1))/((1/k_MC)*(1-(1/k_MC))^(k_MC-1)))
crit_TUFF=chi2inv((1-p),1)
% test passed

%% Christoffersen Interval forecast test

n_00_MC=15
n_10_MC=1
n_01_MC=0
n_11_MC=7

p_0_MC=n_01_MC/(n_00_MC+n_01_MC)
```

```
p_1_MC=n_11_MC/(n_10_MC+n_11_MC)
p_MC=(n_01_MC+n_11_MC)/(n_00_MC+n_01_MC+n_11_MC)

LRCCI_MC=-2*log(((1-p_MC)^(n_00_MC+n_10_MC)*p_MC^(n_01_MC+n_11_MC))/((1-p_0_MC
%check passed
LRCC_MC=LRCCI_MC+LR_pof_MC
crit2=chi2inv((1-p),2)
%check passed

%% Mixed Kupiec test, not working

days_MC=[0,0]

sumTB_MC=zeros(1,2)

for i=1:2
sumTB_MC=log((p*(1-p)^(days_MC(i)-1))/((1/days_MC(i))*(1-1/days_MC(i))^(days_M
end

LRTBFI_MC=-2*sum(sumTB_MC)
crit_TBFI_MC=chi2inv((1-p),2)
```

```
%% Montecarlo VaR Geometric Brownian motion

%% Initialization

S=FTSEMIB(2:150)% data set for estimation of parameters
B=FTSEMIB(152:300)% backtesting dataset
S_0=FTSEMIB(151)
n=1000%100
k=25%149 %target number of VaR estimations=lenght backtesting dataset
a=0.05

hist(S)
%% Parameter estimation

m=mean(S)
s=std(S)

%% MC
%initialyze space
S_s_1=S(1)*ones(1,n)
S_s_2=S(1)*ones(1,n)
S_s_3=S(1)*ones(1,n)
S_s_4=S(1)*ones(1,n)
S_s_5=S(1)*ones(1,n)
S_s_6=S(1)*ones(1,n)
S_s_7=S(1)*ones(1,n)
```

```
S_s_8=S(1)*ones(1,n)
S_s_9=S(1)*ones(1,n)
S_s_10=S(1)*ones(1,n)
S_s_11=S(1)*ones(1,n)
S_s_12=S(1)*ones(1,n)
S_s_13=S(1)*ones(1,n)
S_s_14=S(1)*ones(1,n)
S_s_15=S(1)*ones(1,n)
S_s_16=S(1)*ones(1,n)
S_s_17=S(1)*ones(1,n)
S_s_18=S(1)*ones(1,n)
S_s_19=S(1)*ones(1,n)
S_s_20=S(1)*ones(1,n)
S_s_21=S(1)*ones(1,n)
S_s_22=S(1)*ones(1,n)
S_s_23=S(1)*ones(1,n)
S_s_24=S(1)*ones(1,n)
S_s_25=S(1)*ones(1,n)

% generation of simulated path
for i=2:n
S_s_1(i)=S_s_1(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end

for i=2:n
S_s_2(i)=S_s_2(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_3(i)=S_s_3(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_4(i)=S_s_4(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_5(i)=S_s_5(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_6(i)=S_s_6(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_7(i)=S_s_7(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_8(i)=S_s_8(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_9(i)=S_s_9(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
```

```
for i=2:n
S_s_10(i)=S_s_10(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_11(i)=S_s_11(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_12(i)=S_s_12(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_13(i)=S_s_13(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_14(i)=S_s_14(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_15(i)=S_s_15(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_16(i)=S_s_16(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_17(i)=S_s_17(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_18(i)=S_s_18(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_19(i)=S_s_19(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_20(i)=S_s_20(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_21(i)=S_s_21(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_22(i)=S_s_22(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_23(i)=S_s_23(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_24(i)=S_s_24(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
for i=2:n
S_s_25(i)=S_s_25(i-1)+(randn(1,1)*s*sqrt(1/250)+m*1/250)
end
```

```
%% P&L
%initialyze space
PL_1=zeros(1,n)
PL_2=zeros(1,n)
PL_3=zeros(1,n)
PL_4=zeros(1,n)
PL_5=zeros(1,n)
PL_6=zeros(1,n)
PL_7=zeros(1,n)
PL_8=zeros(1,n)
PL_9=zeros(1,n)
PL_10=zeros(1,n)
PL_11=zeros(1,n)
PL_12=zeros(1,n)
PL_13=zeros(1,n)
PL_14=zeros(1,n)
PL_15=zeros(1,n)
PL_16=zeros(1,n)
PL_17=zeros(1,n)
PL_18=zeros(1,n)
PL_19=zeros(1,n)
PL_20=zeros(1,n)
PL_21=zeros(1,n)
PL_22=zeros(1,n)
PL_23=zeros(1,n)
PL_24=zeros(1,n)
PL_25=zeros(1,n)

for i=1:n
PL_1(i)=S_s_1(i)-S_0
end
for i=1:n
PL_2(i)=S_s_2(i)-S_0
end
for i=1:n
PL_3(i)=S_s_3(i)-S_0
end
for i=1:n
PL_4(i)=S_s_4(i)-S_0
end
for i=1:n
PL_5(i)=S_s_5(i)-S_0
end
for i=1:n
PL_6(i)=S_s_6(i)-S_0
end
for i=1:n
```

```
PL_7(i)=S_s_7(i)-S_0
end
for i=1:n
PL_8(i)=S_s_8(i)-S_0
end
for i=1:n
PL_9(i)=S_s_9(i)-S_0
end
for i=1:n
PL_10(i)=S_s_10(i)-S_0
end
for i=1:n
PL_11(i)=S_s_11(i)-S_0
end
for i=1:n
PL_12(i)=S_s_12(i)-S_0
end
for i=1:n
PL_13(i)=S_s_13(i)-S_0
end
for i=1:n
PL_14(i)=S_s_14(i)-S_0
end
for i=1:n
PL_15(i)=S_s_15(i)-S_0
end
for i=1:n
PL_16(i)=S_s_16(i)-S_0
end
for i=1:n
PL_17(i)=S_s_17(i)-S_0
end
for i=1:n
PL_18(i)=S_s_18(i)-S_0
end
for i=1:n
PL_19(i)=S_s_19(i)-S_0
end
for i=1:n
PL_20(i)=S_s_20(i)-S_0
end
for i=1:n
PL_21(i)=S_s_21(i)-S_0
end
for i=1:n
PL_22(i)=S_s_22(i)-S_0
end
for i=1:n
```

```
PL_23(i)=S_s_23(i)-S_0
end
for i=1:n
PL_24(i)=S_s_24(i)-S_0
end
for i=1:n
PL_25(i)=S_s_25(i)-S_0
end

%% VaR

PL_ord_1=sort(PL_1)
PL_ord_2=sort(PL_2)
PL_ord_3=sort(PL_3)
PL_ord_4=sort(PL_4)
PL_ord_5=sort(PL_5)
PL_ord_6=sort(PL_6)
PL_ord_7=sort(PL_7)
PL_ord_8=sort(PL_8)
PL_ord_9=sort(PL_9)
PL_ord_10=sort(PL_10)
PL_ord_11=sort(PL_11)
PL_ord_12=sort(PL_12)
PL_ord_13=sort(PL_13)
PL_ord_14=sort(PL_14)
PL_ord_15=sort(PL_15)
PL_ord_16=sort(PL_16)
PL_ord_17=sort(PL_17)
PL_ord_18=sort(PL_18)
PL_ord_19=sort(PL_19)
PL_ord_20=sort(PL_20)
PL_ord_21=sort(PL_21)
PL_ord_22=sort(PL_22)
PL_ord_23=sort(PL_23)
PL_ord_24=sort(PL_24)
PL_ord_25=sort(PL_25)


VaR_1=PL_ord_1(n*a)
VaR_2=PL_ord_2(n*a)
VaR_3=PL_ord_3(n*a)
VaR_4=PL_ord_4(n*a)
VaR_5=PL_ord_5(n*a)
VaR_6=PL_ord_6(n*a)
VaR_7=PL_ord_7(n*a)
VaR_8=PL_ord_8(n*a)
VaR_9=PL_ord_9(n*a)
VaR_10=PL_ord_10(n*a)
```

```
VaR_11=PL_ord_11(n*a)
VaR_12=PL_ord_12(n*a)
VaR_13=PL_ord_13(n*a)
VaR_14=PL_ord_14(n*a)
VaR_15=PL_ord_15(n*a)
VaR_16=PL_ord_16(n*a)
VaR_17=PL_ord_17(n*a)
VaR_18=PL_ord_18(n*a)
VaR_19=PL_ord_19(n*a)
VaR_20=PL_ord_20(n*a)
VaR_21=PL_ord_21(n*a)
VaR_22=PL_ord_22(n*a)
VaR_23=PL_ord_23(n*a)
VaR_24=PL_ord_24(n*a)
VaR_25=PL_ord_25(n*a)


Var_sim=[VaR_1,VaR_2,VaR_3,VaR_4,VaR_5,VaR_6,VaR_7,VaR_8,VaR_9,VaR_10,VaR_11,V

%% Backtesting

PL_B=B(65:89)-S_0


figure;
plot( Var_sim, "r");
hold on;
plot(PL_B, "g");
hold off;

%% Quality checks for the model
%compute the difference between VaR estimate and actual losses
error_MC=zeros(1,25)

for i=1:25
error_MC(i)=Var_sim(i)-PL_B(i)
end

% number of times VaR fails to cover losses
failure_MC=zeros(1,25)
for i=1:25
if error_MC(i)>0
failure_MC(i)=1
end
end

x_MC=sum(failure_MC)
p=0.05 %level of confidence
```

```
N=25 %number of estimates

%% Binomial test, compare actual and expected number of vailures

exp_val_bin=p*N
test_bin_MC=(x_MC-N*p)/(sqrt(N*p*(1-p)))
crit_val_bin=norminv(1-p) %test failed
%% Kupiec proportion of failures test

LR_pof_MC=-2*log(((1-p)^(N-x_MC)*p^x_MC)/((1-x_MC/N)^(N-x_MC)*(x_MC/N)^x_MC))
crit=chi2inv((1-p),1) %test failed

%% Kupiec time untill first failure test
%number of days before first rejection
k_MC=3

LR_TUFF_MC=-2*log((p*(1-p)^(k_MC-1))/((1/k_MC)*(1-(1/k_MC))^(k_MC-1)))
crit_TUFF=chi2inv((1-p),1)
% test passed

%% Christoffersen Interval forecast test

n_00_MC=21
n_10_MC=2
n_01_MC=2
n_11_MC=0

p_0_MC=n_01_MC/(n_00_MC+n_01_MC)
p_1_MC=n_11_MC/(n_10_MC+n_11_MC)
p_MC=(n_01_MC+n_11_MC)/(n_00_MC+n_01_MC+n_11_MC)

LRCCI_MC=-2*log(((1-p_MC)^(n_00_MC+n_10_MC)*p_MC^(n_01_MC+n_11_MC))/((1-p_0_MC)^(n
%check passed
LRCC_MC=LRCCI_MC+LR_pof_MC
crit2=chi2inv((1-p),2)
%check passed

%% Mixed Kupiec test, not working

days_MC=[0,0]

sumTB_MC=zeros(1,2)

for i=1:2
sumTB_MC=log((p*(1-p)^(days_MC(i)-1))/((1/days_MC(i))*(1-1/days_MC(i))^(days_MC(i)
end

LRTBFI_MC=-2*sum(sumTB_MC)
```

```
crit_TBFI_MC=chi2inv((1-p),2)
```

Bibliography

[1] Antonelli, Sabrina & Iovino, Maria. (2003). Optimization of Monte Carlo Procedures for Value at Risk Estimates. Economic Notes. 31. 59 - 78. 10.1111/1468-0300.00072.

[2] Asmussen, S., & Glynn, P.W. (2007). Stochastic simulation - algorithms and analysis. Stochastic modeling and applied probability.

[3] Bansal, A., Kauffman, R. J., Mark, R. M. & Peters, E. (1993). Financial risk and financial risk management technology (RMT): Issues and advances.. Information & Management, 24, 267-281.

[4] Basel Committee on Banking Supervision. (2006). Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version, Bank for International Settlements.

[5] Batten, J.A. and Fetherston, T.A., eds. 2002. Financial risk and financial risk management. Jai Press, Amsterdam.

[6] Bauer, W. (1958). The Monte Carlo Method. Journal of the Society for Industrial and Applied Mathematics, 6(4), 438-451.

[7] Beder, Tanya. (1995). VAR: Seductive but dangerous. Financial Analysts Journal - FINANC ANAL J. 51. 12-24. 10.2469/faj.v51.n5.1932.

[8] Beichl, I. & Sullivan, F. (2000). The Metropolis Algorithm. In Computing in Science and Engg. , Vol. 2 (pp. 65–69). IEEE Educational Activities Department.

[9] Bernard, C., Rüschendorf, L. & Vanduffel, S. (2015). Value-at-Risk Bounds with Variance Constraints. Journal of Risk and Insurance, 84, 923-959. doi: 10.2139/ssrn.2342068

[10] Better, M., Glover, F. W., Kochenberger, G. A. & Wang, H. (2008). Simulation Optimization: Applications in Risk Management.. International Journal of Information Technology and Decision Making, 7, 571-587.

[11] Beygelzimer, A. & Rish, I. (2003). Approximability of Probability Distributions.. In S. Thrun, L. K. Saul & B. Schölkopf (eds.), NIPS (p./pp. 377-384), : MIT Press. ISBN: 0-262-20152-6

[12] Binder, K. (1992). The Monte Carlo Method in Condensed Matter Physics. Berlin: Springer Verlag.

[13] Botev Z. and L'Ecuyer P., "Simulation from the Normal Distribution Truncated to an Interval in the Tail," Proceedings of ValueTools 2016.

[14] Briys, E., ed. 1998. Options, futures, and exotic derivatives. Wiley, Chichester.

[15] Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. 2011. Handbook of Markov Chain Monte Carlo. CRC press.

[16] Brooks, S. (1998). Markov chain Monte Carlo method and its application. Journal of the Royal Statistical Society: Series D (The Statistician), 47, 69–100. doi: 10.1111/1467-9884.00117

[17] Câmpeanu, C. (1994). Random Numbers.. Sci. Ann. Cuza Univ., 3, 53-64.

[18] Ling Chen, Tze Leung Lai, and Tiong Wee Lim. 2011. Option prices and pricing theory: combining financial mathematics with statistical modeling. WIREs Comput. Stat. 3, 6 (November 2011), 566-576. DOI: https://doi.org/10.1002/wics.186

[19] Chorafas D.N. (2000) Model Risk and the Control of Eigenmodels by the Supervisors. In: New Regulation of the Financial Industry. Palgrave Macmillan, London

[20] Christoffersen, P. (1998) Evaluating Interval Forecasts. International Economic Review, 39, 841-862.

[21] Christoffersen, Peter and Pelletier, Denis, Backtesting Value-at-Risk: A Duration-Based Approach (January 31, 2003).

[22] Conte, T. M. (2005). Insight, not (random) numbers.. ISPASS (p./pp. 101), : IEEE Computer Society. ISBN: 0-7803-8965-4

[23] Corelli, A., 2016. "Analytical Corporate Finance," Springer Texts in Business and Economics, Springer, number 978-3-319-39549-4.

[24] Couture R. and L'Ecuyer P., "Linear Recurrences with Carry a Uniform Random Number Generators", Proceedings of the 1995 Winter Simulation Conference, Dec 1995, 263–267.

[25] Crandall, R. E. & Bailey, D. H. (2002). Random Generators and Normal Numbers.. Experimental Mathematics, 11, 527-546.

[26] Dash, J. (2012). Stressed Value-at-Risk.. CIFEr (p./pp. 1), : IEEE. ISBN: 978-1-4673-1802-0

[27] Das, S., ed. 1998. Risk management and financial derivatives. McGraw Hill, New York, NY.

[28] D'Ecclesia, R. L. (2005). Financial modelling and risk management.. European Journal of Operational Research, 163, 1-4.

[29] Diggelmann, P. B. (1999). Value at risk. Zürich: Versus. ISBN: 3908143691

[30] Dowd, Kevin. 1998. Beyond value at risk: the new science of risk management. Chichester: Wiley.

[31] Draghi, M., Giavazzi, F.,, Merton, R. C. (2003). Transparency, risk management and international financial fragility. London: CEPR. ISBN: 1898128685

[32] Embrechts, P., McNeil, A. & Straumann, D. (2002). Risk management: value at risk and beyond. In M. Dempster (ed.), . Cambridge University Press.

[33] Fu F., Niederreiter H., Özbudak F., Joint linear complexity of multisequences consisting of linear recurring sequences. Cryptography and Communications 1(1): 3-29 (2009).

[34] Gaglianone, Wagner & Lima, Luiz & Linton, Oliver & Smith, Daniel. (2008). Evaluating Value-at-Risk Models via Quantile Regression. Journal of Business & Economic Statistics. 29. 150-160. 10.2307/25800786.

[35] Geçkinli, N. C. & Apohan, M. A. (2001). Power spectrum tests of random numbers.. Signal Processing, 81, 1389-1405.

[36] Gentle, J.E. 2004. Random Number Generation and Monte Carlo Methods. Springer.

[37] Giot, Pierre & Laurent, Sébastien. (2003). Value-at-Risk for Long and Short Trading Positions. Journal of Applied Econometrics. 18. 641 - 663. 10.1002/jae.710.

[38] Glasserman, Paul & Heidelberger, Philip & Shahabuddin, Perwez. (2000). Efficient Monte Carlo Methods for Value-at-Risk. Master. Risk. 2.

[39] Glasserman, P. 2004. Monte Carlo methods in financial engineering. Springer, New York.

[40] Glasserman, P., Heidelberger, P. & Shahabuddin, P. (2000). Value-at-risk with heavy-tailed risk factors.. CIFEr (p./pp. 58-61), : IEEE. ISBN: 0-7803-6429-5

[41] Göb, R. (2011). Estimating value at risk and conditional value at risk for count variables.. Quality and Reliability Eng. Int., 27, 659-672.

[42] Goodman, J. & Irwin, J. (2006). Special random numbers: Beyond the illusion of control. Organizational Behavior and Human Decision Processes, 99, 161–174. doi: 10.1016/j.obhdp.2005.08.004

[43] Graham, Carl & Talay, Denis. (2013). Stochastic Simulation and Monte Carlo Methods. Mathematical Foundations of Stochastic Simulation. 10.1007/978-3-642-39363-1.

[44] Gutmann, P. (1998). Software Generation of Practically Strong Random Numbers.. In A. D. Rubin (ed.), USENIX Security Symposium, : USENIX Association.

[45] Herwartz, H. & Waichman, I. Comput Stat .2010. 25: 725. https://doi.org/10.1007/s00180-010-0194-4

[46] Hofmann, M. (2014). Risk disclosure, risk perception and the firm's market risk. Unpublished doctoral dissertation , Universität für Wirtschaft und Recht Wiesbaden .

[47] Hong, L. J. & Liu, G. (2011). Monte Carlo estimation of value-at-risk, conditional value-at-risk and their sensitivities.. In S. Jain, R. R. C. Jr., J. Himmelspach, K. P. White & M. C. Fu (eds.), Winter Simulation Conference (p./pp. 95-107), : IEEE. ISBN: 978-1-4577-2107-6

[48] Hong, L. J., Hu, Z. & Liu, G. (2014). Monte Carlo Methods for Value-at-Risk and Conditional Value-at-Risk: A Review.. ACM Trans. Model. Comput. Simul., 24, 22:1-22:37.

[49] Hong, L. J., Hu, Z. & Zhang, L. (2014). Conditional Value-at-Risk Approximation to Value-at-Risk Constrained Programs: A Remedy via Monte Carlo.. INFORMS Journal on Computing, 26, 385-400.

[50] Hong, L. J., Hu, Z. & Liu, G. (2014). Monte Carlo Methods for Value-at-Risk and Conditional Value-at-Risk: A Review.. ACM Trans. Model. Comput. Simul., 24, 22:1-22:37.

[51] Hong, L. J. & Liu, G. (2011). Monte Carlo estimation of value-at-risk, conditional value-at-risk and their sensitivities.. In S. Jain, R. R. C. Jr., J. Himmelspach, K. P. White & M. C. Fu (eds.), Winter Simulation Conference (p./pp. 95-107), : IEEE. ISBN: 978-1-4577-2107-6

[52] Hong L., Hu Z., and Liu G. 2014. Monte Carlo Methods for Value-at-Risk and Conditional Value-at-Risk: A Review. ACM Trans. Model. Comput. Simul. 24, 4, Article 22 (November 2014), 37 pages. DOI=http://dx.doi.org/10.1145/2661631

[53] Horcher, K. A. (2005). Essentials of financial risk management. Hoboken, NJ: Wiley. ISBN: 0471706167

[54] Hull, J.C. 2006. Options, futures, and other derivatives. Pearson Prentice Hall, Upper Saddle River, NJ.

[55] Houston, D. X., Mackulak, G. T. & Collofello, J. S. (2001). Stochastic simulation of risk factor potential effects for software development risk management.. Journal of Systems and Software, 59, 247-257.

[56] Jacoboni, C., Lugli, P. (1989). The Monte Carlo Method for Semiconductor Device Simulation. Wien: Springer.

[57] James, F., Hoogland, J. & Kleiss, R. (1996). Multidimensional sampling for simulation and integration: measures, discrepancies, and quasi-random numbers. Computer Physics Communications, 99, 180–220. doi: 10.1016/s0010-4655(96)00108-7

[58] Jampani R.,Xu F., Wu M., Perez L.,Jermaine C., and Haas P. 2008. MCDB: a monte carlo approach to managing uncertain data. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08). ACM, New York, NY, USA, 687-700. DOI: https://doi.org/10.1145/1376616.1376686

[59] Jin, X. & Zhang, A. X. (2006). Reclaiming Quasi - Monte Carlo Efficiency in Portfolio Value-at-Risk Simulation Through Fourier Transform.. Management Science, 52, 925-938.

[60] Jorion, P. (2001) Value at Risk: The New Benchmark for Managing Financial Risk. 2nd Edition, McGraw-Hill, United States of America

[61] Kashima, H. (2006). Risk-Sensitive Learning via Expected Shortfall Minimization.. In J. Ghosh, D. Lambert, D. B. Skillicorn & J. Srivastava (eds.), SDM (p./pp. 529-533), : SIAM. ISBN: 978-1-61197-276-4

[62] Kilian, L., Manganelli, S. (2003). The central bank as a risk manager. Frankfurt am Main: European Central Bank.

[63] Kneusel, R. T. (2018). Random Numbers and Computers. Springer. ISBN: 978-3-319-77696-5

[64] Kou, S., Peng, X. H. & Heyde, C. C. (2013). External Risk Measures and Basel Accords.. Math. Oper. Res., 38, 393-417.

[65] Kritzer P., Niederreiter H.,Pillichshammer F.,Winterhof A. Uniform Distribution and Quasi-Monte Carlo Methods - Discrepancy, Integration and Applications. Radon Series on Computational and Applied Mathematics 15, De Gruyter 2014, ISBN 978-3-11-031793-0.

[66] Kroese D.P. and Rubinstein R.Y. 2012. Monte Carlo methods. WIREs Comput. Stat. 4, 1 (January 2012), 48-58. DOI: https://doi.org/10.1002/wics.194.

[67] Kroese, D.P., Taimre, T., & Botev, Z.I. (2011). Handbook of Monte Carlo Methods.

[68] Kupiec, J. (1989). Probabilistic Models of Short and Long Distance Word Dependencies in Running Text.. HLT (1), : ACL. ISBN: 978-1-55860-073-7.

[69] Kupiec, Paul, Techniques for Verifying the Accuracy of Risk Measurement Models. FEDS Paper Number: 95-24. Available at SSRN: https://ssrn.com/abstract=6697.

[70] Landau D. and Binder K. 2005. A Guide to Monte Carlo Simulations in Statistical Physics. Cambridge University Press, New York, NY, USA.

[71] Landskroner Y., Ruthenberg D., Zaken D. (1999) Market Risks—the Amendment to the Basel Capital Accord and Internal Model Approach: The Israeli Case. In: Galai D., Ruthenberg D., Sarnat M., Schreiber B.Z. (eds) Risk Management and Regulation in Banking. Springer, Boston, MA.

[72] Lange, T., Lubicz, D. & Weigl, A. (2005). Random Numbers Generation and Testing.. In H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen & F. Vercauteren (ed.), Handbook of Elliptic and Hyperelliptic Curve Cryptography (pp. 715-735) . Chapman and Hall/CRC . ISBN: 978-1-4200-3498-1.

[73] L'Ecuyer P. and Blouin F., "Linear Conguential Generators of Order $k > 1$", Proceedings of the 1988 Winter Simulation Conference, Dec. 1988, 432–439.

[74] L'Ecuyer P., "A Tutorial on Uniform Variate Generation", 1989 Winter Simulation Conference Proceedings, Dec. 1989, 40–49.

[75] L'Ecuyer P., "Random Numbers for Simulation", Communications of the ACM, 33 (1990), 85–98.

[76] L'Ecuyer P. and Tezuka S., "Structural Properties for Two Classes of Combined Generators", Mathematics of Computation, 57, (1991), 735–746.

[77] L'Ecuyer P., "Testing Random Number Generators", Proceedings of the 1992 Winter Simulation Conference, dec 1992, 305–313.

[78] L'Ecuyer P., Blouin F., and Couture R., "A Search for Good Multiple Recursive Generators", ACM Trans. on Modeling and Computer Simulation, 3, 2 (1993), 87–98.

[79] L'Ecuyer P., Giroux N., and Glynn P., "Stochastic Optimization by Simulation: Numerical Experiments with the M/M/1 Queue in Steady-State", Management Science, 40, 10 (Oct. 1994), 1245–1261.

[80] L'Ecuyer P., "Efficiency Improvement and Variance Reduction", Proceedings of the 1994 Winter Simulation Conference, Dec. 1994, 122–132.

[81] L'Ecuyer P., "Recent Advances in Uniform Random Number Generation", Proceedings of the 1994 Winter Simulation Conference, Dec. 1994, 122–132.

[82] L'Ecuyer P., "Uniform Random Number Generation", Annals of Operations Research, 53 (1994), 77–120.

[83] L'Ecuyer P., "Combined Multiple Recursive Generators", Operations Research, 44, 5 (1996), 816–822.

[84] L'Ecuyer P., "Random Number Generators", in Encyclopedia of Operations Research and Management Science, S. I. Gass and C. M. Harris Eds., Kluwer Academic Publishers, 1996, 571–578.

[85] L'Ecuyer P., "History of Uniform Random Number Generation," Proceedings of the 2017 Winter Simulation Conference, invited paper, 2017, 202-230.

[86] L'Ecuyer P., Munger D., Oreshkin B., and Simard R., "Random Numbers for Parallel Computers: Requirements and Methods," Mathematics and Computers in Simulation, 135, (2017), 3-17.

[87] L'Ecuyer P., "Discussion of "Sequential Quasi-Monte-Carlo Sampling" by M. Gerber and N. Chopin," Journal of the Royal Statistical Society, Series B, 77, part 3 (2015), 565–566.

[88] L'Ecuyer P. 1990. Random numbers for simulation. Commun. ACM 33, 10 (October 1990), 85-97. DOI: https://doi.org/10.1145/84537.84555.

[89] Lemieux, Christiane. (2009). Monte Carlo and Quasi-Monte Carlo Sampling. 10.1007/978-0-387-78165-5.

[90] Lin, J., Jia, S. & Deng, J. (2017). Smart risk management with financial big data.. SII (p./pp. 60-65), : IEEE. ISBN: 978-1-5386-2263-6

[91] Linsmeier, T.J. and Pearson, N.D. (2000) Value at Risk. Financial Analysts Journal, 56, 47-67.

[92] Linsmeier, Thomas J. & Pearson, Neil D., 1996. "Risk measurement: an introduction to value at risk," ACE Reports 14796, University of Illinois at Urbana-Champaign, Department of Agricultural and Consumer Economics.

[93] Liu, Y. & Ralescu, D. A. (2017). Value-at-risk in uncertain random risk analysis.. Inf. Sci., 391, 1-8.

[94] Liyanage, D. N. S. S., Fernando, G. V. M. P. A., Arachchi, D. D. M. M., Karunathilaka, R. D. D. T. & Perera, A. S. (2017). Utilizing Intel Advanced Vector Extensions for Monte Carlo Simulation based Value at Risk Computation.. In P. Koumoutsakos, M. Lees, V. V. Krzhizhanovskaya, J. J. Dongarra & P. M. A. Sloot (eds.), ICCS (p./pp. 626-634), : Elsevier.

[95] Luizi, Paulo & Cruz, Frederico & van de Graaf, Jeroen. (2010). Assessing the Quality of Pseudo-Random Number Generators. Computational Economics. 36. 57-67. 10.1007/s10614-010-9210-6.

[96] Manganelli, S., Ceci, V.,, Vecchiato, W. (2002). Sensitivity analysis of volatility. Frankfurt am Main: European Central Bank.

[97] Manganelli, Simone and Engle, Robert F., Value at Risk Models in Finance (August 2001). ECB Working Paper No. 75.

[98] Marsaglia, G. Generating Exponential Random Variables. Ann. Math. Statist. 32 (1961), no. 3, 899–900. doi:10.1214/aoms/1177704984. https://projecteuclid.org/euclid.aoms/1177704984

[99] Marshall, Christopher and Siegel, Michael, Value-at-Risk: Implementing a Risk Measurement Standard. 1996. 96-47.

[100] Martino, L. & Elvira, V. (2017). Metropolis Sampling (cite arxiv:1704.04629Comment: Wiley StatsRef-Statistics Reference Online, 2017)

[101] Metropolis N. & Ulam S. (1949). The Monte-Carlo method. J. Am. Stat. Ass., 44, 335–341.

[102] Morningstar, C. (2007). The Monte Carlo method in quantum field theory.

[103] Niederreiter H., The independence of two randomness properties of sequences over finite fields. J. Complexity 28(2): 154-161 (2012).

[104] Niederreiter H., Probability and computing: randomized algorithms and probabilistic analysis. Math. Comput. 75(255) (2006).

[105] Niederreiter H., Arne Winterhof, On the Distribution of Some New Explicit Nonlinear Congruential Pseudorandom Numbers. SETA 2004: 266-274.

[106] Mausser, H. & Rosen, D. (1999). Beyond VaR: parametric and simulation-based risk management tools.. CIFEr (p./pp. 159-162), : IEEE. ISBN: 0-7803-5663-2

[107] Oppong, Stephen & Asamoah, Dominic & Oppong, Emmanuel. (2016). VALUE AT RISK: HISTORICAL SIMULATION OR MONTE CARLO SIMULATION.

[108] Oran, E., Oh, C. & Cybyk, B. (1998). DIRECT SIMULATION MONTE CARLO: Recent Advances and Applications. Annu. Rev. Fluid Mech., 30, 403-441.

[109] Perignon, Christophe and Deng, Zi Yin and Wang, Zhi Jun, Do Banks Overstate Their Value-at-Risk? (May 29, 2007).

[110] Qian, L. (2007). Simulation Techniques in Financial Risk Management.. Technometrics, 49, 222.

[111] Robert C.P. and Casella G. 2005. Monte Carlo Statistical Methods (Springer Texts in Statistics). Springer-Verlag, Berlin, Heidelberg.

[112] Robert C.P. and Casella G. 2009. Introducing Monte Carlo Methods with R (Use R) (1st ed.). Springer-Verlag, Berlin, Heidelberg.

[113] Roberts, G. O.; Gelman, A.; Gilks, W. R. Weak convergence and optimal scaling of random walk Metropolis algorithms. Ann. Appl. Probab. 7 (1997), no. 1, 110–120. doi:10.1214/aoap/1034625254. https://projecteuclid.org/euclid.aoap/1034625254.

[114] Roccioletti, Simona. (2016). Backtesting Value at Risk and Expected Shortfall. 10.1007/978-3-658-11908-9.

[115] Ross S.M. 2006. Simulation, Fourth Edition. Academic Press, Inc., Orlando, FL, USA.

[116] Rubinstein R.Y. and Kroese D.P. 2016. Simulation and the Monte Carlo Method (3rd ed.). Wiley Publishing.

[117] Ruijter M.J.and Oosterlee C.W. 2016. Numerical Fourier method and second-order Taylor scheme for backward SDEs in finance. Appl. Numer. Math. 103, C (May 2016), 1-26. DOI: https://doi.org/10.1016/j.apnum.2015.12.003

[118] Schaumburg, J. (2013). Quantile methods for financial risk management. Unpublished doctoral dissertation , HU Berlin .

[119] Schwartz, Robert J.. (ed.) (1993). Advanced strategies in financial risk management. New York [u.a.]: New York Inst. of Finance. ISBN: 0130688835

[120] Shonkwiler R. and Mendivil F. 2009. Explorations in Monte Carlo Methods (1st ed.). Springer Publishing Company, Incorporated.

[121] Skiadopoulos, George & Lambadiaris, Greg & Papadopoulou, Louiza & Zoulis, Yiannis. (2003). VaR: History or Simulation?. RISK.

[122] Smith, R. T. (1993). Market risk and asset prices. Journal of Economic Dynamics and Control, 17, 555–569.

[123] Stockmal, F. (1964). Calculations with Pseudo-Random Numbers.. J. ACM, 11, 41-52.

[124] Suhobokov, Alexander. (2007). Application of Monte Carlo simulation methods in Risk Management. Journal of Business Economics and Management. 8. 165-168. 10.1080/16111699.2007.9636165.

[125] Teplytskyi, I. O. & Semerikov, S. O. (2018). Simulation using random numbers.. CoRR, abs/1809.05379.

[126] Tezuka S. and L'Ecuyer P., "Efficient and Portable Combined Tausworthe Random Number Generators", ACM Trans. on Modeling and Computer Simulation, 1, 2 (1991), 99–112

[127] Tezuka, S., Murata, H., Tanaka, S. & Yumae, S. (2005). Monte Carlo grid for financial risk management.. Future Generation Comp. Syst., 21, 811-821.

[128] Thomopoulos, Nick. (2013). Essentials of Monte Carlo simulation: Statistical methods for building simulation models. 10.1007/978-1-4614-6022-0.

[129] Ulam, S. (1960). A Collection of Mathematical Problems. New York, NY, USA: Interscience.

[130] Various authors (2017). Monte Carlo Simulation.. In S. Shekhar, H. Xiong & X. Zhou (ed.), Encyclopedia of GIS (pp. 1361) . Springer . ISBN: 978-3-319-17885-1.

[131] Various authors(2001). Proceedings of the International Conference "Managing Credit and Market Risk - New Techniques for New Sources of Risk" (30.2001,2) . In Andrea. Berardi (ed.).

[132] von Neumann, J. (1996). Mathematical Foundations of Quantum Mechanics. Princeton University Press. ISBN: 0691028931

[133] Warthen, I. & Arvind, K. (2005). Probabilistic simulation analysis of risk potential in cost management programs.. Winter Simulation Conference (p./pp. 48), : IEEE Computer Society. ISBN: 0-7803-9519-0

[134] Wichmann, B. A. & Hill, I. D. (2006). Generating good pseudo-random numbers. Computational Statistics & Data Analysis, 51, 1614–1622.

[135] Zhou, P. & Leung, H. K. N. (2012). A Stochastic Simulation Model for Risk Management Process.. In K. R. P. H. Leung & P. Muenchaisri (eds.), APSEC (p./pp. 737-742), : IEEE. ISBN: 978-0-7695-4922-4

Montecarlo and Value at Risk: empirical evidence from the Italian stock market

Martina Aquila

LUISS University — September 30, 2019

The main topic of this thesis is the Montecarlo method. A Montecarlo method is usually referred to as a technique that involves the generation of random numbers for the determination of the (approximate) solution of a deterministic problem. Montecarlo methods are then useful tools only in the case in which the actual solution of the problem is impossible or extremely difficult to find. The application of Montecarlo methods was, at a first point, limited to natural sciences such as physics, but now it includes many different disciplines such as biology, psychology, mathematics, statistics, economics, thermo-dynamics and medicine.

Montecarlo methods (that date back to the XVIII Century) are based on the generation of random processes or variables. Even if random number generation functions are today pre-built in most software, the definition of an efficient random number generator is non-trivial.

Usually, random number generators are defined for uniform random numbers and then some transformations are applied so to obtain a stream that obeys to the selected distribution.

The most used random number generators are multiple recursive. This implies that previous numbers are determinant of the next ones, in a fashion that appears to be random. A generic recursive generator is then in the form:

$$x_t = f(x_{t-1}, \dots, x_{t-k}) \tag{1}$$

Where f is the generation law. Since f is simply a function that links previous values to the current one, it is hard to accept the definition of this number generator as random. Every modern random number generator should, in fact, be called pseudo-random, since the application of a relationship which is entirely deterministic can never generate a random sequence. The generation law f has been defined in many different ways (from a simple linear function to a trigonometric one) and it usually includes modulo reductions.

Once a uniform stream of random number has been generated, $X \sim U(0,1)$, it is only necessary to apply some transformation to the stream X so to obtain pseudo-random numbers that follow the desired distribution. The most widely used transformation techniques are the inverse-transform, the acceptance-rejection, the ratio of uniform and the alias method. In 3.2, a list of 22 standard random variables (both continuous and discrete) is given, together with the most famous generation algorithms and their Matlab implementation. Efficiency considerations and optimal parameter choices are given too.

The description of random number generation techniques represents the description of a Montecarlo method.

One of the most central applications of the Montecarlo method to finance (and in particular to risk management) is related to the computation of Value at Risk (VaR).

Value at risk is an important measure to assess the level of exposure of a portfolio (or asset) to market risk. Market risk is formally defined by the Basel Committee for Banking supervision (2019) as the risk of losses that arise from movements in market prices. VaR is an absolute measure of market risk in the fact that it is represented by an absolute amount and not by a percentage. VaR can be defined as the absolute maximum loss in value of an asset or portfolio of (financial) assets over a certain horizon (t) and with a specified probability (α, confidence level). Similarly, VaR gives information about how much a portfolio is expected to lose in absolute terms with probability α and over the horizon t, assuming that the composition of the portfolio remains unchanged over that horizon. In order to compute VaR, it is necessary to have the distribution of P&L of the relevant portfolio. If this is available, in fact, given the confidence level $1-\alpha$ and the time horizon t, VaR is simply defined as the number such that the probability of incurring in higher losses is equal to α, or, similarly, the number such that the probability of incurring in lower losses is $1-\alpha$:

$$P(P\&L_t \leq VaR) = 1 - \alpha \tag{2}$$

The differences among VaR approaches lie in the computation of the P&L distribution. Literature is generally in agreement in enumerating three most common approaches to the computation of VaR: Variance-Covariance (VC) approach, Historical simulation (HS) and Montecarlo simulation (MC). All three methods can be extensively modified and improved so that there are indeed many different available models. A complete list of VaR models can be found at www.GloriaMundi.org.

In the VC approach, it is assumed that all components of the portfolio (we usually call them market factors) are normally distributed and that the relationship between the value of the portfolio and the components is linear. Under these two (rather restrictive) assumptions, the distribution of portfolio P&L is again Normal since the sum of Normal random variables is Normal. According to the VC approach, VaR is computed as:

$$VaR = -(\mu + z_{1-\alpha}\sigma) \tag{3}$$

Where μ and σ are, respectively, the mean and the standard deviation of the portfolio. They are computed based on the result that the sum of Normal random variables is again Normal. $z_{1-\alpha}$ is the $1-\alpha\%$ percentile of the Standard Normal distribution.

In the HS approach, no assumption on the statistical distribution of portfolio components is made. However, the critical assumption is that the behavior of portfolio components is completely determined by history. This means in technical terms that, for each market factor, an historical time series is selected and considered as emblematic for future evolutions. Given this set of historical observations for portfolio component i, $M_i = \{m_{i,1}, m_{i,2}, ..., m_{i,n+1}\}$, the rate of change must be computed as $r_{i,j} = \frac{m_{i,j+1}}{m_{i,j}} - 1$. Once a vector of rates is available for every component, those rates are applied to current values so to obtain a simulated historical-based evolution for each market factor. This procedure implies that in the HS approach, portfolio value is based on historical information but not equal to it. Once simulated paths for the components are available, it is only necessary to combine them so to obtain simulated portfolio values. Based on those values, the distribution of P&L is computed. VaR is then found as the $1-\alpha$ percentile observation in the ordered P&L distribution.

The MC is similar to HS in terms of steps to be performed but the assumptions are significantly distant. As in the HS case, possible future values for portfolio components must be generated. In this generation lie the critical differences between the two methods. In the MC a random number generator is used in order to produce a stream of values for each market factor. As specified above, when a random number generator is used, it is necessary to specify to which distribution the generated numbers will belong. The choice of the statistical distribution for each portfolio component is by far the most complex and critical step in MC implementation. If the distributional choice is unrealistic, then the generated random numbers are unreliable as the VaR estimate that results. In practical terms, historical time series of portfolio components are used as a reference point in order to choose the distribution from which to generate random numbers. The advantage of the MC over the other two methods is that MC does not require sticking to historical trends. This means that if the analyst has some beliefs regarding the future evolution of the distribution of portfolio components, those can be taken into account when the distribution is chosen. As an example, consider the figure below (Figure 1) in which the historical distribution of T-bill rates (monthly observations, from October 2010 to February 2019) is plotted against the Normal.

From the figure, it is evident that the Normal seems not to be the best possible approximation of this distribution, under the assumption that the past distribution is close to the future one. If the agent has reasons to believe that the distribution will anyway be Normal, then a Normal random number generator will be used, irrespective of the historical path. If, on the other hand, history is considered a good predictor, it may be more appropriate to use a distributional assumption such as the exponential, as evident from the plot in Figure 2. The technique can be refined further by imposing that the generated numbers do not exceed a specified threshold (such as 3%, which seems already quite high for a short term government security rate).

Once the distribution has been selected and the relevant parameters have been estimated, then it is only necessary to generate random numbers according to the specified distribution for each portfolio component. From this moment on, the steps overlap with the HS case. The components must, in fact, be combined so to obtain portfolio values and then P&L should be computed. Finally, once P&L have been ordered, the VaR estimate coincides with the $1-\alpha$ percentile.

These three techniques are different in terms of assumptions, complexity, computational effort and methodology. VC is the most simple to compute, but it is also based on the most restrictive assumptions. HS is probably the easiest one to explain to senior management since it is intuitive. This method is also appropriate if the behavior of portfolio components is not expected to change significantly in the near future. However, this technique may not be applied if portfolio components are not frequently traded or

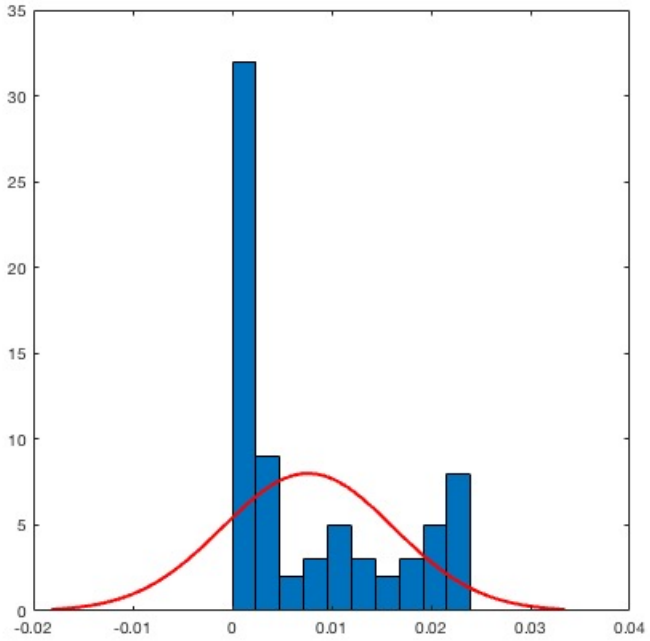

Figure 1: T-bill rates and Normal distribution

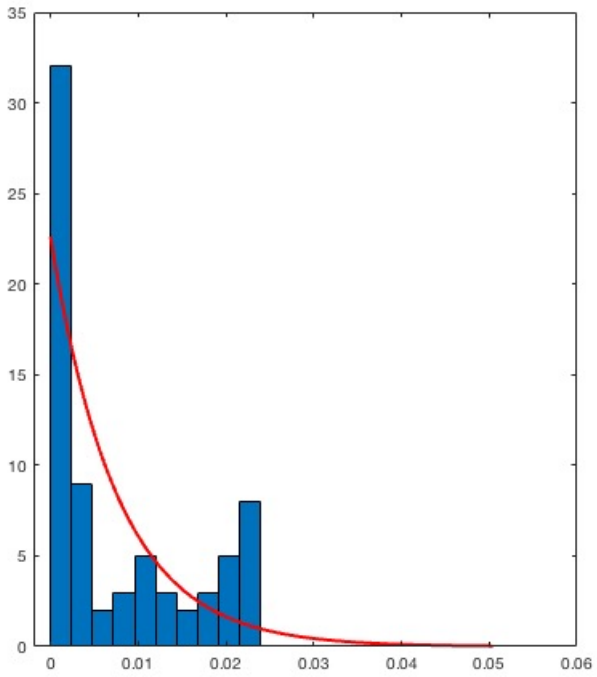

Figure 2: T-bill rates and Exponential distribution

a market for them does not exist. In this case, in fact, historical time series are not available. MC is the most sophisticated approach. It is computationally intensive and time-consuming. However, MC is the best model when the portfolios to be considered are complex or when it is necessary to incorporate in the analysis views about future evolutions.

It is then evident that the three approaches to VaR computation have both advantages and disadvantages and that none can be ex-ante considered as superior to the others. The last chapter of the thesis (5) tries to analyze the performance of different VaR models in different markets, in order to assess if, given some conditions, it is possible to consider one model as better than the others.

The relative performance of each model is analyzed with respect to the Italian and the US stock market. Specifically, following some analyses performed on various stock exchanges, VaR according to the three methods will be computed for the main stock market index (FTSE MIB in Italy and S&P500 in the US). This study aims to assess which model among the three proposed (if any) seems to be the best performer relatively to a specific stock market, assuming that the relevant index is representative of the whole market. If the best performer is found, then it may be concluded that a specific model better captures the risk profile of that market. Note that in MC VaR, Geometric Brownian Motion has been assumed as the generating process.

Daily adjusted closing prices for the two indexes were collected for the period 25 October 2016-1 January 2018. The dataset was then divided into two parts. The first one is used as in-sample, i.e., it is the basis for the estimations necessary in the application of each method. The second half is the out-of-sample, i.e., it is used for backtesting. The analysis will be a backtesting analysis, in the sense that the estimated VaR (VC, HS and MC) will be compared with the observed loss in the out-of-sample period.

Now it is necessary to define what is considered a good outcome of this comparison. Recall that VaR estimates have a convenient application: they are in fact used to determine a bank's capital requirements for market risk. An undesirable situation occurs if the amount of capital proves to be lower than the realized loss. In this case, in fact, the bank is not able to cover the loss it has incurred. Since capital requirements are computed based on VaR estimates, we can conclude that a problematic situation occurs if the estimated VaR is lower than the realized loss in absolute terms, i.e., $|VaR_i| < |P\&L_i|$ for some i. If this happens, we say that we have incurred in a failure. The concept of failure is the basis for all test that will be performed in order to assess the quality of a VaR model.

Binomial test The Binomial test is the most communicative. In this test, in fact, we compare the number of failures to the expected number of failures, assuming that failures follow a Binomial distribution. Define x as the number of times in which $|VaR_i| < |P\&L_i|$, N as the length of the vectors VaR and P&L (i.e., the number of times the estimation has been performed) and p as α, where α is the confidence level used in the estimation of VaR (in our case 5%). Assume that the failures are independent. Then the vector that contains the failures follows a Binomial distribution with parameters p and N. Define now the observed failure rate as the ratio between the observed number of failures and the total number of trials $\hat{f} = \frac{x}{N}$. The expected number of failures under the Binomial distribution is simply Np, and the standard deviation of the number of failures is $\sqrt{Np(1-p)}$. We want then to test the null hypothesis that the expected failure rate is equal to the observed failure rate (or, at the same way that the total number of observed failures is equal to the expected number of failures):

$$H_0 : \hat{f} = p = \frac{x}{N}$$

This means that we want to test that the number of failures x follows a Binomial distribution $f(x) = \binom{N}{x} p^x (1-p)^{N-x}$. As N increases and under the null hypothesis, the test statistic

$$Z = \frac{x - Np}{\sqrt{Np(1-p)}} \sim N(0,1) \tag{4}$$

is distributed as a Standard Normal. Once the desired level of confidence has been chosen, it is only necessary to compare the observed value of the test statistic with the critical value of the Standard Normal. If the observed value is lower than the critical value, then the null hypothesis cannot be rejected and thus the model should be considered as well-performing. This test is indeed an elementary test for one-population mean in which the expected number of failures is Np and $Np(1-p)$ is their variance.

Kupiec's tests Kupiec (1996) proposed two different tests. The first one, the Proportion of Failures (POF) test, basically gives the same information of the Binomial test, since it evaluates the empirical proportion of failures compared to the expected one. The null hypothesis is then the same as the Binomial test. This implies that we are still checking whether the observed number of failures is significantly distant from the expected one. The test takes the form of a likelihood ratio test (and not anymore a test for the mean) where the statistic is:

$$LR_{POF} = -2log(\frac{(1-p)^{N-x}p^x}{(1-\frac{x}{N})^{N-x}(\frac{x}{N})^x}) \tag{5}$$

Under H_0, it is distributed as a χ^2 with one degree of freedom. Again, if the observed value of the statistic is lower than the critical value, then there is not enough evidence to reject H_0 and thus the model can be considered as well-performing. The Binomial and POF tests thus carry the same information.

As the Binomial test, Kupiec's POF test provides information only with respect to the number of failures. This implies that the moment in which the failure occurs is not relevant. Since the moment in which the first failures occur is indeed of great interest, Kupiec also proposed a second test, which provides some additional useful information. This test is again a likelihood ratio type test and it is usually called Time until first failure. As the name may suggest, this test evaluates after how many successes, the first failure occurs. Specifically, the test aims to check whether the moment in which the first failure occurs is consistent with the VaR confidence level. The test statistic is this time in the form:

$$LR_{TUFF} = -2log(\frac{p(1-p)^{n-1}}{(\frac{1}{n})(1-\frac{1}{n})^{n-1}}) \tag{6}$$

Under H_0, this statistic is distributed according to a χ^2 with one degree of freedom and again, if the observed value of the statistic is lower than the critical value, then there is not enough evidence to reject H_0 and thus the model can be considered as well-performing.

Christoffersen's test The tests above consider the first or the total number of failures in isolation. However, an essential piece of information is the relative distance among failures. In particular, the level of clustering of failures may be an essential piece of information for risk managers. If failures tend to be subsequent to each other, in fact, this means that the bank may not be able to cover losses for many consecutive days. This is a significant concern and some test are needed in order to address it. Christoffersen (1998) was the first to work in this direction by considering the interaction among failures. His test is, again, a likelihood ratio type test. In order to define the test statistic, it is necessary to define several variables. n_{00} is the number of periods with no failures followed by a period with no failure. n_{10} is the number of periods with failures followed by a period with no failure. n_{01} is the number of periods with no failures followed by a period with a failure. n_{11} is the number of periods with failures followed by a period with a failure. $\pi_0 = \frac{n_{01}}{(n_{01}+n_{00})}$ is the probability of having a failure in t conditioned on the fact that no failure happened in $t-1$. $\pi_1 = \frac{n_{11}}{(n_{10}+n_{11})}$ is the probability of having a failure in t conditioned on the fact that a failure happened in $t-1$. Finally, $\pi = \frac{n_{01}+n_{11}}{(n_{00}+n_{01}+n_{10}+n_{11})}$ is the probability of having a failure in period t. The test statistic is defined as

$$LR_{CCI} = -2log(\frac{(1-\pi)^{n_{00}+n_{10}}\pi^{n_{01}+n_{11}}}{(1-\pi_0)^{n_{00}}\pi_0^{n_{01}}(1-\pi_1)^{n_{10}}\pi^{n_{11}}}) \tag{7}$$

and it is distributed as a χ^2 with one degree of freedom.

Christoffersen's test and the POF test can also be combined obtaining the conditional coverage test, $LR_{CC} = LR_{CCI} + LR_{POF}$, which is distributed according to a χ^2 with two degrees of freedom. In this kind of test, we combine two relevant information: the correct number of failures and their relative distribution.

We now turn to the application of these tests to the three analyses that have been described.

In the following figures, 3,4 and 5, the VaR estimates for the three techniques in the Italian stock market are compared with actual losses. In Table 1, test statistics for the various tests are given, in red tests that are passed. Starting from a graphical analysis, it is immediate to conclude that MC is superior to the other two techniques since the observed failure is only one. The VC approach can indeed be disregarded since it systematically underestimates the loss. In the HS case, failures are concentrated at the end of the dataset. This may suggest a mistake in the choice of the estimation period. Test results confirm these observations, since MC passes all tests and proves to be superior to the other two techniques, at least in this application.

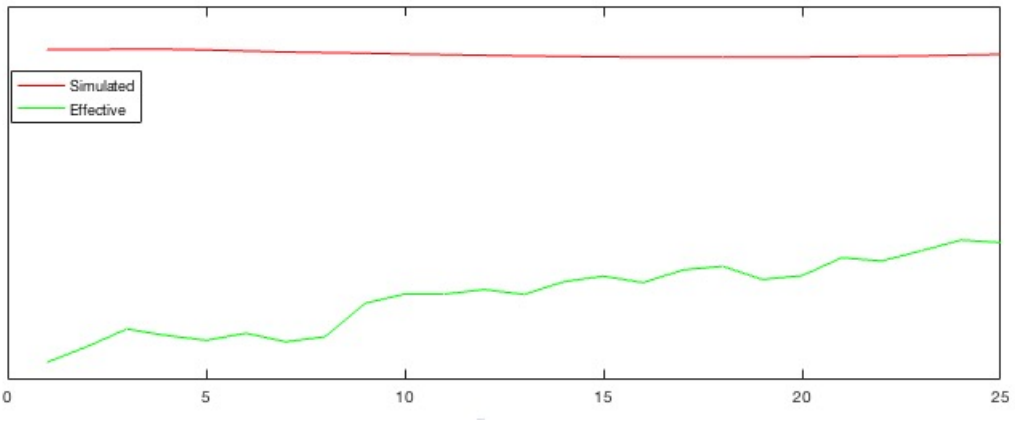

Figure 3: VaR estimates from Variance-Covariance method and realized profits or losses

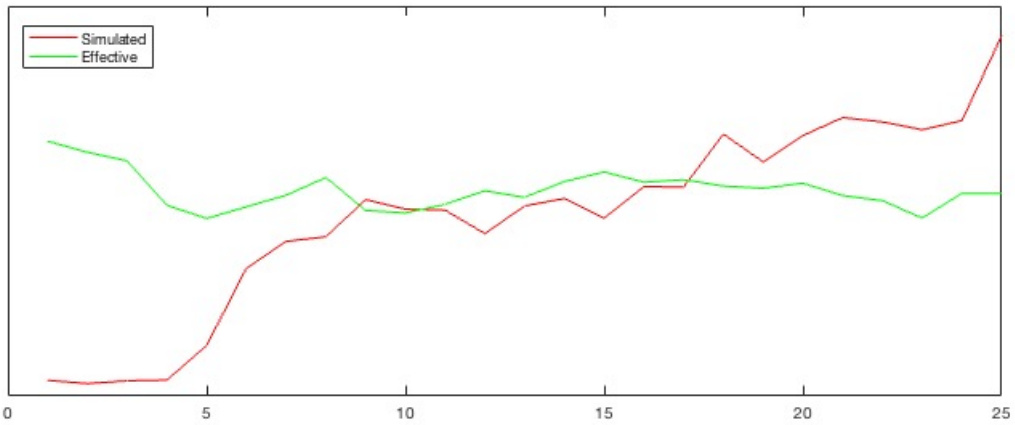

Figure 4: VaR estimates from Historical Simulation method and realized profits or losses

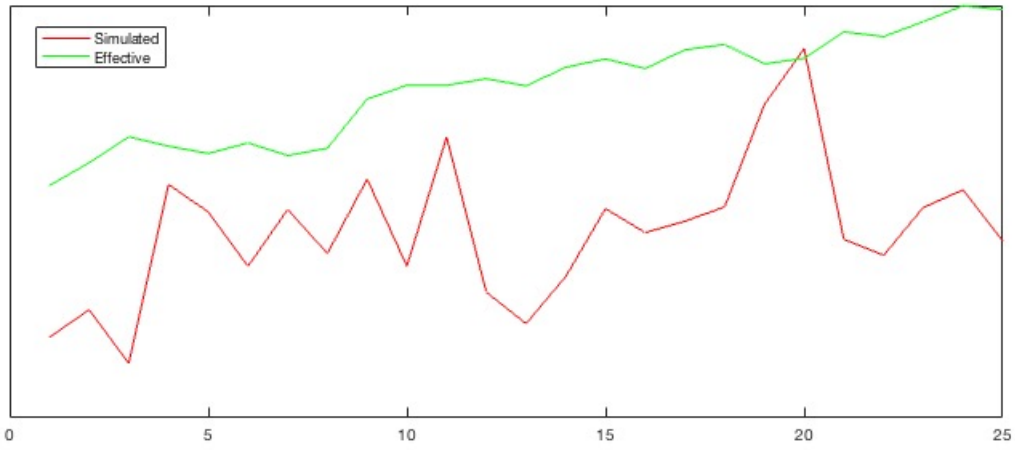

Figure 5: VaR estimates from Montecarlo Simulation method and realized profits or losses

Table 1: FTSE MIB test statistics values

	Montecarlo	Historical
Binomial	0.2294	8.0296
LR_{POF}	0.0563	27.8029
LR_{TUFF}	0.0027	0.6812
LR_{CCI}	0.0889	13.6970
LR_{CC}	0.1452	41.4999

For the US case, as shown in Figures 6,7 and 8, MC seems to be again the best performer. However, failures are much more than in the Italian case (4 vs 1). This observation is again confirmed by test results since MC passes only Christoffersen's tests (while HS fails all of them). Overall, results are consistent with the Italian stock market study (over-performance of MC).

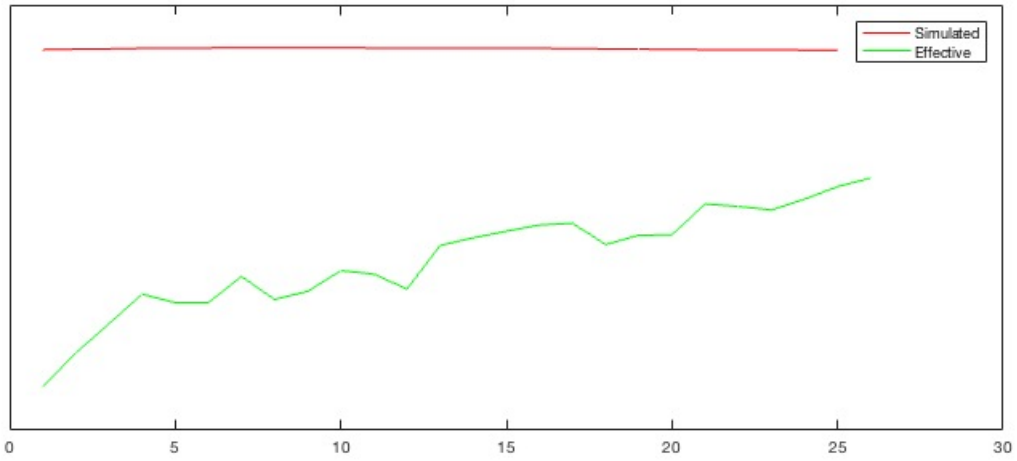

Figure 6: VaR estimates from Variance-Covariance method and realized profits or losses of SP500

Table 2: SP500 test statistics values

	Montecarlo	Historical
Binomial	2.5236	18.1238
LR_{POF}	4.1367	102.2476
LR_{TUFF}	5.9915	5.9915
LR_{CCI}	1.2332	9.4462
LR_{CC}	5.3699	113.6939

The last analysis that has been performed is the comparison of MC VaR under different distributional assumptions. The test asset is again the Italian stock market index and the assumptions that are compared are Normal, Log-Normal and Geometric Brownian motion. The aim of this study is to clarify that different distributional assumptions can lead to very different VaR estimates and to draw a conclusion regarding the best distributional assumption in the case of the Italian stock market. Results for the analysis are shown in Figures 9,10 and 11 and in Table 3 below.

By looking at the figures, we can conclude that the Geometric Brownian motion displays fewer failures and that Normal and Log-Normal tend to be very close. This observation is confirmed by test results, that clearly elect the Geometric Brownian Motion as the best distributional assumption in terms of performance. At the same time, Normal and Log-Normal seem to be exchangeable.

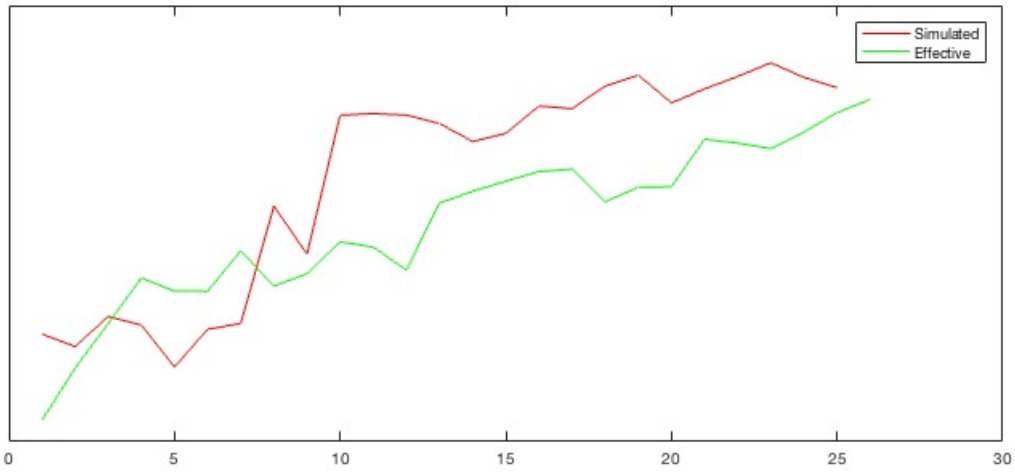

Figure 7: VaR estimates from Historical Simulation method and realized profits or losses of SP500

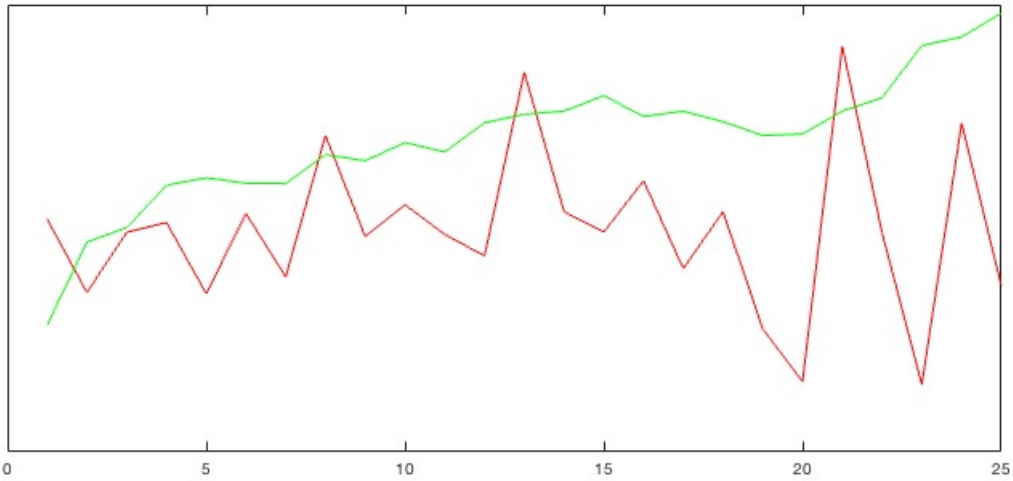

Figure 8: VaR estimates from Montecarlo Simulation method and realized profits or losses of SP500

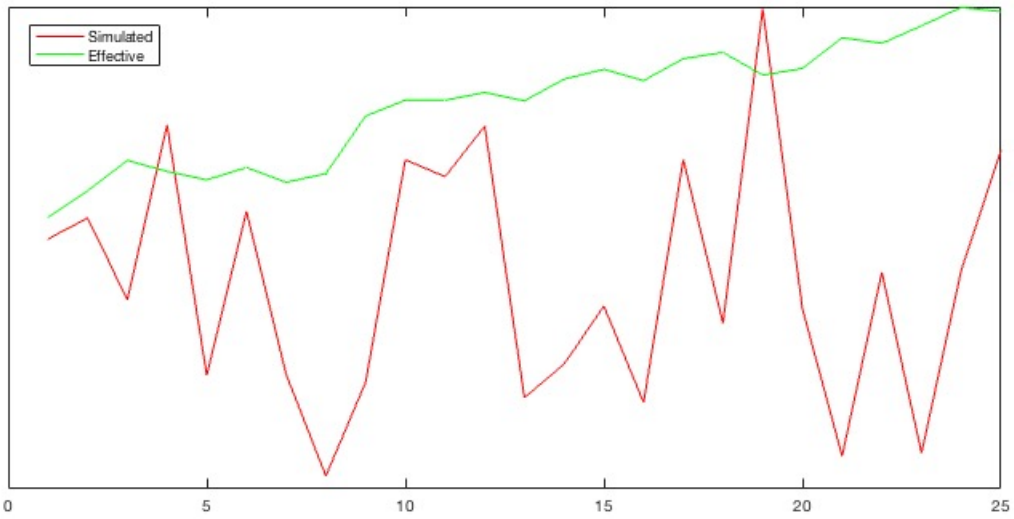

Figure 9: VaR estimates from Geometric Brownian Motion assumption and realized profits or losses

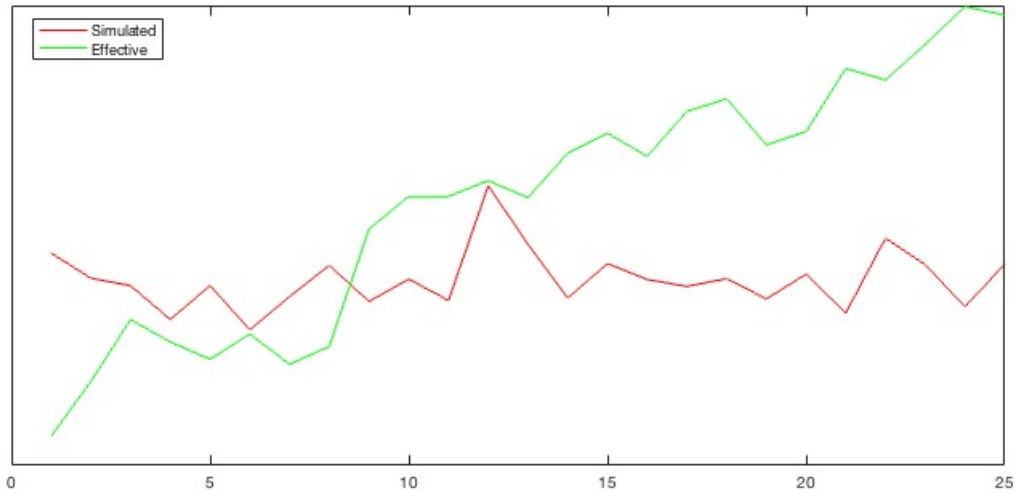

Figure 10: VaR estimates from Normal assumption and realized profits or losses

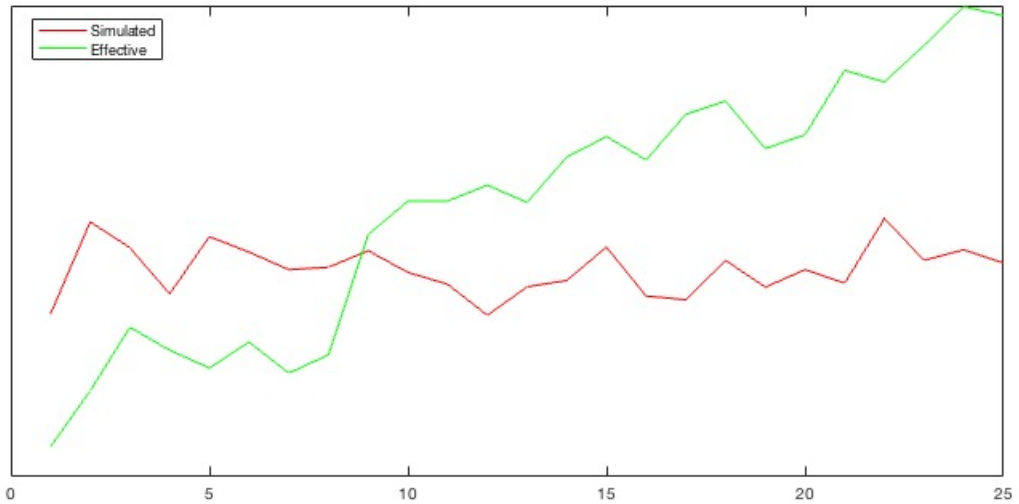

Figure 11: VaR estimates from Log-Normal assumption and realized profits or losses

Table 3: Kupiec's tests statistic values: distributional study

	GBM	Normal	Log-Normal
Binomial	0.6882	6.1942	6.1942
LR_{POF}	0.4040	18.3322	18.3322
LR_{TUFF}	2.3776	5.9915	5.9915
LR_{CCI}	0.3639	22.2593	22.2593
LR_{CC}	0.7679	40.5915	40.5915

The analyses that have been performed in the final chapter have explained why it was worth deeply discussing random number generators and the Montecarlo method. In the cases that have been analyzed, in fact, Montecarlo seems to be the best performer, i.e., the method that underestimates less frequently realized losses and thus capital requirements for market risk.

This result is somehow positive because it implies that model increased sophistication has positive effects on performance. However, some limitations of this study have already been underlined. First, the analysis was performed considering only the stock market. Specifically, a cap-weighted stock portfolio (FTSE-MIB and S&P500 Indexes) was used as test asset. The reason for the use of an index is computational ease since it allows to draw conclusions at portfolio level with no need to perform portfolio construction and evaluation. It was critically assumed that the FTSE MIB represented a good approximation of the Italian stock market (and the S&P500 of the US). If we accept this assumption, then results can be generalized to be applicable to the two stock markets. A first interesting topic to be further investigated is the consistency of results for different portfolios (for example debt or derivatives).

Another possible evolution of the analysis would work at model level. As already pointed out, VaR models are continually changing and growing compared to the three basic versions analyzed in this thesis. It would be highly interesting to analyze how those results change as models are slightly modified.

Finally, the perspective that has been endorsed here is a "supervisory perspective", meaning that the sole focus of attention was the underestimation of actual losses. Of course, other aspects deserve attention, such as the accuracy of the models, i.e., its ability to closely predict the exact amount of the loss. Tests for accuracy may thus be the object of future research.