RMBI 4990 CAPSTONE PROJECT

Development of a Copula Tool For Risk Integration

11th May 2014

Written By: CHAN, Chin Hung and WONG, Yuet Yuen
Supervised By: PENG, Xianhua

BSc in Risk Management and Business Intelligence
理學士（風險管理與商業智能學）
Hong Kong University of Science and Technology

ACKNOWLEGEMENT

We would like to express our deepest gratitude to our industrial supervisor, Dr Gary Chen, CEO of Shanghai Raiyun Financial Technologies Inc. for his invaluable insights and great support for our capstone project. We greatly appreciate the assistance provided by Mr Huang Qinghui, technology expert at Raiyun who has generously helped us through our difficulties in Java programming and implementation.

We would also like to take the opportunity to thank Prof. Peng Xianhua for his time and patience throughout our project. We are extremely grateful for his excellent advice, recommendations and enthusiastic encouragement during the planning and analysis of this research work.

It is great delight to be in part of the RMBI program for the past three years, enjoying the teaching of many exceptional professors and working with so many fellow students. It has been a very meaningful and wonderful journey. The help we received has been crucial to this study and without any of the above; this project would have been a distant reality.

TABLE OF CONENT

EXECUTIVE SUMMARY

With the increasing complexity and challenges in financial risk integration, copulas are often used in high-dimensional statistical applications to model the marginal and dependence structure of multivariate probability distributions separately. In our project, we examine and compare both the top-down and bottom-up approaches to develop a comprehensive copula tool to simulate joint capital requirements for market risk and credit risk. We first implement Monte Carlo simulations to generate bivariate samples for normal and t-copula. We followed by performing maximum likelihood estimation (MLE) for the copula parameters, and calculating their VaR and expected shortfall for evaluation. We successfully back-tested our model with actual financial data and developed a web-based demonstration of significant copula functions using GWT.

Our findings show that our models with normal copula are accurate for the top-down implementation. However, we observe that estimation models with t-copula are insufficient to reflect intrinsic heavy tail influences in the actual financial data. We identify two causes, the first one relates to its small sample size, and the second relates to our model limitation when using quasi-MLE for estimation. Finally, we discuss known inadequacies of the top-down approach and suggest more extensive research on improving the models.

1. INTRODUCTION

In recent years, we have witnessed an overwhelming degree of complexity and volatility in financial markets that has made it increasingly difficult for institutions to manage the variety of risk exposures. The global financial crisis is a great reminder of how often we fail to expect the unexpected despite highly sophisticated tools and policies that were constantly being developed. There is a need for the Basel committee and financial practitioners to push for more vigorous risk management approaches with an ever stronger emphasis on quantitative measures to aggregate risk in a holistic and systematic way for entire banks and firms.

One huge concern that arises is a potentially significant underestimation of risk when splitting value changes of the portfolio value function into its pure market risk and pure credit risk components. As described, the additive of separate risk capital cannot be our most conservative estimate and might be grossly insufficient in encompassing interactions between risk components and hence inaccurately assessing true portfolio risk. Findings in a published paper by Kupiec (2007) concludes with ambiguity that capital requirements derived from an integrated market and credit risk measures can possibly either be larger or smaller than that from the piecemeal risk measures. Kupiec noted that the sign and magnitude of the differences is largely dependent on the piecemeal approaches applied.

In the broad literature, there are two general approaches to integrate risks, which are known as the top-down and bottom-up approaches as was also demonstrated by Grundlk (2006). The former employs adequate copula functions to capture the dependence structure of individual marginal loss distributions of various risks. The latter models different risk types simultaneously in one common, integrated framework. The determination of the necessary amount of economic capital needed is central to the discussion. The primary objective of this project will be to examine and compare both of these approaches to develop a comprehensive copula tool to simulate joint capital requirements for market risk and credit risk.

2. LITERATURE REVIEW

Traditionally as almost frequently taken to be convenient, loss distributions are often assumed to be multivariate normally distributed. This is certainly false for credit or operational losses, of which we are careful to omit this strong assumption in our methodology. Consistent with prior research, we will first begin with the highly popular and easiest-to-implement top-down approach by identifying separate marginal distributions of losses and linking them with appropriate copula functions. Grudke (2006) highlighted several limitations of the top-down approach, including the difficulty in selecting the correct copula function with limited amounts of financial data and an inability to model complex interactions between risk types. The bottom-up approach is contrasted with a more exact and detailed description of the total loss distribution, where possible stochastic dependencies are taken into consideration without needing later aggregation of the risk-specific loss distributions by copulas.

One of the first attempts for risk integration in market and credit risk portfolio models using the bottom-up approach was made by Kijima and Muromachi (2000) modelling interest rate risk as a stochastic process into an intensity-based credit portfolio model. Barth (2000) implemented Monte Carlo simulations to calculate worst-case risk measures for a portfolio of interest rate swaps with counterparty risk. Following that, Jobst and Zenios (2003) introduced the tracking of corporate bond indices and explored the intertemporal price sensitivity of coupon bonds to changes in interest rates. Walder (2002) investigated the effect using extended Vasicek processes for interest rates and default intensities.

An extensive study conducted by Branhill and Maxwell (2002) assumed correlations between various factors and simulated the risk-free term structure, credit spreads, exchange rate and equity market indices. Grudke (2005) proceeded to provide an improved CreditMetrics model with correlated interest rate and credit spread risk and also varied the parameterization and specification of the model to analyse its impact on the newly added integrated market risk factors.

Using the top-down approach, Ward and Lee (2002) started applying the normal copula for risk integration in an insurance company. Subsequent research by Rosenberg and Schuermann (2006) aggregated market, credit and operational risk in a large international bank and derived VaR and expected shortfall estimates for the inter-risk correlations. Assuming no diversification effects, they discovered that their simple additive approach overestimates risk by more 40%. Moreover, a typical alternative, assuming joint normality of the risks, underestimates risk by a similar amount. Their findings indicated that bottom-up approach remains relevant for integrating risk types with not too dissimilar distributional behaviour, such as that of market and credit risk where their interactions between measures of correlation and the higher moments of the marginals are often subtle and complicated. Top-down approach can be suggested to combine dissimilar distributions, such as that of integrated market and credit risk to aggregate with operational risk.

In a more recent paper, Grunke (2009) reassessed the accuracy of the total economic capital using the top-down approach. He found a significant underestimation of total economic capital for lower credit qualities. Moreover, he also discovered that the direction and strength of the stochastic dependence between the risk types, the copula function employed, and the loss definitions all having an impact on the performance of the top-down approach. This finding further indicates potential flaws in the top-down approach that we might need to take note during implementation.

3. RESEARCH METHODOLOGY

3.1 Copulas and Concept for Top-down Approach

Copulas are described as multivariate probability distribution defined on the n-dimensional unit cube [0, 1] such that every marginal distribution is uniform on the interval [0, 1]. Specifically, the following properties are characteristic and must be present in any copula.

- $C(u) = C(u_1, u_2, \dots, u_n)$ is a distribution $C: [0, 1]^d \rightarrow [0, 1]$.
- $C(u)$ is increasing in each component u_i
- $C(1, \dots, 1, u_i, 1, \dots, 1) = u_i$, $i = 1, 2, \dots, n$

Sklar's Theorem states that any joint distribution function $F_{X,Y}(x, y)$ can be expressed in terms of its copula function and the corresponding marginal distribution functions $F_X(x)$ and $F_Y(y)$. The density representation is defined.

$$F_{X,Y}(x, y) = \mathrm{C}(F_X(x), F_Y(y))$$

$$f_{X,Y}(x, y) = f_X(x) f_Y(y) c(F_X(x), F_Y(y))$$

where $c(u, v) = \left(\frac{\partial^2}{\partial u\, \partial v}\right) C(u, v)$ denotes the copula density function, and $f_X(x)$ and $f_Y(y)$ are the marginal density functions.

To recover the copula function of a multivariate distribution $F_{X,Y}(x, y)$, the method of inversion can be applied.

$$C(u, v) = F_{X,Y}(F_X^{-1}(u), F_Y^{-1}(v))$$

where $F_X^{-1}(u)$ and $F_Y^{-1}(v)$ are the inverse marginal distribution functions.

Two copula functions most commonly used for risk management purposes are the normal copula and the t-copula, which are defined below.

$$C(u, v; \rho) = \int_{-\infty}^{\Phi^{-1}(u)} \int_{-\infty}^{\Phi^{-1}(v)} \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left\{-\frac{s^2 - 2\rho st + t^2}{2(1-\rho^2)}\right\} ds\, dt$$

where $\Phi^{-1}(.)$ denotes the inverse of the cumulative density function of the standard normal distribution.

$$C(u,v,n;\rho) = \int_{-\infty}^{T_n^{-1}(u)} \int_{-\infty}^{T_n^{-1}(v)} \frac{1}{2\pi\sqrt{1-\rho^2}} \left(1+\frac{s^2-2\rho st+t^2}{2(1-\rho^2)}\right)^{-(n+2)/2} ds\, dt$$

where $T_n^{-1}(.)$ denotes the inverse of the cumulative density function of the t-distribution with n degrees of freedom. The only parameter of the bivariate normal copula is the correlation parameter ρ whereas the bivariate t-copula has two parameters that are the correlation parameter ρ and the number of degrees of freedom n.

Generating random vectors X and Y from the bivariate normal and t-copula is described in the algorithms shown below.

Normal copula:

1. Draw a realization (a, b) of two standard normally distributed random variables A and B with $Corr(A, B) = \rho$ where ρ is the parameter of the normal copula.
2. Transform the realizations (a, b) into the realizations $(\tilde{a}, \tilde{b})$ of two random variables $\tilde{A}$ and $\tilde{B}$ which are uniformly distributed on the interval [0,1] by computing $\tilde{a} = \Phi(a)$ and $\tilde{b} = \Phi(b)$ where $\Phi(.)$ denotes the cumulative density function of the standard normal distribution.
3. Finally, to ensure the correct marginal distribution, compute $X = F_X^{-1}(\tilde{a})$ and $Y = F_Y^{-1}(\tilde{b})$.

t-copula:

1. Draw a realization (a, b) of two standard normally distributed random variables A and B with $Corr(A, B) = \rho$ where ρ is the parameter of the t-copula.
2. Transform the realizations (a, b) into the realizations $(\tilde{a}, \tilde{b})$ of two random variables $\tilde{A}$ and $\tilde{B}$ which are t-distributed with n degrees of freedom by computing $\tilde{a} = a\sqrt{n/s}$ and $\tilde{b} = b\sqrt{n/s}$ where s is a realization of $S \sim \chi^2(n)$ independent of A and B.

3. Transform the realizations $(\tilde{a}, \tilde{b})$ into the realizations $(\hat{a}, \hat{b})$ of two random variables $\tilde{A}$ and $\tilde{B}$ which are uniformly distributed on the interval [0,1] by computing $\hat{a} = t_n(\tilde{a})$ and $\hat{b} = t_n(\tilde{b})$ where $t_n(.)$ denotes the cumulative density function of the t-distribution with n degrees of freedom.
4. Finally, to ensure the correct marginal distribution, compute $X = F_X^{-1}(\tilde{a})$ and $Y = F_Y^{-1}(\tilde{b})$.

3.2 Proposed Stages for Project

Stage 1 (Sep 2013 – Nov 2013): We will begin by implementing the top-down approach. Denoting random variables X and Y as the market risk return and the credit risk return respectively, we assume certain dummy parameters of the distributions. In our project, we will model market risk with a normal distribution or a t-distribution and credit risk with a beta distribution. Subsequently, we will run Monte Carlo simulation to repeat the process for a large sample of 100,000 data points to compute X + Y. Value at Risk (VaR) and expected shortfall are computed for evaluation and analysis.

Stage 2 (Dec 2013 – Jan 2014): We will back-test our model with actual financial data, from which we estimate the correct parameters for the corresponding distributions of market risk and credit risk. Maximum likelihood method and the related method of inference functions for margins (IFM) are applied to estimate the parameters for the specific copulas.

Stage 3 (Feb 2014 – Mar 2014): We will develop a web-based demonstration of our copula tool that comprises of a user-friendly interface. The features of the web platform will include reading bivariate data of loss distributions from an array, controlling of parameters, simulating copulas and displaying simulated data and showing calculation details of the likelihood function and VaR.

Stage 4 (Apr 2014): We will compare our results with the bottom up approach according to and implemented by Grundke (2006). We will identify limitations and suggest further improvements for our model.

3.3 Maximum Likelihood Estimation for Top-down Approach

The sample data matrix D =$\{r_{L_1}(t), r_{L_2}(t)\}_{t=1}^{T}$ of credit and market risk loss returns of the banking book is simulated. The parameter $\hat{\rho}$ of the bivariate normal copula and the parameters $(\hat{n}, \hat{\rho})$ of the bivariate t copula are to be estimated. The parameters of the marginal distributions and the parameters of the copula, combined in the parameter vector θ, can be estimated simultaneously by the maximum likelihood method. Its log-likelihood function is given as follows.

$$l(\theta) = \ln\left(\prod_{t=1}^{T} f_{r_{L1}, r_{L_2}}(r_{L_1}(t), r_{L_2}(t); \theta)\right)$$

$$= \ln\left(\prod_{t=1}^{T} \frac{\partial^2 C(u,v;\theta)}{\partial u \partial v} f_{r_{L_1}}\big(r_{L_1}(t);\theta\big)\, f_{r_{L2}}(r_{L2}(t);\theta))\right)$$

$$= \sum_{t=1}^{T}\left(\ln\left(\frac{\partial^2 C(u,v;\theta)}{\partial u \partial v}\right) + \ln\left(f_{r_{L1}}\big(r_{L_1}(t);\theta\big)\, f_{r_{L_2}}(r_{L_2}(t);\theta))\right)\right)$$

where $(u, v) = \left(F_{r_{L1}}(r_{L1}(t);\theta), F_{r_{L2}}(r_{L2}(t);\theta)\right)$

The maximum likelihood estimator (MLE) is obtained by maximizing the log-likelihood function: $\hat{\theta}_{MLE} = \arg max\, l(\theta)$

To reduce computational cost in solving the optimization problem, the method of inference functions for margins (IFM) is applied. The IFM method is a two-step-method where the parameters θ_1 of the marginal distributions are first estimated separately and followed by the determination of the parameters θ_2 of the copula given the prior value of θ_1.

$$\hat{\theta}_1^{IFM} = \arg max \sum_{t=1}^{T}\left(\ln\left(f_{r_{L1}}\big(r_{L_1}(t);\theta_1\big)\, f_{r_{L_2}}(r_{L_2}(t);\theta_1))\right)\right)$$

$$\hat{\theta}_2^{IFM} = \arg max \sum_{t=1}^{T}\left(\ln\left(\frac{\partial^2 C\big(u,v;\theta_2;\hat{\theta}_1\big)}{\partial u \partial v}\right)\right)$$

The complete IFM estimator is defined: $\hat{\theta}^{IFM} = (\hat{\theta}_1^{IFM}, \hat{\theta}_2^{IFM})$

In general, an equivalence of $\hat{\theta}_{MLE}$ and $\hat{\theta}^{IFM}$ does not hold, but it can be shown that under regularity conditions, the IFM estimator is asymptotic normally distributed. For our approach, the IFM procedure is used for estimating the parameters $\hat{\theta}_2^{IFM}$ of the copula function. We compute the parameters $\hat{\theta}_1^{IFM}$ of the marginal distributions by the method of moments instead of employing the first equation.

For the normal copula, the second equation is equivalent to:

$\hat{\rho}_{normal}^{IFM}$ = arg max

$$\sum_{t=1}^{T}\left(\ln\left(\frac{1}{2\pi\sqrt{1-\rho^2}}\exp\left(-\frac{\left(\Phi^{-1}(u)\right)^2-2\rho\Phi^{-1}(u)\Phi^{-1}(v)+\left(\Phi^{-1}(v)\right)^2}{2(1-\rho^2)}\right)\frac{1}{\phi\left(\Phi^{-1}(u)\right)}\frac{1}{\phi\left(\Phi^{-1}(v)\right)}\right)\right)$$

For the t copula, the second equation is equivalent to:

$\left(\hat{n}^{IFM},\hat{\rho}_{t-cop}^{IFM}\right)$ = arg max

$$\sum_{t=1}^{T}\left(\ln\left(\frac{1}{2\pi\sqrt{1-\rho^2}}\left(1+\frac{\left(T_n^{-1}(u)\right)^2-2\rho T_n^{-1}(u)T_n^{-1}(v)+\left(T_n^{-1}(v)\right)^2}{n(1-\rho^2)}\right)^{-\frac{n+2}{2}}\frac{1}{t_n\left(T_n^{-1}(u)\right)}\frac{1}{t_n\left(T_n^{-1}(v)\right)}\right)\right)$$

where $(u,v)=\left(F_{r_{L1}}\left(r_{L1}(t);\hat{\theta}_1\right),F_{r_{L2}}\left(r_{L2}(t);\hat{\theta}_1\right)\right)$

3.4 VaR and Expected Shortfall Computation for Top-down Approach

As taken from Glasserman (2004, p. 491), we can construct a confidence interval for VaR estimator $\hat{\text{VaR}}(p)$ given the fact that the number of samples which are smaller than VaR(p) has a binomial distribution with parameters D and p.

When $r_{L_{(i)}}$ ($i \in \{ 1,\dots ,D \}$) with $r_{L_{(1)}} \leq r_{L_{(2)}} \leq \dots \leq r_{L_{(D)}}$ denote the order statistics of D simulations of the credit portfolio loss, this yields a following formula.

$$P\left(r_{L_{(r)}} \leq \text{VaR}(p) < r_{L_{(s)}}\right) = \sum_{i=r}^{s-1}\binom{D}{i}p^i\,(1-p)^{D-i}$$

The numbers r and s are chosen such that the above result will be close to 1 – α.

$[L_{(r)}(H), L_{(s)}(H)]$ represents a $(1 - \alpha)$ confidence interval for the sample VaR estimator V$\hat{\text{a}}$R(p). This confidence interval is not only asymptotically valid, but also for a finite number D of simulation runs.

Employing the normal approximation $N($ Dp, $D(1$-$p)p$ $)$ for a binomially distributed random variable with parameters D and p , the numbers r and s are given by the following according to Gupton et al. (1997, pp. 150).

$$r = \left\lfloor Dp - \Phi^{-1}\left(1 - \frac{\alpha}{2}\right)\sqrt{Dp(1-p)}\right\rfloor$$

$$s = \left\lceil Dp + \Phi^{-1}\left(1 - \frac{\alpha}{2}\right)\sqrt{Dp(1-p)}\right\rceil$$

An approximate $(1 - \alpha)$ confidence interval for the estimate of the expected shortfall $\text{E}\hat{S}(p)$ is given by:

$$\left[\text{E}\hat{S}(p) - \Phi^{-1}\left(1 - \frac{\alpha}{2}\right)\sqrt{Var\left(\text{E}\hat{S}(p)\right)}, \text{E}\hat{S}(p) + \Phi^{-1}\left(1 - \frac{\alpha}{2}\right)\sqrt{Var\left(\text{E}\hat{S}(p)\right)}\ \right]$$

$$\text{where } Var\left(\text{E}\hat{S}(p)\right) = \frac{Var\left(V\hat{a}R(p), \ldots, r_{L_{(D)}}\right) + p\left(\text{E}\hat{S}(p) - V\hat{a}R(p)\right)^2}{k}$$

$$Var\left(V\hat{a}R(p), \ldots, r_{L_{(D)}}\right) = \frac{\sum_{j=D-k+1}^{D}\left(r_{L_{(f)}} - \text{E}\hat{S}(p)\right)^2}{k-1}$$

$$\text{E}\hat{S}(p) = \frac{1}{k}\sum_{j=D-k+1}^{D} r_{L_{(f)}}$$

$$V\hat{a}R(p) = r_{L_{(D-k+1)}}$$

$$k = D(1-p)$$

The variance formula for $\text{E}\hat{S}(p)$ considers both sources of uncertainty, that include the uncertainty in VaR estimate $V\hat{a}R(p) = r_{L_{(D-k+1)}}$ and the uncertainty in the conditional expectation of r_L, given that r_L is larger than $V\hat{a}R(p)$.

4. RESULTS SUMMARY

4.1 Java Code Implementation

We have initiated our project by searching relevant Java packages and designing the structure of our program. For the first and second stages, we used several mathematical functions from open source external libraries and wrote the full programming code in Java from scratch to implement our copula tool.

For the third stage, we improved and integrated code into web programming languages including Jquery and PHP. We subsequently chose to build an interactive Java web applet that is very easy-to-use, cross-platform compatible and secure for our purposes. Google Web ToolKit (GWT) is used because it supports complex Java programming, has built-in Remote Protocol Call (RPC) framework for client-server communication and is friendly for our testing and debugging. RPC specifically handles the serialization and exchange of Java objects over HTTP in the arguments in the method calls and return values through invoking several services (See Appendix A).

There are several many useful Java packages selected for our program.

Package	**Description**
Apache POI	Functionalities included reading and manipulating Office document file formats such as Excel.
Apache Math Commons (Linear Algebra)	Functionalities included computing eigenvectors, real matrix operations and solving linear systems.
Apache Math Commons (Distributions)	Functionalities included computing cumulative probability distribution probabilities and inverse values.
Apache Math Commons (Statistics)	Functionalities included evaluating descriptive statistics, covariance matrices and standard statistical tests.
Apache Math Commons (Numerical Analysis)	Functionalities included computing numerical root-finding, integration, interpolation and differentiation.
Apache Math Commons (Optimization)	Functionalities included direct search and advanced methods to optimizing objective cost functions.
Apache Math Commons (Data Generation)	Functionalities included generating cryptographically secure sequences or pseudo-random numbers and vectors.

There are a total of 10 functions in our final client-side code sample for the class WebCopulaTool, which is summarized below.

Function	Description
onModuleLoad	Draws user interface, defines initial parameters and sets up all onClick event handlers.
readExcelFile	Reads input parameters, invokes server-side functions readExcelData and estimateSampleParamatrix, and outputs summary results.
simulateNormCopula, simulateTCopula	Reads input parameters, invokes server-side functions generateNormSamples / generateTSamples and estimateSampleParamatrix, and outputs summary results.
estimateNormCopula, estimateTCopula	Reads input parameters, invokes server-side functions estimateNormParameters / estimateTParameters, and outputs calculation details.
calcVaRNormCopula, calVaRTCopula	Reads input parameters, invokes server-side functions calcVaRNormCopula / calcVaRNormCopula, and outputs calculation details.
calcESNormCopula, calcESTCopula	Reads input parameters, invokes server-side functions calcESNormCopula / calcESTCopula, and outputs calculation details.

There are also a total of 10 functions in the server-side code sample for the class MathFunctionsServiceImpl, which is described below.

Function	Description
readExcelData	Retrieves new bivariate array from given Excel file.
generateNormSamples, generateTSamples	Runs Monte Carlo simulation to generate specified samples with given parameters.
estimateSampleParamatrix	Creates list of descriptive statistics for data sample.
estimateNormParameters, estimateTParameters	Estimates maximum log-likelihood for copula parameters of given bivariate distributions.
calcVaRNormCopula, calVaRTCopula	Computes VaR within a given confidence interval for data sample.
calcESNormCopula, calcESTCopula	Computes expected shortfall within a given confidence interval for data sample.

Client-side Applet Code Sample: WebCopulaTool.java
(Additional Details Are Elaborated In The Comments For Readability)

```
package web.copulatool.client;

// Import Relevant GWT Functions
import static java.lang.Math.*;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Timer;
import com.google.gwt.i18n.client.NumberFormat;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.TextArea;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.AsyncCallback;

public class WebCopulaTool implements EntryPoint {

    // Define Refresh Interval For Server Callback
    private static final int REFRESH_INTERVAL = 100;

    // Initiate Panels For Top User Interface
    private VerticalPanel mainPanel = new VerticalPanel();
    private HorizontalPanel addPanel = new HorizontalPanel();
    private VerticalPanel part1Panel = new VerticalPanel();
    private VerticalPanel part2Panel = new VerticalPanel();
    private VerticalPanel part3Panel = new VerticalPanel();

    // Initiate Panels For Bottom User Interface
    private Label outputLabel = new Label();
    private TextArea outputArea = new TextArea();

    // Initiate Big Labels For User Interface
    private Label ReadLabel = new Label();
    private Label EstimateLabel = new Label();
    private Label SimulateLabel = new Label();
    private Label CalculateLabel = new Label();

    // Initiate Button Labels For User Interface
    private Label initialCorrelLabel = new Label();
    private Label initialNDegreesLabel = new Label();
    private Label meanXLabel = new Label();
    private Label meanYLabel = new Label();
    private Label sdXLabel = new Label();
    private Label sdYLabel = new Label();
    private Label sampleSizeLabel = new Label();
    private Label correlLabel = new Label();
    private Label NDegreesLabel = new Label();
    private Label probLabel = new Label();
    private Label confidenceIntervalLabel = new Label();
    private Label spaceSeparation = new Label();

    // Initiate Textboxes and Buttons For First Top User Interface
    private TextBox fileAddress = new TextBox();
    private Button readFile = new Button("Read Excel Data");
    private TextBox initialCorrel = new TextBox();
    private TextBox initialNDegrees = new TextBox();
    private Button estimateNormCopula = new Button("Estimate Normal Copula
              Parameters");
    private Button estimateTCopula = new Button("Estimate T Copula
              Parameters");
    // Initiate Textboxes and Buttons For Second Top User Interface
    private TextBox meanX = new TextBox();
    private TextBox meanY = new TextBox();
    private TextBox sdX = new TextBox();
    private TextBox sdY = new TextBox();
    private TextBox sampleSize = new TextBox();
    private TextBox correl = new TextBox();
    private TextBox NDegrees = new TextBox();
    private Button simulateNormCopula = new Button("Simulate Normal
            Copula");
    private Button simulateTCopula = new Button("Simulate T Copula");

    // Initiate Textboxes and Buttons For Third Top User Interface
    private TextBox prob = new TextBox();
    private TextBox confidenceInterval = new TextBox();
    private Button calcVaRNormCopula = new Button("Calculate VaR for
            Normal Copula");
    private Button calcVaRTCopula = new Button("Calculate VaR for T
            Copula");
    private Button calcESNormCopula = new Button("Calculate ES for Normal
            Copula");
    private Button calcESTCopula = new Button("Calculate ES for T
            Copula");

    // Declare Other Relevant Variables
    private int getsampleSize;
    private double[] XNormalSampleVector;
    private double[] YNormalSampleVector;
    private double[] NormalSampleVector;
    private double[] XTSampleVector;
    private double[] YTSampleVector;
    private double[] TSampleVector;
    private double[][] NormalSampleParamatrix = new double[2][2];
    private MathFunctionsServiceAsync mathFunctionsSvc =
            GWT.create(MathFunctionsService.class);

    public void onModuleLoad() {

        // Assemble First Top Control Panel
        ReadLabel.setText("READ");
        part1Panel.add(ReadLabel);
        part1Panel.add(fileAddress);
        part1Panel.add(readFile);
        EstimateLabel.setText("ESTIMATE");
        part1Panel.add(EstimateLabel);
        part1Panel.add(initialCorrelLabel);
        initialCorrelLabel.setText("Initial Correlation");
        part1Panel.add(initialCorrel);
        part1Panel.add(initialNDegreesLabel);
        initialNDegreesLabel.setText("Initial Degrees of Freedom");
        part1Panel.add(initialNDegrees);
        part1Panel.add(estimateNormCopula);
        part1Panel.add(estimateTCopula);

        // Assemble Second Top Control Panel
        SimulateLabel.setText("SIMULATE");
        part2Panel.add(SimulateLabel);
        part2Panel.add(meanXLabel);
        meanXLabel.setText("Mean of X");
        part2Panel.add(meanX);
        part2Panel.add(meanYLabel);
        meanYLabel.setText("Mean of Y");
        part2Panel.add(meanY);
        part2Panel.add(sdXLabel);
        sdXLabel.setText("Standard Deviation of X");
        part2Panel.add(sdX);
        part2Panel.add(sdYLabel);
        sdYLabel.setText("Standard Deviation of Y");
        part2Panel.add(sdY);
        part2Panel.add(sampleSizeLabel);
        sampleSizeLabel.setText("Sample Size");
        part2Panel.add(sampleSize);
        part2Panel.add(correlLabel);
        correlLabel.setText("Correlation (Prior Distribution)");
        part2Panel.add(correl);
        part2Panel.add(NDegreesLabel);
        NDegreesLabel.setText("Degrees of Freedom (Prior Distribution)");
        part2Panel.add(NDegrees);
        part2Panel.add(simulateNormCopula);
        part2Panel.add(simulateTCopula);

        // Assemble Third Top Control Panel
        CalculateLabel.setText("CALCULATE");
        part3Panel.add(CalculateLabel);
        part3Panel.add(probLabel);
        probLabel.setText("Probability for VaR / ES");
        part3Panel.add(prob);
        part3Panel.add(confidenceIntervalLabel);
        confidenceIntervalLabel.setText("Confidence Interval");
        part3Panel.add(confidenceInterval);
        part3Panel.add(calcVaRNormCopula);
        part3Panel.add(calcVaRTCopula);
        part3Panel.add(spaceSeparation);
        part3Panel.add(calcESNormCopula);
        part3Panel.add(calcESTCopula);

        // Assemble All Parts Of Control Panel
        addPanel.add(part1Panel);
        addPanel.add(part2Panel);
        addPanel.add(part3Panel);

        // Assemble Main Panel
        mainPanel.add(addPanel);
        outputLabel.setText("OUTPUT RESULTS");
        mainPanel.add(outputLabel);
        mainPanel.add(outputArea);
        outputArea.setCharacterWidth(100);
        outputArea.setVisibleLines(32);

        // Add Style For User Interface
        part1Panel.addStyleName("partPanel");
        part2Panel.addStyleName("partPanel");
        part3Panel.addStyleName("partPanel");
        ReadLabel.addStyleName("BigLabel");
        EstimateLabel.addStyleName("BigLabel2");
        SimulateLabel.addStyleName("BigLabel");
        CalculateLabel.addStyleName("BigLabel");
        spaceSeparation.addStyleName("SpaceLabel");
        outputLabel.addStyleName("outputLabel");
        outputArea.addStyleName("outputArea");
        mainPanel.addStyleName("mainPanel");

        // Define Default Values For Parameters
        fileAddress.setText("C:\\Users\\workspace\\WebCopulaTool
             \\RiskLossData.xlsx");
        meanX.setText("0.2");
        meanY.setText("0.530");
        sdX.setText("0.3");
        sdY.setText("0.2686");
        sampleSize.setText("1000");
        correl.setText("0.1");
        NDegrees.setText("1.5");
        initialCorrel.setText("0.1");
        initialNDegrees.setText("1.5");
        prob.setText("0.99");
        confidenceInterval.setText("0.95");

        // Associate Main Panel With HTML Host Page
        RootPanel.get("mainCopulaTool").add(mainPanel);

        // Onclick Event Handler For Read Excel Data
        readFile.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                readExcelFile();
            }
        });

        // Onclick Event Handler For Simulate Normal Copula
        simulateNormCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                simulateNormCopula();
            }
        });

        // Onclick Event Handler For Simulate T Copula
        simulateTCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                simulateTCopula();
            }
        });

        // Onclick Event Handler For Calculate VaR (Normal Copula)
        calcVaRNormCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                calcVaRNormCopula();
            }
        });

        // Onclick Event Handler For Calculate VaR (T Copula)
        calcVaRTCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                calcVaRTCopula();
            }
        });

        // Onclick Event Handler For Calculate ES (Normal Copula)
        calcESNormCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                calcESNormCopula();
            }
        });

        // Onclick Event Handler For Calculate ES (T Copula)
        calcESTCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                calcESTCopula();
            }
        });

        // Onclick Event Handler For Estimate Parameters (Normal Copula)
        estimateNormCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                estimateNormCopula();
            }
        });
        // Onclick Event Handler For Estimate Parameter (T Copula)
        estimateTCopula.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                estimateTCopula();
            }
        });
    }

    public void readExcelFile() {

        // Get Input Text As Strings
        String StrfileAddress = fileAddress.getText();

        // Define Read Excel File Callback Function
        AsyncCallback<double[][]> ReadCallback = new
             AsyncCallback<double[][]>() {

            public void onFailure(Throwable caught) {

                outputArea.setText(outputArea.getText() + "Error " +
                caught + "\n");
                 }

            public void onSuccess(double[][] result) {

                getsampleSize = (int) (result[0][0]-1);
                XTSampleVector = new double[(int) (result[0][0]-1)];
                YTSampleVector = new double[(int) (result[0][0]-1)];
                TSampleVector = new double[(int) (result[0][0]-1)];

                for(int i=0; i<result[0][0]-1; i++){
                    XTSampleVector[i] = result[i+1][0];
                    YTSampleVector[i] = result[i+1][1];
                    TSampleVector[i] =  result[i+1][2];
                    outputArea.setText(outputArea.getText() + "<" + i + ">
                    X: " + result[i+1][0]  + ", Y: " + result[i+1][1] + ",
                       X+Y: " + result[i+1][2] + "\n");
                }

             AsyncCallback<double[][]> EstCallback = new
             AsyncCallback<double[][]>() {

                    public void onFailure(Throwable caught) {
                        outputArea.setText(outputArea.getText() + "Error "
                        + caught + "\n");
                    }

                    public void onSuccess(double[][] result) {
                        TSampleParamatrix[0][0] = result[0][0];
                        TSampleParamatrix[0][1] = result[0][1];
                        TSampleParamatrix[1][0] = result[2][0];
                        TSampleParamatrix[1][1] = result[2][1];

                        outputArea.setText(outputArea.getText() +
                        "\nDescriptive Stats For Bivariate Data (Sample
                        Size: " + getsampleSize + ")");
                        outputArea.setText(outputArea.getText() + "\nMean
                        of Sample X: " + result[0][0] + ", SD of Sample X:
                        " + result[0][1]);
                        outputArea.setText(outputArea.getText() + "\nMean
                        of Sample Y: " + result[1][0] + ", SD of Sample Y:
                        " + result[1][1]);
                        outputArea.setText(outputArea.getText() + "\nAlpha
                        of Sample Y: " + result[2][0] + ", Beta of Sample
                        Y: " + result[2][1] + "\n\n");
                    }
                };
                mathFunctionsSvc.estimateSampleParamatrix(XTSampleVector,
                YTSampleVector, getsampleSize, EstCallback);
            }
        };

        // Set Up Callback Function
        outputArea.setText(outputArea.getText() + "Reading Excel Data...
        \n\n");
        mathFunctionsSvc.readExcelData(StrfileAddress, ReadCallback);
    }

    public void simulateNormCopula() {

        // Get Input Text As Strings
        final String StrmeanX = meanX.getText().toUpperCase().trim();
        final String StrmeanY = meanY.getText().toUpperCase().trim();
        final String StrsdX = sdX.getText().toUpperCase().trim();
        final String StrsdY = sdY.getText().toUpperCase().trim();
        final String Strcorrel = correl.getText().toUpperCase().trim();
        final String StrsampleSize =
              sampleSize.getText().toUpperCase().trim();

        // Convert String Text To Numbers
        final double getmeanX =
             NumberFormat.getDecimalFormat().parse(StrmeanX);
        final double getmeanY =
             NumberFormat.getDecimalFormat().parse(StrmeanY);
        final double getsdX =
             NumberFormat.getDecimalFormat().parse(StrsdX);
        final double getsdY =
             NumberFormat.getDecimalFormat().parse(StrsdY);
        final double getcorrel =
             NumberFormat.getDecimalFormat().parse(Strcorrel);
        getsampleSize = (int)
        NumberFormat.getDecimalFormat().parse(StrsampleSize);

        // Define Parameters Of Initial Simulated Variables
        final double[] initialmean = {0, 0};
        final double[] initialsd = {1, 1};

        // Compute Covariance Matrix For Initial Simulated Variables
        double Xvar = initialsd[0] * initialsd[0];
        double Yvar = initialsd[1] * initialsd[1];
        double cov = getcorrel * initialsd[0] * initialsd[1];
        final double[][] covmatrix = { {Xvar, cov}, {cov, Yvar} };

        // Define Parameters Of X and Y
        final double[] actualMean = {getmeanX, getmeanY};
        final double[] actualSd = {getsdX, getsdY};

        // Compute Alpha and Beta for Beta Distribution
        final double alpha = -actualMean[1] * (-actualMean[1] +
         pow(actualMean[1],2) + pow(actualSd[1], 2)) / pow(actualSd[1],2);
        final double beta = (actualMean[1] - 1) * (-actualMean[1] +
         pow(actualMean[1],2) + pow(actualSd[1], 2)) / pow(actualSd[1],2);

        // Compute Parameter Matrix for X and Y
        final double[][] paramatrix =
             { { actualMean[0], actualSd[0] }, { alpha, beta } };

        // Initiate Relevant Variables
        XNormalSampleVector = new double[getsampleSize];
        YNormalSampleVector = new double[getsampleSize];
        NormalSampleVector = new double[getsampleSize];
        // Define Monte Carlo Simulation To Generate Normal Copula Samples
        final int[] count = new int[1];
        final int[] getServerResponse = new int[1];
        count[0] = 0; getServerResponse[0] = 1;

        Timer NormSimulation = new Timer() {
              public void run() {
                  if (count[0] == getsampleSize){

                    outputArea.setText(outputArea.getText() + "\nMean of
                    Population X: " + getmeanX + ", SD of Population X: "
                    + getsdX);
                    outputArea.setText(outputArea.getText() + "\nMean of
                    Population Y: " + getmeanY + ", SD of Population Y: "
                    + getsdY);
                    outputArea.setText(outputArea.getText() + "\nAlpha of
                    Population Y: " + (double)
                    Math.round(alpha*1000000)/1000000 + ", Beta of
                    Population Y: " + (double)
                    Math.round(beta*1000000)/1000000);
                    outputArea.setText(outputArea.getText() +
                    "\n\nDescriptive Stats For Normal Copula (Sample Size:
                    " + getsampleSize + ")");

                    AsyncCallback<double[][]> EstCallback = new
                    AsyncCallback<double[][]>() {

                            public void onFailure(Throwable caught) {
                                outputArea.setText(outputArea.getText() +
                                "Error " + caught + "\n");
                            }

                            public void onSuccess(double[][] result) {
                                NormalSampleParamatrix[0][0] =
                                result[0][0];
                                NormalSampleParamatrix[0][1] =
                                result[0][1];
                                NormalSampleParamatrix[1][0] =
                                result[2][0];
                                NormalSampleParamatrix[1][1] =
                                result[2][1];

                                outputArea.setText(outputArea.getText() +
                                "\nMean of Sample X: " + result[0][0] + ",
                                SD of Sample X: " + result[0][1]);
                                outputArea.setText(outputArea.getText() +
                                "\nMean of Sample Y: " + result[1][0] + ",
                                SD of Sample Y: " + result[1][1]);
                                outputArea.setText(outputArea.getText() +
                                "\nAlpha of Sample Y: " + result[2][0] +
                                ", Beta of Sample Y: " + result[2][1] +
                                "\n\n");
                                getServerResponse[0] = 1;
                                count[0]++;
                            }
                          };

                    mathFunctionsSvc.estimateSampleParamatrix(XNormalSampl
                    eVector, YNormalSampleVector, getsampleSize,
                    EstCallback);
                    cancel(); return;
                  }

                  if(getServerResponse[0] == 0){
                  }
                  else{
                    getServerResponse[0] = 0;
                    AsyncCallback<double[]> GenCallback = new
                        AsyncCallback<double[]>() {

                        public void onFailure(Throwable caught) {
                           outputArea.setText(outputArea.getText() +
                           "Error " + caught + "\n");
                        }

                        public void onSuccess(double[] result) {
                           XNormalSampleVector[count[0]] = result[0];
                           YNormalSampleVector[count[0]] = result[1];
                           NormalSampleVector[count[0]] =  result[2];

                           outputArea.setText(outputArea.getText() + "<" +
                           count[0] + "> X: " + result[0]  + ", Y: " +
                           result[1] + ", X+Y: " + result[2] + "\n");
                           getServerResponse[0] = 1;
                           count[0]++;
                        }
                      };

                    mathFunctionsSvc.GenerateNormSamples(initialmean,
                    covmatrix, paramatrix, GenCallback);
                  }
              }
        };

        // Set Up Timer For Callback Function
        outputArea.setText(outputArea.getText() + "Monte Carlo Simulation:
             Sample Data for Normal Copula \n\n");
        NormSimulation.scheduleRepeating(REFRESH_INTERVAL);
    }

    public void simulateTCopula() {

        // Get Input Text As Strings
        final String StrmeanX = meanX.getText().toUpperCase().trim();
        final String StrmeanY = meanY.getText().toUpperCase().trim();
        final String StrsdX = sdX.getText().toUpperCase().trim();
        final String StrsdY = sdY.getText().toUpperCase().trim();
        final String Strcorrel = correl.getText().toUpperCase().trim();
        final String StrNDegrees =
             NDegrees.getText().toUpperCase().trim();
        final String StrsampleSize =
             sampleSize.getText().toUpperCase().trim();

        // Convert String Text To Numbers
        final double getmeanX =
             NumberFormat.getDecimalFormat().parse(StrmeanX);
        final double getmeanY =
             NumberFormat.getDecimalFormat().parse(StrmeanY);
        final double getsdX =
             NumberFormat.getDecimalFormat().parse(StrsdX);
        final double getsdY =
             NumberFormat.getDecimalFormat().parse(StrsdY);
        final double getcorrel =
             NumberFormat.getDecimalFormat().parse(Strcorrel);
        final double getNDegrees =
             NumberFormat.getDecimalFormat().parse(StrNDegrees);
        getsampleSize =
             (int)NumberFormat.getDecimalFormat().parse(StrsampleSize);

        // Define Parameters Of Initial Simulated Variables
        final double[] initialmean = {0, 0};
        final double[] initialsd = {1, 1};
```

```
        // Compute Covariance Matrix For Initial Simulated Variables
        double Xvar = initialsd[0] * initialsd[0];
        double Yvar = initialsd[1] * initialsd[1];
        double cov = getcorrel * initialsd[0] * initialsd[1];
        final double[][] covmatrix = { {Xvar, cov}, {cov, Yvar} };

        // Define Parameters Of X and Y
        final double[] actualMean = {getmeanX, getmeanY};
        final double[] actualSd = {getsdX, getsdY};

        // Compute Alpha and Beta for Beta Distribution
        final double alpha = -actualMean[1] * (-actualMean[1] +
         pow(actualMean[1],2) + pow(actualSd[1], 2)) / pow(actualSd[1],2);
        final double beta = (actualMean[1] - 1) * (-actualMean[1] +
         pow(actualMean[1],2) + pow(actualSd[1], 2)) / pow(actualSd[1],2);

        // Compute Parameter Matrix for X and Y
        final double[][] paramatrix =
             { { actualMean[0], actualSd[0] }, { alpha, beta } };

        // Initiate Relevant Variables
        XTSampleVector = new double[getsampleSize];
        YTSampleVector = new double[getsampleSize];
        TSampleVector = new double[getsampleSize];

        // Define Monte Carlo Simulation To Generate T Copula Samples
        final int[] count = new int[1];
        final int[] getServerResponse = new int[1];
        count[0] = 0; getServerResponse[0] = 1;

        Timer TSimulation = new Timer() {
              public void run() {
                  if (count[0] == getsampleSize){

                    outputArea.setText(outputArea.getText() + "\nMean of
                    Population X: " + getmeanX + ", SD of Population X: "
                    + getsdX);
                    outputArea.setText(outputArea.getText() + "\nMean of
                    Population Y: " + getmeanY + ", SD of Population Y: "
                    + getsdY);
                    outputArea.setText(outputArea.getText() + "\nAlpha of
                    Population Y: " + (double)
                    Math.round(alpha*1000000)/1000000 + ", Beta of
                    Population Y: " + (double)
                    Math.round(beta*1000000)/1000000);
                    outputArea.setText(outputArea.getText() +
                    "\n\nDescriptive Stats For T Copula (Sample Size: " +
                    getsampleSize + ")");

                    AsyncCallback<double[][]> EstCallback = new
                           AsyncCallback<double[][]>() {

                            public void onFailure(Throwable caught) {
                                  outputArea.setText(outputArea.getText() +
                                  "Error " + caught + "\n");
                            }

                            public void onSuccess(double[][] result) {
                                  TSampleParamatrix[0][0] = result[0][0];
                                  TSampleParamatrix[0][1] = result[0][1];
                                  TSampleParamatrix[1][0] = result[2][0];
                                  TSampleParamatrix[1][1] = result[2][1];

                                  outputArea.setText(outputArea.getText() +
                                  "\nMean of Sample X: " + result[0][0] +
                                  ", SD of Sample X: " + result[0][1]);
```

```
                                   outputArea.setText(outputArea.getText() +
                                   "\nMean of Sample Y: " + result[1][0] +
                                   ", SD of Sample Y: " + result[1][1]);
                                   outputArea.setText(outputArea.getText() +
                                   "\nAlpha of Sample Y: " + result[2][0] +
                                   ", Beta of Sample Y: " + result[2][1] +
                                   "\n\n");
                                   getServerResponse[0] = 1;
                                   count[0]++;
                              }
                            };

                      mathFunctionsSvc.estimateSampleParamatrix(XTSampleVect
                      or, YTSampleVector, getsampleSize, EstCallback);
                      cancel(); return;
                    }

                    if(getServerResponse[0] == 0){
                    }
                    else{
                      getServerResponse[0] = 0;
                      AsyncCallback<double[]> GenCallback = new
                        AsyncCallback<double[]>() {

                          public void onFailure(Throwable caught) {
                             outputArea.setText(outputArea.getText() +
                             "Error " + caught + "\n");
                          }
                          public void onSuccess(double[] result) {
                             XTSampleVector[count[0]] = result[0];
                             YTSampleVector[count[0]] = result[1];

                             TSampleVector[count[0]] =  result[2];
                             outputArea.setText(outputArea.getText() + "<" +
                             count[0] + "> X: " + result[0]  + ", Y: " +
                             result[1] + ", X+Y: " + result[2] + "\n");
                             getServerResponse[0] = 1;
                             count[0]++;
                          }
                        };

                      mathFunctionsSvc.GenerateTSamples(initialmean,
                      covmatrix, paramatrix, getNDegrees, GenCallback);
                    }
               }
         };

         // Set Up Timer For Callback Function
        outputArea.setText(outputArea.getText() + "Monte Carlo Simulation:
              Sample Data for T Copula \n\n");
         TSimulation.scheduleRepeating(REFRESH_INTERVAL);
    }

    public void estimateNormCopula() {

         // Get Input Text As Strings
         final String StrinitialCorrel =
              initialCorrel.getText().toUpperCase().trim();

         // Convert String Text To Numbers
         final double getinitialCorrel =
              NumberFormat.getDecimalFormat().parse(StrinitialCorrel);
         // Define Estimate Parameters (Normal Copula) Callback Function
         AsyncCallback<double[][]> EstCallback = new
             AsyncCallback<double[][]>() {

             public void onFailure(Throwable caught) {
                 outputArea.setText(outputArea.getText() + "Error " +
                      caught + "\n");
             }
             public void onSuccess(double[][] result) {
                 for(int i=1; i<=result[0][0]; i++){
                      outputArea.setText(outputArea.getText() +
                      "<Correlation of " + result[i][0] + "> Log-likelihood:
                      " + result[i][1] + "\n");

                      if(i==result[0][0]){ outputArea.setText(outputArea.get
                      Text() + "\n"); }
                 }
             }
         };

         // Set Up Callback Function
         outputArea.setText(outputArea.getText() + "Computing MLE
              Parameters for Normal Copula... \n\n");
         mathFunctionsSvc.estimateNormParameters(XNormalSampleVector,
            YNormalSampleVector, NormalSampleParamatrix, getinitialCorrel,
            getsampleSize, EstCallback);
    }

    public void estimateTCopula() {

         // Get Input Text As Strings
         final String StrinitialCorrel =
              initialCorrel.getText().toUpperCase().trim();
         final String StrinitialNDegrees =
              initialNDegrees.getText().toUpperCase().trim();

         // Convert String Text To Numbers
         final double getinitialCorrel =
              NumberFormat.getDecimalFormat().parse(StrinitialCorrel);
         final double getinitialNDegrees =
              NumberFormat.getDecimalFormat().parse(StrinitialNDegrees);

         // Define Estimate Parameters (T Copula) Callback Function
         AsyncCallback<double[][]> EstCallback = new
             AsyncCallback<double[][]>() {

             public void onFailure(Throwable caught) {
                 outputArea.setText(outputArea.getText() + "Error " +
                 caught + "\n");
             }

             public void onSuccess(double[][] result) {
                 for(int i=1; i<=result[0][0]; i++){
                 outputArea.setText(outputArea.getText() + "<Correlation of
                 " + result[i][0] + ", df of " + result[i][1] + "> Log-
                 likelihood: " + result[i][2] + "\n");

                 if(i==result[0][0]){
                      outputArea.setText(outputArea.getText() + "\n");
                 }
                 }
             }
         };
         // Set Up Callback Function
         outputArea.setText(outputArea.getText() + "Computing MLE
              Parameters for T Copula... \n\n");
         mathFunctionsSvc.estimateTParameters(XTSampleVector,
              YTSampleVector, TSampleParamatrix, getinitialCorrel,
              getinitialNDegrees, getsampleSize, EstCallback);
    }

    public void calcVaRNormCopula() {

         // Get Input Text As Strings
         final String Strprob = prob.getText().toUpperCase().trim();
         final String StrconfidenceInterval =
              confidenceInterval.getText().toUpperCase().trim();

         // Convert String Text To Numbers
         final double getprob =
              NumberFormat.getDecimalFormat().parse(Strprob);
         final double getconfidenceInterval =
              NumberFormat.getDecimalFormat().parse(StrconfidenceInterval);

         // Define Calculate VaR (Normal Copula) Callback Function
         AsyncCallback<double[]> CalcCallback = new
             AsyncCallback<double[]>() {

             public void onFailure(Throwable caught) {
                 outputArea.setText(outputArea.getText() + "Error " +
                      caught + "\n");
             }

             public void onSuccess(double[] result) {
                 outputArea.setText(outputArea.getText() + "VaR under
                      Normal Copula (Sample Size: " + getsampleSize + ")");
                 outputArea.setText(outputArea.getText() + "\nVaR at " +
                      getprob * 100 + "%: " + result[0]);
                 outputArea.setText(outputArea.getText() + "\nLower Bound
                      of " + getprob * 100 + "% VaR: " + result[1]);
                 outputArea.setText(outputArea.getText() + "\nUpper Bound
                      of " + getprob * 100 + "% VaR: " + result[2] +
                      "\n\n");
             }
         };

         // Set Up Callback Function
         mathFunctionsSvc.calculateVaRForNorm(NormalSampleVector,
              getsampleSize, getprob, getconfidenceInterval, CalcCallback);
    }

    public void calcVaRTCopula() {

         // Get Input Text As Strings
         final String Strprob = prob.getText().toUpperCase().trim();
         final String StrconfidenceInterval =
              confidenceInterval.getText().toUpperCase().trim();

         // Convert String Text To Numbers
         final double getprob =
              NumberFormat.getDecimalFormat().parse(Strprob);
         final double getconfidenceInterval =
              NumberFormat.getDecimalFormat().parse(StrconfidenceInterval);
         // Define Calculate VaR (T Copula) Callback Function
         AsyncCallback<double[]> CalcCallback = new
              AsyncCallback<double[]>() {

             public void onFailure(Throwable caught) {
                 outputArea.setText(outputArea.getText() + "Error " +
                      caught + "\n");
             }

             public void onSuccess(double[] result) {
                 outputArea.setText(outputArea.getText() + "VaR under T
                      Copula (Sample Size: " + getsampleSize + ")");
                 outputArea.setText(outputArea.getText() + "\nVaR at " +
                      getprob * 100 + "%: " + result[0]);
                 outputArea.setText(outputArea.getText() + "\nLower Bound
                      of " + getprob * 100 + "% VaR: " + result[1]);
                 outputArea.setText(outputArea.getText() + "\nUpper Bound
                      of " + getprob * 100 + "% VaR: " + result[2] +
                      "\n\n");
             }
         };

         // Set Up Callback Function
         mathFunctionsSvc.calculateVaRForT(TSampleVector, getsampleSize,
              getprob, getconfidenceInterval, CalcCallback);
    }

    public void calcESNormCopula() {

         // Get Input Text As Strings
         final String Strprob = prob.getText().toUpperCase().trim();
         final String StrconfidenceInterval =
              confidenceInterval.getText().toUpperCase().trim();

         // Convert String Text To Numbers
         final double getprob =
              NumberFormat.getDecimalFormat().parse(Strprob);
         final double getconfidenceInterval =
              NumberFormat.getDecimalFormat().parse(StrconfidenceInterval);

         // Define Calculate ES (Normal Copula) Callback Function
         AsyncCallback<double[]> CalcCallback = new
             AsyncCallback<double[]>() {

             public void onFailure(Throwable caught) {
                 outputArea.setText(outputArea.getText() + "Error " +
                      caught + "\n");
             }

             public void onSuccess(double[] result) {
                 outputArea.setText(outputArea.getText() + "Expected
                      Shortfall under Normal Copula (Sample Size: " +
                      getsampleSize + ")");
                 outputArea.setText(outputArea.getText() + "\nExpected
                      Shortfall at " + getprob * 100 + "%: " + result[0]);
                 outputArea.setText(outputArea.getText() + "\nLower Bound
                      of " + getprob * 100 + "% ES: " + result[1]);
                 outputArea.setText(outputArea.getText() + "\nUpper Bound
                      of " + getprob * 100 + "% ES: " + result[2] + "\n\n");
             }
         };

         // Set Up Callback Function
         mathFunctionsSvc.calculateESForNorm(NormalSampleVector,
              getsampleSize, getprob, getconfidenceInterval, CalcCallback);
    }
    public void calcESTCopula() {

         // Get Input Text As Strings
         final String Strprob = prob.getText().toUpperCase().trim();
         final String StrconfidenceInterval =
              confidenceInterval.getText().toUpperCase().trim();

       // Convert String Text To Numbers
         final double getprob =
              NumberFormat.getDecimalFormat().parse(Strprob);
         final double getconfidenceInterval =
              NumberFormat.getDecimalFormat().parse(StrconfidenceInterval);

         // Define Calculate ES (T Copula) Callback Function
         AsyncCallback<double[]> CalcCallback = new
             AsyncCallback<double[]>() {

             public void onFailure(Throwable caught) {
                 outputArea.setText(outputArea.getText() + "Error " +
                      caught + "\n");
             }

             public void onSuccess(double[] result) {
                 outputArea.setText(outputArea.getText() + "Expected
                      Shortfall under T Copula (Sample Size: " +
                      getsampleSize + ")");
                 outputArea.setText(outputArea.getText() + "\nExpected
                      Shortfall at " + getprob * 100 + "%: " + result[0]);
                 outputArea.setText(outputArea.getText() + "\nLower Bound
                      of " + getprob * 100 + "% ES: " + result[1]);
                 outputArea.setText(outputArea.getText() + "\nUpper Bound
                      of " + getprob * 100 + "% ES: " + result[2] + "\n\n");
             }
         };

         // Set Up Callback Function
         mathFunctionsSvc.calculateESForT(TSampleVector, getsampleSize,
              getprob, getconfidenceInterval, CalcCallback);
    }
}
```

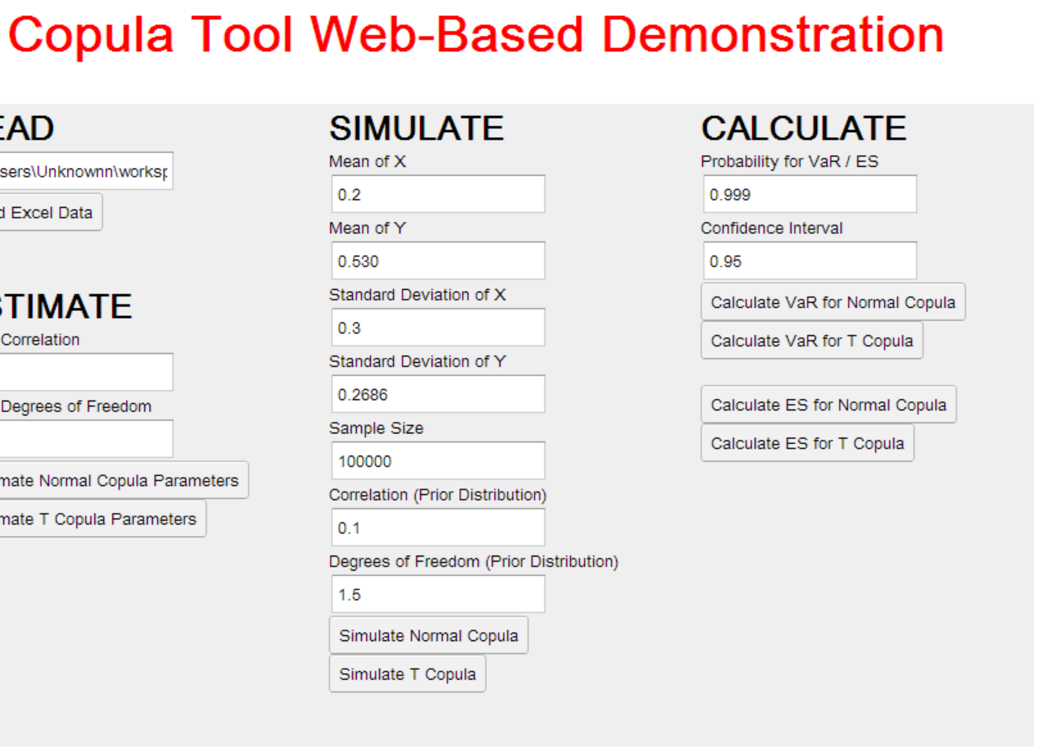
Server-side Applet Code Sample: MathFunctionsServiceImpl.java
(Additional Details Are Elaborated In The Comments For Readability)

```
package web.copulatool.server;

//Import Relevant Java Packages
import java.io.*;
import static java.lang.Math.*;
import java.text.DecimalFormat;
import org.apache.commons.math3.linear.*;
import org.apache.commons.math3.distribution.*;
import org.apache.commons.math3.random.*;
import org.apache.commons.math3.stat.descriptive.*;
import org.apache.commons.math3.analysis.*;
import org.apache.commons.math3.optimization.*;
import org.apache.poi.xssf.usermodel.*;
import org.apache.poi.ss.usermodel.*;

import web.copulatool.client.MathFunctionsService;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class MathFunctionsServiceImpl extends RemoteServiceServlet
implements MathFunctionsService {

        public double[][] readExcelData(String fileaddress) {

            // Initiate Dummy Variable
            double[][] a = new double[1][1];
            try{
                // Initiate Variables For Excel File Reading
                FileInputStream file = new FileInputStream(fileaddress);
                XSSFWorkbook workbook = new XSSFWorkbook(file);
                XSSFSheet worksheet = workbook.getSheetAt(0);
                int numberOfData = worksheet.getPhysicalNumberOfRows();
                double[][] resultArray = new double[numberOfData][3];
                resultArray[0][0] = numberOfData;
                int count = 0;
                DecimalFormat df = new DecimalFormat("#.######");

                // Extract Values For X And Y On Excel
                for (Row row : worksheet) {
                    for (Cell cell : row) {
                        if (count > 0) {
                            double value = cell.getNumericCellValue();
                            resultArray[count][cell.getColumnIndex()] =
                                 Double.parseDouble(df.format(value));
                        }
                    }

                // Compute Values For X + Y
                resultArray[count][2] =
                    Double.parseDouble(df.format(resultArray[count][0] +
                    resultArray[count][1]));
                count++;
                }

                // Close File And Return
                file.close();
                return resultArray;

            } catch (FileNotFoundException e) {
                // Return Error
                return a;
            } catch (IOException e) {
                // Return Error
                return a;
            }
        }

        public double[] GenerateNormSamples(double[] mean,
            double[][] covmatrix, double[][] paramatrix) {

            // Create Random Gaussian Variable
            RealMatrix covariance =
                MatrixUtils.createRealMatrix(covmatrix);
            RandomGenerator rg = new Well19937a();
            GaussianRandomGenerator rawGen = new
                GaussianRandomGenerator(rg);

            // Create Another Correlated Gaussian Variable
            CorrelatedRandomVectorGenerator generator =
                new CorrelatedRandomVectorGenerator(mean, covariance,
                    1.0e-12 * covariance.getNorm(), rawGen);

            // Store Correlated Vector
            double[] randomVector = generator.nextVector();

            // Implement First Transform
            double[] newVector = new double[2];
            NormalDistribution n = new NormalDistribution();
            newVector[0] = n.cumulativeProbability(randomVector[0]);
            newVector[1] = n.cumulativeProbability(randomVector[1]);

            // Implement Second Transform
            double[] finalVector = new double[2];
            NormalDistribution n2 = new
                NormalDistribution(paramatrix[0][0], paramatrix[0][1]);
            BetaDistribution b = new
                BetaDistribution(paramatrix[1][0], paramatrix[1][1]);
            finalVector[0] =
                n2.inverseCumulativeProbability(newVector[0]);
            finalVector[1] = b.inverseCumulativeProbability(newVector[1]);

            // Implement Rounding Off And Compute Values For X + Y
            double[] roundedfinalVector = new double[3];
            DecimalFormat df = new DecimalFormat("#.######");
            roundedfinalVector[0] =
                Double.parseDouble(df.format(finalVector[0]));
            roundedfinalVector[1] =
                Double.parseDouble(df.format(finalVector[1]));
            roundedfinalVector[2] =
             Double.parseDouble(df.format(finalVector[0]+finalVector[1]));

            // Return
            return roundedfinalVector;
        }

        public double[] GenerateTSamples(double[] mean,
           double[][] covmatrix, double [][] paramatrix, double nDegree) {

            // Create Random Gaussian Variable
            RealMatrix covariance =
                MatrixUtils.createRealMatrix(covmatrix);
            RandomGenerator rg = new Well19937a();
            GaussianRandomGenerator rawGen = new
                GaussianRandomGenerator(rg);
            // Create Another Correlated Gaussian Variable
            CorrelatedRandomVectorGenerator generator =
                new CorrelatedRandomVectorGenerator(mean, covariance,
                    1.0e-12 * covariance.getNorm(), rawGen);

            // Store Correlated Vector
            double[] randomVector = generator.nextVector();

            // Implement First Transform
            double[] newVector = new double[2];
            RandomGenerator rg2 = new Well19937a();
            RandomDataGenerator rawGen2 = new RandomDataGenerator(rg2);
            double chiRandom = rawGen2.nextChiSquare(nDegree);
            newVector[0] = randomVector[0] * sqrt(nDegree/chiRandom);
            newVector[1] = randomVector[1] * sqrt(nDegree/chiRandom);

            // Implement Second Transform
            double[] secondVector = new double[2];
            TDistribution t = new TDistribution(nDegree);
            secondVector[0] = t.cumulativeProbability(newVector[0]);
            secondVector[1] = t.cumulativeProbability(newVector[1]);

            // Implement Third Transform
            double[] finalVector = new double[2];
            NormalDistribution n = new
                NormalDistribution(paramatrix[0][0], paramatrix[0][1]);
            BetaDistribution b = new
                BetaDistribution(paramatrix[1][0], paramatrix[1][1]);
            finalVector[0] =
                n.inverseCumulativeProbability(secondVector[0]);
            finalVector[1] =
                b.inverseCumulativeProbability(secondVector[1]);

            // Implement Rounding Off And Compute Values For X + Y
            double[] roundedfinalVector = new double[3];
            DecimalFormat df = new DecimalFormat("#.######");
            roundedfinalVector[0] =
                Double.parseDouble(df.format(finalVector[0]));
            roundedfinalVector[1] =
                Double.parseDouble(df.format(finalVector[1]));
            roundedfinalVector[2] =
             Double.parseDouble(df.format(finalVector[0]+finalVector[1]));

            // Return
            return roundedfinalVector;
        }

        public double[] calculateVaRForNorm(double[] sampleVector,
            int sampleSize, double prob, double confidenceInterval) {

            // Create DescriptiveStatisitics Variable
            DescriptiveStatistics normStats = new DescriptiveStatistics();
            for(int i = 0; i < sampleSize; i++) {
                normStats.addValue(sampleVector[i]);
            }

            // Compute VaR
            NormalDistribution n = new NormalDistribution();
            double normVaR = normStats.getPercentile((1 - prob) * 100);

            // Compute Lower and Upper Bound for VaR
            double r = (sampleSize * (1 - prob)) -
               (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
                * sqrt(sampleSize * prob * (1 - prob));
            double s = (sampleSize * (1 - prob)) +
              (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
                * sqrt(sampleSize * prob * (1 - prob));
            double lowerBound =
                normStats.getPercentile(r/sampleSize * 100);
            double upperBound =
                normStats.getPercentile(s/sampleSize * 100);

            // Implement Rounding Off
            double[] VaRVector = new double[3];
            DecimalFormat df = new DecimalFormat("#.######");
            VaRVector[0] = Double.parseDouble(df.format(normVaR));
            VaRVector[1] = Double.parseDouble(df.format(lowerBound));
            VaRVector[2] = Double.parseDouble(df.format(upperBound));

            // Return
            return VaRVector;
        }

        public double[] calculateVaRForT(double[] sampleVector,
            int sampleSize, double prob, double confidenceInterval) {

            // Create DescriptiveStatisitics Variable
            DescriptiveStatistics TStats = new DescriptiveStatistics();
            for(int i = 0; i < sampleSize; i++) {
                TStats.addValue(sampleVector[i]);
            }

            // Compute VaR
            NormalDistribution n = new NormalDistribution();
            double normVaR = TStats.getPercentile((1 - prob) * 100);

            // Compute Lower and Upper Bound for VaR
            double r = (sampleSize * (1 - prob)) -
               (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
                * sqrt(sampleSize * prob * (1 - prob));
            double s = (sampleSize * (1 - prob)) +
               (n.inverseCumulativeProbability((1 + confidenceInterval)/2)
                * sqrt(sampleSize * prob * (1 - prob));
            double lowerBound = TStats.getPercentile(r/sampleSize * 100);
            double upperBound = TStats.getPercentile(s/sampleSize * 100);

            // Implement Rounding Off
            double[] VaRVector = new double[3];
            DecimalFormat df = new DecimalFormat("#.######");
            VaRVector[0] = Double.parseDouble(df.format(normVaR));
            VaRVector[1] = Double.parseDouble(df.format(lowerBound));
            VaRVector[2] = Double.parseDouble(df.format(upperBound));

            // Return
            return VaRVector;
        }

        public double[] calculateESForNorm(double[] sampleVector,
            int sampleSize, double prob, double confidenceInterval) {

            // Create DescriptiveStatisitics Variable
            DescriptiveStatistics normStats = new DescriptiveStatistics();
            for(int i = 0; i < sampleSize; i++) {
                normStats.addValue(sampleVector[i]);
            }

            // Compute VaR
            NormalDistribution n = new NormalDistribution();
            double normVaR = normStats.getPercentile((1 - prob) * 100);

            // Compute Shortfall Vectors
            int count = 0;
            double [] shortfallVector = new double[sampleSize];
            for(int i = 0; i < sampleSize; i++) {
                if(sampleVector[i] <= normVaR){
                    shortfallVector[count] = sampleVector[i];
                    count++;
                }

            // Compute Expected Shortfall
            double sumShortfall = 0;
            for(int j=0; j < count; j++) {
                sumShortfall = shortfallVector[j] + sumShortfall;
            }
            double expectedShortfall = sumShortfall / count;

            // Compute Variance of Expected Shortfall
            double squareShortfallDiff = 0;
            for(int k=0; k < count; k++) {
                squareShortfallDiff = pow((shortfallVector[k]
                    expectedShortfall), 2) + squareShortfallDiff;
            }
            double A = squareShortfallDiff / (count-1);
            double B = (1 - prob) * pow((expectedShortfall - normVaR), 2);
            double varianceExpectedShortfall = (A + B) / count;

            // Compute Lower and Upper Bounds for Expected Shortfall
            double lowerBound = expectedShortfall -
              (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
               * sqrt(varianceExpectedShortfall);
            double upperBound = expectedShortfall +
              (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
               * sqrt(varianceExpectedShortfall);

            // Implement Rounding Off
            double[] ESVector = new double[3];
            DecimalFormat df = new DecimalFormat("#.######");
            ESVector[0] =
                 Double.parseDouble(df.format(expectedShortfall));
            ESVector[1] = Double.parseDouble(df.format(lowerBound));
            ESVector[2] = Double.parseDouble(df.format(upperBound));

            // Return
            return ESVector;
        }

        public double[] calculateESForT(double[] sampleVector,
            int sampleSize, double prob, double confidenceInterval) {

            // Create DescriptiveStatisitics Variable
            DescriptiveStatistics TStats = new DescriptiveStatistics();
            for(int i = 0; i < sampleSize; i++) {
                TStats.addValue(sampleVector[i]);
            }

            // Compute VaR
            NormalDistribution n = new NormalDistribution();
            double normVaR = TStats.getPercentile((1 - prob) * 100);

            // Compute Shortfall Vectors
            int count = 0;
            double [] shortfallVector = new double[sampleSize];
            for(int i = 0; i < sampleSize; i++) {
                if(sampleVector[i] <= normVaR){
                    shortfallVector[count] = sampleVector[i];
                    count++;
                }
            }
            // Compute Expected Shortfall
            double sumShortfall = 0;
            for(int j=0; j < count; j++) {
                sumShortfall = shortfallVector[j] + sumShortfall;
            }
            double expectedShortfall = sumShortfall / count;

            // Compute Variance of Expected Shortfall
            double squareShortfallDiff = 0;
            for(int k=0; k < count; k++) {
                squareShortfallDiff = pow((shortfallVector[k] -
                    expectedShortfall), 2) + squareShortfallDiff;
            }
            double A = squareShortfallDiff / (count-1);
            double B = (1 - prob) * pow((expectedShortfall - normVaR), 2);
            double varianceExpectedShortfall = (A + B) / count;

            // Compute Lower and Upper Bounds for Expected Shortfall
            double lowerBound = expectedShortfall
              (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
               * sqrt(varianceExpectedShortfall);
            double upperBound = expectedShortfall +
              (n.inverseCumulativeProbability((1 + confidenceInterval)/2))
               * sqrt(varianceExpectedShortfall);

            // Implement Rounding Off
            double[] ESVector = new double[3];
            DecimalFormat df = new DecimalFormat("#.######");
            ESVector[0] =
                Double.parseDouble(df.format(expectedShortfall));
            ESVector[1] = Double.parseDouble(df.format(lowerBound));
            ESVector[2] = Double.parseDouble(df.format(upperBound));

            // Return
            return ESVector;
        }

        public double[][] estimateSampleParamatrix(double[] XSampleVector,
            double[] YSampleVector, int sampleSize) {

            // Compute Sum And Sum Of Squares For X And Y
            double totalX1 = 0; double totalX2 = 0;
            double totalY1 = 0; double totalY2 = 0;
            for(int i = 0; i < sampleSize; i++) {
                totalX1 = totalX1 + XSampleVector[i];
                totalY1 = totalY1 + YSampleVector[i];
                totalX2 = totalX2 + pow(XSampleVector[i], 2);
                totalY2 = totalY2 + pow(YSampleVector[i], 2);
            }

            // Compute Average of Sum And Sum Of Squares For X And Y
            double mX1 = totalX1 / sampleSize;
            double mX2 = totalX2 / sampleSize;
            double mY1 = totalY1 / sampleSize;
            double mY2 = totalY2 / sampleSize;

            // Compute Mean And SD For X And Y
            double meanX = mX1;
            double meanY = mY1;
            double sdX = sqrt (mX2 - pow(meanX, 2));
            double sdY = sqrt (mY2 - pow(meanY, 2));

            // Compute Alpha And Beta For Y
            double mY1square = pow(mY1, 2);
            double alphaY = (((mY1 * mY2) - mY1square)/(mY1square - mY2));
            double betaY = alphaY * (1 - mY1)/mY1;
            // Implement Rounding Off
            DecimalFormat df = new DecimalFormat("#.######");
            double roundedmeanX = Double.parseDouble(df.format(meanX));
            double roundedmeanY = Double.parseDouble(df.format(meanY));
            double roundedsdX = Double.parseDouble(df.format(sdX));
            double roundedsdY = Double.parseDouble(df.format(sdY));
            double roundedalphaY = Double.parseDouble(df.format(alphaY));
            double roundedbetaY = Double.parseDouble(df.format(betaY));

            // Return
            double[][] sampleParamatrix = { {roundedmeanX, roundedsdX},
              {roundedmeanY, roundedsdY}, {roundedalphaY, roundedbetaY} };
            return sampleParamatrix;
        }

        public double[][] estimateNormParameters(double[] XSampleVector,
            double[] YSampleVector, double[][] sampleParamatrix,
            double initialCorrel, final int sampleSize) {

            // Generate Normal And Beta Distributions
            final double[] u = new double[sampleSize];
            final double[] v = new double[sampleSize];
            NormalDistribution n = new
                 NormalDistribution(sampleParamatrix[0][0],
                    sampleParamatrix[0][1]);
            BetaDistribution b = new
                 BetaDistribution(sampleParamatrix[1][0],
                    sampleParamatrix[1][1]);

            // Implement Transform
            for (int i = 0; i<sampleSize; i++){
                u[i] = n.cumulativeProbability(XSampleVector[i]);
                v[i] = b.cumulativeProbability(YSampleVector[i]);
            }

            // Initiate Variables For MLE
            int maxIterations = 10000;
            final double[][] logLikelihoodSet = new
                 double[maxIterations][2];
            final int[] count = new int[1];
            count[0] = 0;

            // Define MLE Function For Normal Copula
            class MaxLikelihoodFunction implements MultivariateFunction {
                public double value(double[] variables) {

                    NormalDistribution n = new NormalDistribution();
                    double newlogLikelihood = 0.0;
                    double A = 2 * 3.14159 *
                           sqrt(1 - pow(variables[0], 2));
                    double[] U = new double[sampleSize];
                    double[] V = new double[sampleSize];

                    for (int i = 0; i<sampleSize; i++){
                        U[i] = n.inverseCumulativeProbability(u[i]);
                        V[i] = n.inverseCumulativeProbability(v[i]);
                        double B = n.cumulativeProbability(U[i]);
                        double C = n.cumulativeProbability(V[i]);
                        double D = pow(U[i], 2) - (2 * variables[0]
                           * U[i] * V[i]) + pow(V[i], 2);
                        double E = -2 * (1 - pow(variables[0], 2));
                        double F = exp(D/E) *  (1/(A * B * C));
                        newlogLikelihood =
                           java.lang.Math.log(F) + newlogLikelihood;
                    }
```

```
                    logLikelihoodSet[0][0] = ++count[0];
                    DecimalFormat df = new DecimalFormat("#.######");
                    logLikelihoodSet[count[0]][0] =
                        Double.parseDouble(df.format(variables[0]));
                    logLikelihoodSet[count[0]][1] =
                        Double.parseDouble(df.format(newlogLikelihood));

                    return newlogLikelihood;
                }
            }

            // Set Up Optimizer
            double[] variables = new double[2];
            variables[0] = initialCorrel;
            double[] bestCorrel = null;
            MaxLikelihoodFunction objectiveFunction = new
                MaxLikelihoodFunction();
           org.apache.commons.math3.optimization.direct.BOBYQAOptimizer
                optimizer = new org.apache.commons.math3.optimization.
                    direct.BOBYQAOptimizer(5, 0.1, 0.00001);

            // Alternative CMAES Optimizer
            // org.apache.commons.math3.optimization.direct.CMAESOptimizer
                    optimizer = new org.apache.commons.math3.optimization.
                    direct.CMAESOptimizer();

            // Implement Optimizer
            PointValuePair result = optimizer.optimize(maxIterations,
                objectiveFunction, GoalType.MAXIMIZE, variables);
            bestCorrel = result.getPoint();

            // Return
            return logLikelihoodSet;
        }

        public double[][] estimateTParameters(double[] XSampleVector,
            double[] YSampleVector, double[][] sampleParamatrix,
            double initialCorrel, double initialNDegrees,
            final int sampleSize) {

            // Generate Normal And Beta Distributions
            final double[] u = new double[sampleSize];
            final double[] v = new double[sampleSize];
            NormalDistribution n = new
                NormalDistribution(sampleParamatrix[0][0],
                    sampleParamatrix[0][1]);
            BetaDistribution b = new
                BetaDistribution(sampleParamatrix[1][0],
                    sampleParamatrix[1][1]);

            // Implement Transform
            for (int i = 0; i<sampleSize; i++){
                u[i] = n.cumulativeProbability(XSampleVector[i]);
                v[i] = b.cumulativeProbability(YSampleVector[i]);
            }

            // Initiate Variables For MLE
            int maxIterations = 10000;
            final double[][] logLikelihoodSet = new
                double[maxIterations][3];
            final int[] count = new int[1];
            count[0] = 0;
            class MaxLikelihoodFunction implements MultivariateFunction {
                public double value(double[] variables) {

                    TDistribution t = new TDistribution(variables[1]);
                    double newlogLikelihood = 0.0;
                    double A = 2 * 3.14159 *
                           sqrt(1 - pow(variables[0], 2));
                    double[] U = new double[sampleSize];
                    double[] V = new double[sampleSize];

                    for (int i = 0; i<sampleSize; i++){
                        U[i] = t.inverseCumulativeProbability(u[i]);
                        V[i] = t.inverseCumulativeProbability(v[i]);
                        double B = t.cumulativeProbability(U[i]);
                        double C = t.cumulativeProbability(V[i]);
                        double D = pow(U[i], 2) - (2 * variables[0]
                           * U[i] * V[i]) + pow(V[i], 2);
                        double E = variables[1]
                           * (1 - pow(variables[0], 2));
                        double F = -(variables[1] + 2)/2;
                        double G = pow((1 + (D/E)), F) *  (1/(A * B * C));
                        newlogLikelihood =
                           java.lang.Math.log(G) + newlogLikelihood;
                    }

                    logLikelihoodSet[0][0] = ++count[0];
                    DecimalFormat df = new DecimalFormat("#.######");
                    logLikelihoodSet[count[0]][0] =
                        Double.parseDouble(df.format(variables[0]));
                    logLikelihoodSet[count[0]][1] =
                        Double.parseDouble(df.format(variables[1]));
                    logLikelihoodSet[count[0]][2] =
                        Double.parseDouble(df.format(newlogLikelihood));
                    return newlogLikelihood;
                }
            }

            // Set Up Optimizer
            double[] variables = new double[2];
            variables[0] = initialCorrel;
            variables[1] = initialNDegrees;
            double[] bestCorrel = null;
            MaxLikelihoodFunction objectiveFunction = new
                 MaxLikelihoodFunction();
            org.apache.commons.math3.optimization.direct.BOBYQAOptimizer
                 optimizer = new org.apache.commons.math3.optimization.
                     direct.BOBYQAOptimizer(5, 0.1, 0.00001);

            // Alternative CMAES Optimizer
            // org.apache.commons.math3.optimization.direct.CMAESOptimizer
                    optimizer = new org.apache.commons.math3.optimization.
                    direct.CMAESOptimizer();

            // Implement Optimizer
            PointValuePair result = optimizer.optimize(maxIterations,
                    objectiveFunction, GoalType.MAXIMIZE, variables);
            bestCorrel = result.getPoint();

            // Return
            return logLikelihoodSet;
        }
}
```

4.2 Web Applet Interface and Results

The final user interface is displayed as follows. Its design is clean and minimalistic, so that to make it user-friendly. Its four main significant features on the control panel are reading, estimating, simulating and calculating. It gives the user ease of access and implementation once the applet can be loaded on a simple website. Moreover, its algorithm is improved and designed for efficiency to minimize the huge burden on computational time.



Copula Tool Web-Based Demonstration

READ

C:\Users\Unknownn\worksp

ESTIMATE

Initial Correlation: 0.1

Initial Degrees of Freedom: 1.5

SIMULATE

Mean of X: 0.2

Mean of Y: 0.530

Standard Deviation of X: 0.3

Standard Deviation of Y: 0.2686

Sample Size: 100000

Correlation (Prior Distribution): 0.1

Degrees of Freedom (Prior Distribution): 1.5

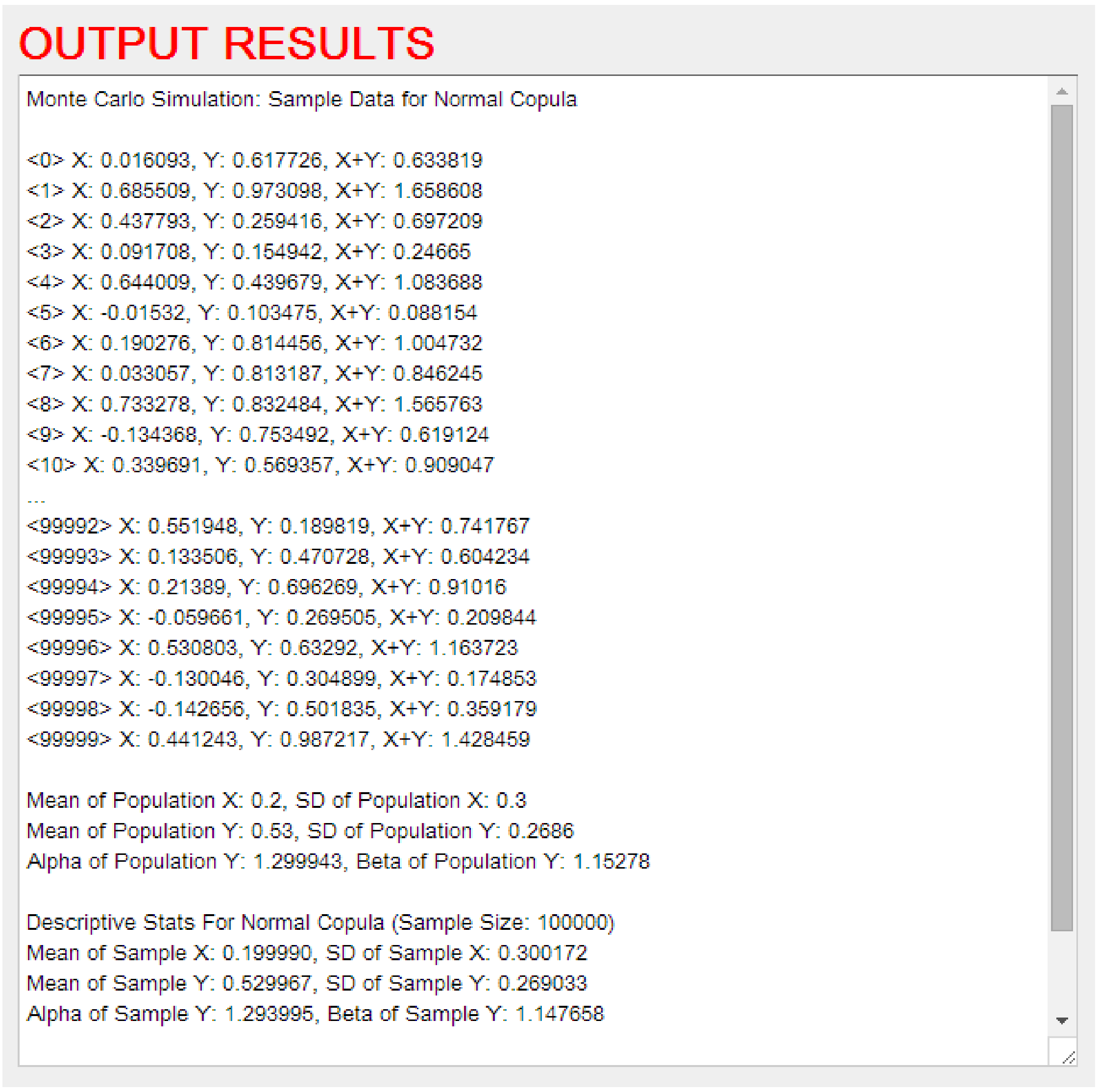
CALCULATE

Probability for VaR / ES: 0.999

Confidence Interval: 0.95

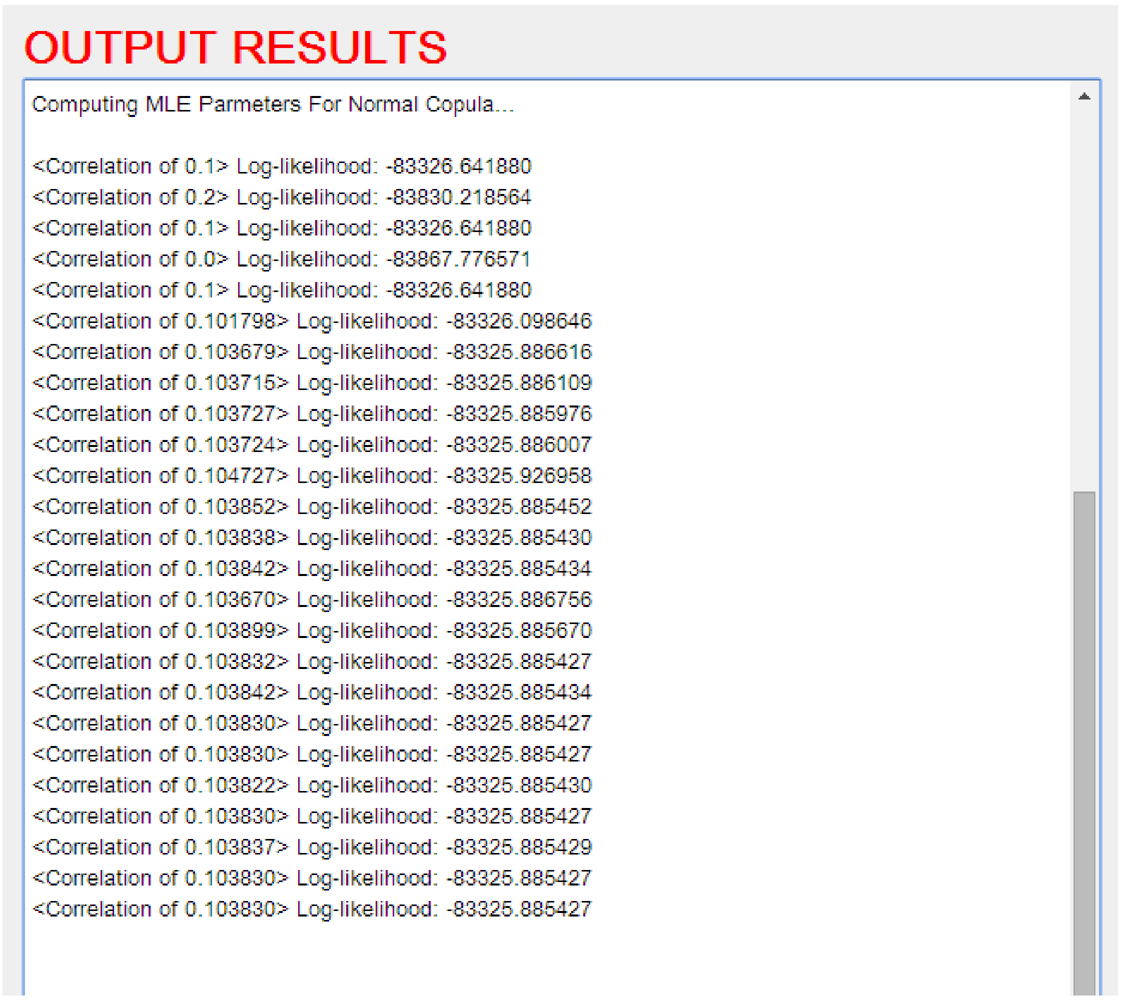
Read Excel Data
Estimate Normal Copula Parameters
Estimate T Copula Parameters
Simulate Normal Copula
Simulate T Copula
Calculate VaR for Normal Copula
Calculate VaR for T Copula
Calculate ES for Normal Copula
Calculate ES for T Copula

The output results for simulating 100,000 samples of X and Y following normal copula with correlation of 0.1 is generated. The computational runtime required is 13846.93 seconds. The values for the sample descriptive statistics (mean and standard deviations of both distributions) are fairly close to its population statistics. This shows that our simulation process is accurate in providing a realistic representation of our desired model.

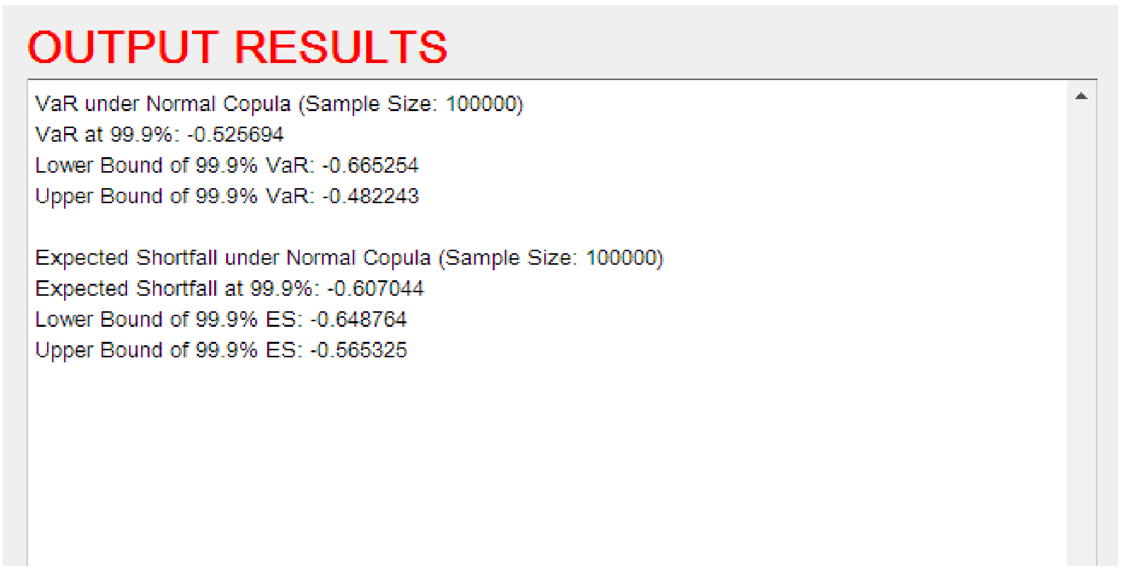


```
Monte Carlo Simulation: Sample Data for Normal Copula

<0> X: 0.016093, Y: 0.617726, X+Y: 0.633819
<1> X: 0.685509, Y: 0.973098, X+Y: 1.658608
<2> X: 0.437793, Y: 0.259416, X+Y: 0.697209
<3> X: 0.091708, Y: 0.154942, X+Y: 0.24665
<4> X: 0.644009, Y: 0.439679, X+Y: 1.083688
<5> X: -0.01532, Y: 0.103475, X+Y: 0.088154
<6> X: 0.190276, Y: 0.814456, X+Y: 1.004732
<7> X: 0.033057, Y: 0.813187, X+Y: 0.846245
<8> X: 0.733278, Y: 0.832484, X+Y: 1.565763
<9> X: -0.134368, Y: 0.753492, X+Y: 0.619124
<10> X: 0.339691, Y: 0.569357, X+Y: 0.909047
...
<99992> X: 0.551948, Y: 0.189819, X+Y: 0.741767
<99993> X: 0.133506, Y: 0.470728, X+Y: 0.604234
<99994> X: 0.21389, Y: 0.696269, X+Y: 0.91016
<99995> X: -0.059661, Y: 0.269505, X+Y: 0.209844
<99996> X: 0.530803, Y: 0.63292, X+Y: 1.163723
<99997> X: -0.130046, Y: 0.304899, X+Y: 0.174853
<99998> X: -0.142656, Y: 0.501835, X+Y: 0.359179
<99999> X: 0.441243, Y: 0.987217, X+Y: 1.428459

Mean of Population X: 0.2, SD of Population X: 0.3
Mean of Population Y: 0.53, SD of Population Y: 0.2686
Alpha of Population Y: 1.299943, Beta of Population Y: 1.15278

Descriptive Stats For Normal Copula (Sample Size: 100000)
Mean of Sample X: 0.199990, SD of Sample X: 0.300172
Mean of Sample Y: 0.529967, SD of Sample Y: 0.269033
Alpha of Sample Y: 1.293995, Beta of Sample Y: 1.147658
```

The maximum likelihood estimation for the correlation parameter of the normal copula is performed. In the example, we find that the final estimated correlation of 0.103830 is close to our initial predefined population correlation of 0.1. This shows that our estimation process for the normal copula works well with our own sample data and has good accuracy. The 99.9% VaR and expected shortfall with their 95% confidence intervals for the sample are also separately determined. As expected, the expected shortfall is observed to be slightly more negative than the VaR.



```
Computing MLE Parmeters For Normal Copula...

<Correlation of 0.1> Log-likelihood: -83326.641880
<Correlation of 0.2> Log-likelihood: -83830.218564
<Correlation of 0.1> Log-likelihood: -83326.641880
<Correlation of 0.0> Log-likelihood: -83867.776571
<Correlation of 0.1> Log-likelihood: -83326.641880
<Correlation of 0.101798> Log-likelihood: -83326.098646
<Correlation of 0.103679> Log-likelihood: -83325.886616
<Correlation of 0.103715> Log-likelihood: -83325.886109
<Correlation of 0.103727> Log-likelihood: -83325.885976
<Correlation of 0.103724> Log-likelihood: -83325.886007
<Correlation of 0.104727> Log-likelihood: -83325.926958
<Correlation of 0.103852> Log-likelihood: -83325.885452
<Correlation of 0.103838> Log-likelihood: -83325.885430
<Correlation of 0.103842> Log-likelihood: -83325.885434
<Correlation of 0.103670> Log-likelihood: -83325.886756
<Correlation of 0.103899> Log-likelihood: -83325.885670
<Correlation of 0.103832> Log-likelihood: -83325.885427
<Correlation of 0.103842> Log-likelihood: -83325.885434
<Correlation of 0.103830> Log-likelihood: -83325.885427
<Correlation of 0.103830> Log-likelihood: -83325.885427
<Correlation of 0.103822> Log-likelihood: -83325.885430
<Correlation of 0.103830> Log-likelihood: -83325.885427
<Correlation of 0.103837> Log-likelihood: -83325.885429
<Correlation of 0.103830> Log-likelihood: -83325.885427
<Correlation of 0.103830> Log-likelihood: -83325.885427
```



```
VaR under Normal Copula (Sample Size: 100000)
VaR at 99.9%: -0.525694
Lower Bound of 99.9% VaR: -0.665254
Upper Bound of 99.9% VaR: -0.482243

Expected Shortfall under Normal Copula (Sample Size: 100000)
Expected Shortfall at 99.9%: -0.607044
Lower Bound of 99.9% ES: -0.648764
Upper Bound of 99.9% ES: -0.565325
```

5. ANALYSIS AND DISCUSSION

5.1 Optimization Methods for MLE

In our investigation of the best optimization approaches to maximize our objective log-likelihood function, we compared the accuracies of two leading advanced and well-known methods to improve our estimation model.

The first iterative algorithm we applied is BOBYQA (Bound Optimization By Quadratic Approximation) developed by Michael J. D. Powell. BOBYQA is a derivative-free method which solves bound constrained optimization problems using a trust region. It seeks to find the minimum of a function $F(x)$, $x \in R_n$, subject to the criteria of $a \leq x \leq b$ on the variables, F being specified as a black box that returns the value $F(x)$ for any feasible x. Each iteration employs a quadratic approximation Q to F that satisfies $Q(y_i) = F(y_i)$, $i = 1, 2, \ldots, m$, the interpolation points y_i being chosen and adjusted automatically, but m is prescribed constant, the value $m = 2n + 1$ being typical.

If the adequate model of the function is found within the trust region, then the region is expanded and conversely, if the estimation is poor, then the region is contracted. The criteria results in a fit with the model function being trusted only in the region where it provides a reasonable approximation. The model is continuously updated by the highly successful technique of minimizing the Frobenius norm of the change to the second derivative matrix of Q. No first derivatives of F are required explicitly, which makes it highly efficient and fast for computation.

The second advanced algorithm of interest is CMA-ES (Covariance Matrix Adaptation Evolution Strategy). It is a stochastic, derivative-free numerical method of non-linear or non-convex continuous optimization problems. Inspired by concepts in biological evolution, it models the repeated interplay of variation (via recombination and mutation) and selection (or iteration) in each generation of new candidate solutions denoted by x, which are generated by current parental individuals. Some individuals are chosen to become parents in the next generation based on their fitness or objective function value $f(x)$. Over the sequence, individuals with better and better f-values are produced.

In the evolution strategy, new candidate solutions are sampled according to a multivariate normal distribution. Recombination refers to selecting a new mean value for the distribution. Mutation refers to adding a random vector, a perturbation with zero mean. Pairwise dependencies between the variables in the distribution are represented by a covariance matrix. Adaptation of the covariance matrix is the learning of a second order model of the underlying objective function similar to the approximation of the inverse Hessian matrix in the quasi-Newton method in classical optimization.

The first part of the algorithm involves applying the adaptation of parameters of the search distribution. The mean is continuously updated such that the likelihood of previously successful candidate solutions is maximized. The covariance matrix of the distribution is updated incrementally such that the likelihood of previously successful candidate solutions is increased. Both updates can be interpreted as a natural gradient descent. In consequence, the CMA conducts an iterated principal components analysis of successful search steps while retaining all principal axes.

The second part of the algorithm involves the time evolution of the distribution mean of the strategy which are recorded in two evolution paths. These paths contain significant information about the correlation between consecutive steps. One path used covariance matrix adaptation procedure in place of single successful search steps and facilitates a possibly faster variance increase of favourable directions. The other path is used to conduct an additional step-size control, which effectively prevents premature convergence to an optimum.

With this understanding, we employed both methods with programming and made comparisons with the same data sample of size 100,000. We run 12 separate normal copula simulations and discovered that their log-likelihood values are similar and that there is no significant outperformance between CMA-ES and BOBYQA optimization methods. Although the sample of simulations might be too small to draw a quick conclusion and due to our limited computational capacity which meant we were unable to conduct more simulations, our best intuition tells that they are both equally sophisticated methods and that we will choose either one for our models. The following shows the results for one of our simulations.

```
Computing MLE Parameters For Normal Copula (BOBYQA)

<Correlation of 0.1> Log-likelihood: -83255.429330
<Correlation of 0.2> Log-likelihood: -83841.219531
<Correlation of 0.1> Log-likelihood: -83255.429330
<Correlation of 0.0> Log-likelihood: -83718.229495
<Correlation of 0.1> Log-likelihood: -83255.429330
<Correlation of 0.094135> Log-likelihood: -83254.856205
<Correlation of 0.104135> Log-likelihood: -83257.965544
<Correlation of 0.093550> Log-likelihood: -83254.992856
<Correlation of 0.096042> Log-likelihood: -83254.654742
<Correlation of 0.096022> Log-likelihood: -83254.654916
<Correlation of 0.096032> Log-likelihood: -83254.654825
<Correlation of 0.096031> Log-likelihood: -83254.654836
<Correlation of 0.093555> Log-likelihood: -83254.991594
<Correlation of 0.096156> Log-likelihood: -83254.654516
<Correlation of 0.096186> Log-likelihood: -83254.654674
<Correlation of 0.095613> Log-likelihood: -83254.667570
<Correlation of 0.096123> Log-likelihood: -83254.654442
<Correlation of 0.096124> Log-likelihood: -83254.654443
<Correlation of 0.096122> Log-likelihood: -83254.654442
<Correlation of 0.096120> Log-likelihood: -83254.654442
<Correlation of 0.096118> Log-likelihood: -83254.654442
<Correlation of 0.096123> Log-likelihood: -83254.654443
<Correlation of 0.096117> Log-likelihood: -83254.654442
<Correlation of 0.096368> Log-likelihood: -83254.657644
<Correlation of 0.096117> Log-likelihood: -83254.654442
<Correlation of 0.096018> Log-likelihood: -83254.654958
<Correlation of 0.096119> Log-likelihood: -83254.654442
<Correlation of 0.096118> Log-likelihood: -83254.654442
<Correlation of 0.096118> Log-likelihood: -83254.654442
<Correlation of 0.096119> Log-likelihood: -83254.654442
<Correlation of 0.096118> Log-likelihood: -83254.654442
<Correlation of 0.096118> Log-likelihood: -83254.654442
<Correlation of 0.096140> Log-likelihood: -83254.654466
<Correlation of 0.096108> Log-likelihood: -83254.654447
<Correlation of 0.096118> Log-likelihood: -83254.654442
Estimated Correlation: 0.096118

Computing MLE Parameters For Normal Copula (CMA-ES)

<Correlation of 0.1> Log-likelihood: -83255.429330
<Correlation of -0.200483> Log-likelihood: -87857.870912
<Correlation of 0.605231> Log-likelihood: -109511.286066
<Correlation of 0.118860> Log-likelihood: -83281.467569
<Correlation of -0.130749> Log-likelihood: -85871.561452
<Correlation of 0.024407> Log-likelihood: -83513.533723
<Correlation of -0.084615> Log-likelihood: -84897.954971
<Correlation of -0.007485> Log-likelihood: -83792.901751
<Correlation of -0.304694> Log-likelihood: -92299.553655
<Correlation of -0.048573> Log-likelihood: -84303.941056
<Correlation of 0.127326> Log-likelihood: -83305.346869
<Correlation of 0.095260> Log-likelihood: -83254.692287
<Correlation of -0.006756> Log-likelihood: -83785.382245
<Correlation of 0.227236> Log-likelihood: -84208.573091
<Correlation of -0.003501> Log-likelihood: -83752.457375
<Correlation of 0.078090> Log-likelihood: -83271.236817
<Correlation of 0.103316> Log-likelihood: -83257.322582
<Correlation of 0.232899> Log-likelihood: -84297.535051
<Correlation of 0.447484> Log-likelihood: -92197.714888
<Correlation of 0.045475> Log-likelihood: -83384.311804
<Correlation of 0.258234> Log-likelihood: -84752.070530
<Correlation of 0.066570> Log-likelihood: -83299.040779
<Correlation of -0.011888> Log-likelihood: -83839.446147
<Correlation of 0.352002> Log-likelihood: -87381.588094
<Correlation of 0.272175> Log-likelihood: -85043.758221
<Correlation of 0.238104> Log-likelihood: -84383.272614
```

```
<Correlation of 0.000927> Log-likelihood: -83709.369811
<Correlation of 0.117990> Log-likelihood: -83279.445076
<Correlation of 0.057843> Log-likelihood: -83328.946769
<Correlation of 0.158186> Log-likelihood: -83458.484080
<Correlation of 0.153343> Log-likelihood: -83427.437152
<Correlation of 0.334307> Log-likelihood: -86754.209823
<Correlation of 0.032884> Log-likelihood: -83456.253541
<Correlation of -0.058729> Log-likelihood: -84457.212886
<Correlation of 0.113325> Log-likelihood: -83269.966836
<Correlation of 0.327769> Log-likelihood: -86539.435073
<Correlation of -0.010901> Log-likelihood: -83828.837100
<Correlation of 0.084119> Log-likelihood: -83262.016284
<Correlation of -0.012339> Log-likelihood: -83844.318987
<Correlation of 0.191961> Log-likelihood: -83751.150710
<Correlation of 0.100021> Log-likelihood: -83255.437884
<Correlation of 0.103478> Log-likelihood: -83257.443735
<Correlation of 0.013963> Log-likelihood: -83593.884697
<Correlation of 0.057399> Log-likelihood: -83330.671793
<Correlation of 0.133382> Log-likelihood: -83327.149017
<Correlation of 0.102577> Log-likelihood: -83256.802086
<Correlation of 0.152586> Log-likelihood: -83422.825039
<Correlation of 0.147353> Log-likelihood: -83392.696985
<Correlation of 0.063595> Log-likelihood: -83308.381036
<Correlation of 0.163915> Log-likelihood: -83498.664437
<Correlation of 0.091644> Log-likelihood: -83255.680779
<Correlation of 0.123122> Log-likelihood: -83292.533204
<Correlation of 0.160763> Log-likelihood: -83476.088792
<Correlation of 0.002168> Log-likelihood: -83697.646315
<Correlation of 0.073110> Log-likelihood: -83281.621770
<Correlation of 0.102176> Log-likelihood: -83256.543460
<Correlation of 0.082433> Log-likelihood: -83264.224197
<Correlation of 0.069538> Log-likelihood: -83290.602380
<Correlation of 0.079230> Log-likelihood: -83269.211425
<Correlation of 0.060461> Log-likelihood: -83319.179322
<Correlation of 0.083375> Log-likelihood: -83262.954840
<Correlation of 0.081692> Log-likelihood: -83265.285777
<Correlation of 0.072736> Log-likelihood: -83282.500976
<Correlation of 0.079379> Log-likelihood: -83268.956954
<Correlation of 0.112963> Log-likelihood: -83269.325874
<Correlation of 0.098000> Log-likelihood: -83254.836398
<Correlation of 0.087248> Log-likelihood: -83258.681900
<Correlation of 0.090034> Log-likelihood: -83256.551312
<Correlation of 0.090341> Log-likelihood: -83256.364539
<Correlation of 0.095070> Log-likelihood: -83254.710826
<Correlation of 0.094413> Log-likelihood: -83254.803682
<Correlation of 0.097612> Log-likelihood: -83254.769139
<Correlation of 0.129782> Log-likelihood: -83313.712868
<Correlation of 0.075587> Log-likelihood: -83276.144402
<Correlation of 0.099659> Log-likelihood: -83255.299110
<Correlation of 0.110930> Log-likelihood: -83265.989239
<Correlation of 0.082187> Log-likelihood: -83264.570885
<Correlation of 0.085037> Log-likelihood: -83260.934954
<Correlation of 0.088408> Log-likelihood: -83257.698234
<Correlation of 0.100447> Log-likelihood: -83255.618307
<Correlation of 0.073762> Log-likelihood: -83280.120133
<Correlation of 0.073555> Log-likelihood: -83280.590536
<Correlation of 0.100884> Log-likelihood: -83255.822946
<Correlation of 0.098716> Log-likelihood: -83255.001406
<Correlation of 0.100191> Log-likelihood: -83255.507455
<Correlation of 0.093090> Log-likelihood: -83255.124829
<Correlation of 0.098174> Log-likelihood: -83254.871546
<Correlation of 0.089090> Log-likelihood: -83257.184177
<Correlation of 0.107413> Log-likelihood: -83261.235206
<Correlation of 0.108052> Log-likelihood: -83262.003386
<Correlation of 0.089719> Log-likelihood: -83256.752232
<Correlation of 0.088775> Log-likelihood: -83257.415843
<Correlation of 0.106034> Log-likelihood: -83259.723229
```

```
<Correlation of 0.096274> Log-likelihood: -83254.655687
<Correlation of 0.105894> Log-likelihood: -83259.581348
<Correlation of 0.098023> Log-likelihood: -83254.840930
<Correlation of 0.101346> Log-likelihood: -83256.060482
<Correlation of 0.115739> Log-likelihood: -83274.584569
<Correlation of 0.094427> Log-likelihood: -83254.801232
<Correlation of 0.095168> Log-likelihood: -83254.700841
Estimated Correlation: 0.096274
```

5.2 Analysis with Real Financial Data

At the second stage of our project, we back-tested our model with actual financial data using normal copula. For simplification, we use Hang Seng Index and Thomson Reuters Counterparty Default Index (CDI for subordinated debt) as proxies to represent market risk and credit risk respectively. The former is chosen as a capitalization-weighted equity index, which tracks the overall market performance in Hong Kong. The latter is chosen as the weighted-average of five-year credit default swap spreads for top dealers in the global market.

The sample data collected from Thomson Reuters database consisted of daily closing prices during the five-year period from April 2009 to April 2014. There are a total of 1256 paired observations of X and Y (which are both assumed to follow normal distributions). The following shows the summary statistics.

```
Reading Excel Data...

<0> X: -0.004169, Y: -0.024469, X+Y: -0.028638
<1> X: 0.071527, Y: 0.022261, X+Y: 0.093788
<2> X: 0.001632, Y: 0.021738, X+Y: 0.02337
<3> X: 0.030625, Y: 0.020519, X+Y: 0.051144
<4> X: -0.004616, Y: 0.008438, X+Y: 0.003822
<5> X: -0.03089, Y: 0.001312, X+Y: -0.029578
<6> X: 0.029042, Y: 0.0, X+Y: 0.029042
<7> X: 0.044542, Y: 0.070645, X+Y: 0.115187
...
<1249> X: -0.0013, Y: -0.002175, X+Y: -0.003475
<1250> X: -0.009772, Y: 0.026328, X+Y: 0.016556
<1251> X: 0.002359, Y: -0.009068, X+Y: -0.006709
<1252> X: -0.015151, Y: -0.055017, X+Y: -0.070168
<1253> X: -0.004103, Y: 2.62E-4, X+Y: -0.003841
<1254> X: 0.014415, Y: 0.019995, X+Y: 0.03441
<1255> X: -0.01435, Y: 0.012804, X+Y: -0.001546

Descriptive Stats For Bivariate Data (Sample Size: 1256)
Mean of Sample X: 0.000389, SD of Sample X: 0.013244
Mean of Sample Y: 0.001053, SD of Sample Y: 0.028245
Alpha of Sample Y: 0.000335, Beta of Sample Y: 0.317612
```

Based on the normal copula, running maximum likelihood estimation using BOBYQA optimization method with initial correlation of 0.3, yields the log-likelihood results. The 99% VaR and expected shortfall with their 95% confidence intervals for the sample are also computed. The 99.9% VaR and expected shortfall are unavailable due to the limited sample size.

```
Computing MLE Parameters for Normal Copula (BOBYQA)

<Correlation of 0.3> Log-likelihood: -963.759669
<Correlation of 0.4> Log-likelihood: -953.096086
<Correlation of 0.3> Log-likelihood: -963.759669
<Correlation of 0.2> Log-likelihood: -987.91622
<Correlation of 0.3> Log-likelihood: -963.759669
<Correlation of 0.429031> Log-likelihood: -953.401679
<Correlation of 0.41> Log-likelihood: -952.996073
<Correlation of 0.409168> Log-likelihood: -952.996458
<Correlation of 0.402929> Log-likelihood: -953.04548
<Correlation of 0.408701> Log-likelihood: -952.997316
<Correlation of 0.400041> Log-likelihood: -953.095257
<Correlation of 0.411029> Log-likelihood: -952.99763
<Correlation of 0.410365> Log-likelihood: -952.996367
<Correlation of 0.411903> Log-likelihood: -953.000724
<Correlation of 0.410044> Log-likelihood: -952.996094
<Correlation of 0.409004> Log-likelihood: -952.996707
<Correlation of 0.409843> Log-likelihood: -952.996034
<Correlation of 0.409851> Log-likelihood: -952.996035
<Correlation of 0.409993> Log-likelihood: -952.996071
<Correlation of 0.409803> Log-likelihood: -952.996032
<Correlation of 0.409859> Log-likelihood: -952.996036
<Correlation of 0.409803> Log-likelihood: -952.996032
<Correlation of 0.409804> Log-likelihood: -952.996032
<Correlation of 0.409803> Log-likelihood: -952.996032
<Correlation of 0.409779> Log-likelihood: -952.996033
<Correlation of 0.409803> Log-likelihood: -952.996032
<Correlation of 0.409803> Log-likelihood: -952.996032
<Correlation of 0.409803> Log-likelihood: -952.996032
Estimated Correlation: 0.409803

VaR under Normal Copula (Sample Size: 1256)
VaR at 99.0%: -0.094915
Lower Bound of 99.0% VaR: -0.121489
Upper Bound of 99.0% VaR: -0.081616

Expected Shortfall under Normal Copula (Sample Size: 1256)
Expected Shortfall at 99.0%: -0.119699
Lower Bound of 99.0% ES: -0.130501
Upper Bound of 99.0% ES: -0.108897
```

To test for the effect of heavy tail in the sample, we attempt to run maximum likelihood estimation on t-copula this time, using the same BOBYQA optimization method with initial correlation of 0.3 and degrees of freedom of 1.5. The following yields results with the highest log-likelihood.

```
Computing MLE Parameters for T Copula (BOBYQA)

<Correlation of 0.3, df of 1.5> Log-likelihood: -2939.254325
<Correlation of 0.4, df of 1.5> Log-likelihood: -2947.103398
<Correlation of 0.3, df of 1.6> Log-likelihood: -2805.022606
<Correlation of 0.2, df of 1.5> Log-likelihood: -2942.456338
<Correlation of 0.3, df of 1.4> Log-likelihood: -3093.675552
<Correlation of 0.298129, df of 1.699982> Log-likelihood: -2687.337649
<Correlation of 0.293933, df of 1.899938> Log-likelihood: -2490.924543
<Correlation of 0.248299, df of 2.297327> Log-likelihood: -2209.229762
<Correlation of -0.50177, df of 2.575531> Log-likelihood: -2401.61751
<Correlation of 0.550451, df of 2.559441> Log-likelihood: -2100.948915
<Correlation of 0.841089, df of 2.834267> Log-likelihood: -2389.058183
<Correlation of 0.445229, df of 2.56105> Log-likelihood: -2073.984673
<Correlation of 0.305311, df of 2.703959> Log-likelihood: -2004.92632
<Correlation of 0.026292, df of 2.990574> Log-likelihood: -1945.567978
<Correlation of -0.069483, df of 3.081927> Log-likelihood: -1946.748355
<Correlation of 0.002459, df of 3.087693> Log-likelihood: -1921.539737
<Correlation of 0.040214, df of 3.180292> Log-likelihood: -1883.18617
<Correlation of 0.017203, df of 3.378964> Log-likelihood: -1835.821556
<Correlation of -0.134714, df of 3.748992> Log-likelihood: -1806.630222
<Correlation of -0.126475, df of 3.814424> Log-likelihood: -1790.223457
<Correlation of -0.070633, df of 4.00647> Log-likelihood: -1732.564454
<Correlation of -0.114132, df of 4.404097> Log-likelihood: -1686.991095
<Correlation of -0.134572, df of 4.803575> Log-likelihood: -1644.041895
<Correlation of -0.345661, df of 5.575223> Log-likelihood: -1693.442898
<Correlation of 0.082956, df of 4.834921> Log-likelihood: -1562.922079
<Correlation of 0.056007, df of 5.234012> Log-likelihood: -1526.619808
<Correlation of -0.022002, df of 6.0302> Log-likelihood: -1483.495872
<Correlation of -0.203017, df of 7.534652> Log-likelihood: -1480.839196
<Correlation of 0.239649, df of 7.58458> Log-likelihood: -1319.056438
<Correlation of 0.162527, df of 8.380854> Log-likelihood: -1301.663941
<Correlation of 0.174048, df of 8.163625> Log-likelihood: -1307.412083
<Correlation of -0.054706, df of 8.369416> Log-likelihood: -1374.161459
<Correlation of 0.180187, df of 8.597669> Log-likelihood: -1289.286927
<Correlation of 0.195653, df of 9.032462> Log-likelihood: -1270.650487
<Correlation of 0.224572, df of 9.466567> Log-likelihood: -1250.746234
<Correlation of 0.261857, df of 10.335903> Log-likelihood: -1219.716485
<Correlation of 0.372008, df of 11.199038> Log-likelihood: -1186.174773
<Correlation of 0.41635, df of 12.938743> Log-likelihood: -1152.500898
<Correlation of 0.633265, df of 14.665441> Log-likelihood: -1194.57551
<Correlation of -0.127585, df of 12.942297> Log-likelihood: -1304.305669
<Correlation of 0.147134, df of 12.111302> Log-likelihood: -1208.486492
<Correlation of 0.490427, df of 12.781269> Log-likelihood: -1160.862492
<Correlation of 0.224684, df of 13.329317> Log-likelihood: -1169.379525
<Correlation of 0.562715, df of 13.032885> Log-likelihood: -1176.240566
<Correlation of 0.416754, df of 12.721209> Log-likelihood: -1156.066455
<Correlation of 0.372312, df of 12.954337> Log-likelihood: -1153.506814
<Correlation of 0.406667, df of 12.915694> Log-likelihood: -1152.889401
<Correlation of 0.409594, df of 12.962813> Log-likelihood: -1152.110195
<Correlation of 0.396864, df of 13.011165> Log-likelihood: -1151.532512
<Correlation of 0.36801, df of 13.106912> Log-likelihood: -1151.371069
<Correlation of 0.339923, df of 13.080032> Log-likelihood: -1154.190774
<Correlation of 0.374767, df of 13.156453> Log-likelihood: -1150.177681
<Correlation of 0.356164, df of 13.254707> Log-likelihood: -1149.9819
<Correlation of 0.335509, df of 13.27689> Log-likelihood: -1151.629089
<Correlation of 0.383857, df of 13.296338> Log-likelihood: -1147.563002
<Correlation of 0.394232, df of 13.395798> Log-likelihood: -1145.675134
<Correlation of 0.396656, df of 13.595783> Log-likelihood: -1142.661048
<Correlation of 0.39905, df of 13.995776> Log-likelihood: -1136.968976
<Correlation of 0.424025, df of 14.795386> Log-likelihood: -1126.576971
<Correlation of 0.392781, df of 16.395081> Log-likelihood: -1109.291806
<Correlation of 0.440131, df of 19.594731> Log-likelihood: -1083.270159
<Correlation of 0.523442, df of 23.030333> Log-likelihood: -1076.142608
<Correlation of 0.481659, df of 22.939251> Log-likelihood: -1067.871055
<Correlation of 0.411255, df of 24.656114> Log-likelihood: -1055.297734
<Correlation of 0.417088, df of 28.092722> Log-likelihood: -1042.467092
```

```
<Correlation of 0.356118, df of 34.965676> Log-likelihood: -1027.337278
<Correlation of 0.343629, df of 37.240958> Log-likelihood: -1024.181671
<Correlation of 0.326877, df of 38.256603> Log-likelihood: -1024.45137
<Correlation of 1.359269, df of 37.257982> Log-likelihood: NaN
<Correlation of 0.444884, df of 37.232865> Log-likelihood: -1021.053385
<Correlation of 0.444854, df of 37.222865> Log-likelihood: -1021.069811
<Correlation of 0.443739, df of 37.2428> Log-likelihood: -1020.961676
<Correlation of 0.444724, df of 37.242973> Log-likelihood: -1021.024425
<Correlation of 0.442739, df of 37.242781> Log-likelihood: -1020.899967
<Correlation of 0.442634, df of 37.243776> Log-likelihood: -1020.89185
<Correlation of 0.442587, df of 37.243864> Log-likelihood: -1020.88884
<Correlation of 0.442495, df of 37.243903> Log-likelihood: -1020.883228
<Correlation of 0.44242, df of 37.243969> Log-likelihood: -1020.878731
<Correlation of 0.442427, df of 37.243977> Log-likelihood: -1020.879119
<Correlation of 0.442413, df of 37.243976> Log-likelihood: -1020.878281
<Correlation of 0.442406, df of 37.243969> Log-likelihood: -1020.877893
<Correlation of 0.442406, df of 37.243969> Log-likelihood: -1020.877893
Estimated Correlation: 0.442406, Estimated df: 37.243969
```

As observed, the estimated degrees of freedom, that is 37.243969 suggests the absence of heavy tail for the sample, which is contrary to our expectations of real financial data. We attribute this to two significant factors. The first reason is that there is too small a sample size for the estimation model to accurately detect the subtle tail influences in the actual population. We decided to rerun multiple estimations for simulated data with our typical sample size of 100,000. The results revealed no conclusive improvement either as there are moderate variations in estimated degree of freedoms from its actual ones. We might believe that we will need significantly even larger samples to fully reflect intrinsic tail dependencies.

The second reason is a model risk and also a limitation of our current implementation. Applying a quasi-MLE algorithm, that is the method of inference functions for margins (IFM) results in a loss in accuracy. This less-than-optimal maximization technique in favour of computational efficiency might be a potential source of discrepancies especially in slightly complex optimization when estimating multiple parameters.

6. CONCLUSION

In the bottom-up approach as published by Grundke (2006), the extended CreditMetrics model with correlated interest rate and credit spread risk is applied to measure the credit and market risk of banking book instruments simultaneously. In this context, the new methodology is specifically examined in contrast with our top-down approach.

It is useful to note some of Grundke remarks on the inadequacies of the top-down approach. He identifies a significant underestimation for VaR and expected shortfall when tested for the initial rating of Baa. His findings also reveal the very low excess kurtosis of the loss distributions when compared to the bottom-up approach. As he explains, the integrated view of the bottom up approach can reproduce the stochastic dependencies between credit quality transitions and the distribution of the credit spreads at the risk horizon. The top-down approach however cannot reflect the extreme losses due to simultaneous downgrades and adverse movements of credit spreads. We believe this applies for our new copula tool as we can seek to provide further insights on this result.

The first stage of future work will be to complete the bottom-up implementation to replicate appropriate scenarios for a quantitative comparison of different credit qualities. Moreover, our preliminary analysis work on real financial data has created new ground for extensive research needed for larger samples and a realistic estimation of heavy tail in the sample. The second stage of future work will be to investigate these effects of model risk and its compatibility with the firm-specific banking book loss returns. This is especially critical since banks often manage their market risks actively for example, through derivatives to restrict market losses during financial distress. We might need to include use of dynamic models to reflect the evolution of risk factors and portfolio compositions over time. Lastly, our research also shows the possibilities for an even wider integration of credit risk and market risk with operational risk, which combines both top-down and bottom-up approaches. The aim is to give a holistic representation of the interactions of multiple risk factors in typical market situations.

In closing, it is very exciting to look forward to great future developments. Taking a step back in retrospect, the project has been fruitful and rewarding so far as we accomplished all four significant stages of our project. We have successfully implemented our top-down approach, back-tested with actual financial data, developed the copula tool applet and compared between approaches. Though we wished we had more time to do even more with further analysis, we would have thought we had come a long way in learning and building sophisticated copula models from scratch and testing results. We sincerely believe that our copula tool has great potential to follow-up in future research on risk integration.

7. REFERENCES

Aas, K., Dimakos, X.K., Øksendal, A., 2007, "Risk capital aggregation. Risk Management 9, 82–107."

Alessandri, P., Drehmann, M., 2007, "An economic capital model integrating credit and interest rate risk. Working Paper, Bank of England and European Central Bank."

Barnhill Jr., T.M., Maxwell, W.F., 2002, "Modeling correlated market and credit risk in fixed income portfolios. Journal of Banking and Finance 26, 347–374."

Basel Committee on Banking Supervision, 2003, "Trends in risk integration and aggregation", Trends in Market Practices, 3-4.

Böcker, K., Hillebrand, M., 2008, "Interaction of market and credit risk: An analysis of inter-risk correlation and risk aggregation."

Bouyé, Eric, 2001, "Multivariate Extremes at Work for Portfolio Risk Measurement".

Dimakos, X.K., Aas, K., 2004, "Integrated risk modelling. Statistical Modelling 4 (4), 265–277."

Drehmann, M., Sorensen, S., Stringa, M., 2007, "The integrated impact of credit and interest rate risk on banks: An economic value and capital adequacy perspective."

Embrechts, Paul, Alexander McNeil and Daniel Straumann, 1999, "Correlation: Pitfalls and alternatives," Risk (May), 69-71.

Frey, Rüdiger and Alexander J. McNeil, 2001, "Modeling Dependent Defaults,".

Gabriel Jim´enez, Javier Menc´ıa, 2007, "Modelling the distribution of credit losses with observable and latent factors".

Glasserman, Paul, Philip Heidelberger and Perwez Shahabuddin, 2002, "Portfolio Value-at-Risk with Heavy-Tailed Risk Factors," Mathematical Finance 12 (July), 239-270.

Jimmy Skoglund, 2010, "Risk Aggregation and Economic Capital".

Jose A. Lopez, Marc R. Saidenberg, 2000, "Evaluating credit risk models".

Joshua V. Rosenberg and Til Schuermann 2004, "A General Approach to Integrated Risk Management with Skewed, Fat-Tailed Risks".

Kuritzkes, A., Schuermann, T., Weiner, S.M., 2002, "Risk Measurement, RiskManagement and Capital Adequacy in Financial Conglomerates. Center for Financial Institutions Working Papers 03-02."

Li, D. X., 2000, "On Default Correlation: A Copula Function Approach," Journal of Fixed Income, March, 43 - 54.

M.J.D. Powell, 2009, "The BOBYQA algorithm for bound constrained optimization without derivatives".

Paul H. Kupiec, 2007, "An integrated Structural Model for Portfolio Market and Credit Risk".

Peter Grundke, 2006, "Integrated Risk Management: Top Down or Bottom Up?".

Peter Grundke, 2009, “Top-down approaches for integrated risk management: How accurate are they? Section 4.3.”

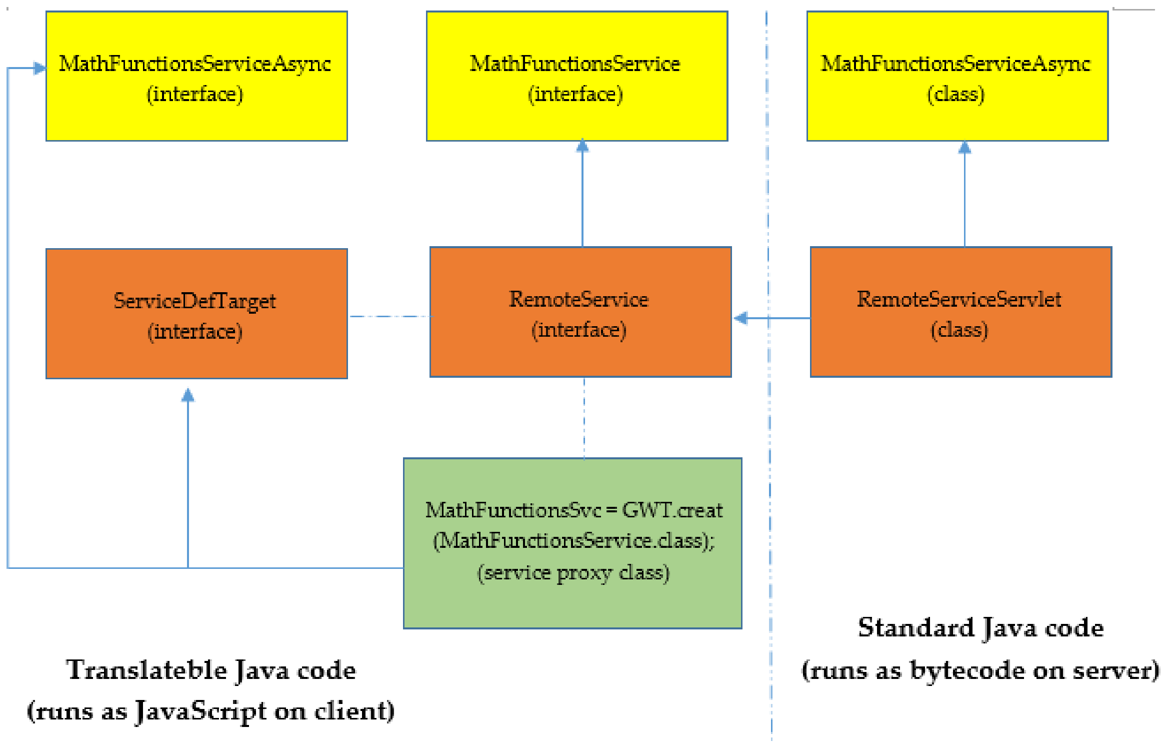
Rosenberg, J.V., Schuermann, T., 2006, “A general approach to integrated risk management with skewed, fat-tailed risks. Journal of Financial Economics 79(3), 569–614.”

Rosenberg & Schuermann, 2004, “A General Approach to Integrated Risk Management with Skewed, Fat-Tailed Risks”.

Ward, L.S., Lee, D.H., 2002. “Practical application of the risk-adjusted return on capital framework. Working Paper.”

8. Appendix A: Remote Protocol Call Framework in GWT

RPC specifically handles the serialization and exchange of Java objects over HTTP in the arguments in the method calls and return values through invoking several services. The following shows the structure of our RPC implementation.


MathFunctionsServiceAsync (interface)
MathFunctionsService (interface)
MathFunctionsServiceAsync (class)
ServiceDefTarget (interface)
RemoteService (interface)
RemoteServiceServlet (class)
MathFunctionsSvc = GWT.creat (MathFunctionsService.class); (service proxy class)
Translateble Java code (runs as JavaScript on client)
Standard Java code (runs as bytecode on server)

In order to define the RPC interface, we have three main components:

1. An interface (MathFunctionsService) for the service that extends RemoteService and lists all RPC methods

2. A class (MathFunctionsServiceImpl) that extends RemoteServiceServlet and implements the interface created above.

3. An asynchronous interface (MathFunctionsServiceAsync) to the service to be called from the client-side code.