



Using Longstaff & Schwartz Method for Pricing American Basket Options By Monte Carlo Simulation

Mathematical Modeling and Simulation
(MC-409)

Dr. Vivek Kumar Aggarwal

10.10.2020

Anish Sachdeva

DTU/2K16/MC/13

Delhi Technological University

Overview

Introduction.....	2-14
Option Pricing.....	2-3
Geometric Brownian Motion (GBM).....	4-7
Monte Carlo Methods.....	8
The LSM (Least Squares Method) Algorithm.....	9-14
The Problem.....	15
Overview of The Modelling Framework.....	16-17
Code and Model.....	18-41
Importing the Supporting Historical Dataset.....	18-19
Extreme Value Theory and Probability Distribution.....	20-24
Copula Calibration.....	25-26
Copula Simulation.....	27
Simulation Using Gaussian Copula.....	28-29
Simulation Using Traditional Brownian Motion.....	30
Longstaff & Schwartz Approach.....	31-35
Monte Carlo Simulations.....	36
American Option Pricing Using the Longstaff & Schwartz Approach.....	37-41
Bibliography.....	42

Introduction

Option Pricing

Options are one of the most common financial derivatives used on financial markets. They are contracts giving the buyer of the option the right, but not the obligation, to buy or sell an underlying asset to a specified strike price. Options can be of two types, put or call. A put option gives the owner the right to sell the underlying asset at the agreed upon strike price, and a call option the right to buy the asset for the strike price. There are several different kinds of options, where the most common are called European and American options. The difference is that a European option only gives the owner the right to exercise at a specific time in the future, whereas an American option can be exercised at any time up until the expiration time.

Arguably one of the most important achievements in the field of financial mathematics was the development of the Black-Scholes model. It is a mathematical model describing the value of an option over time, and it makes it possible to explicitly find the value of a European option, under certain conditions.

The value of the option over time can be described as a partial differential equation, called the Black-Scholes equation, which is

$$\frac{\partial u}{\partial t} + rS \frac{\partial u}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - ru = 0$$

Where:

1. $S = S(t)$ is the price of the underlying asset
2. $u = u(S, t)$ is the value of the option
3. r is the risk free interest rate
4. t is the time, where
 - i. $-t = 0$ denotes the present time and
 - ii. $-t = T$ denotes the expiration time
5. σ is the volatility of the underlying asset
6. K is the strike Price

Given the Boundary Condition

$$u(S, T) = \max(K - S, 0)$$

Which corresponds to a special case of a European put option, where $\max(K - S, 0)$ is the payoff of a put option, the Black-Scholes equation has the Solution:

$$u(S, t) = Ke^{-r(T-t)}\Phi(-d_2) - S\Phi(-d_1)$$

Where,

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t}$$

And Φ is the cumulative distribution function of the standard normal distribution. When it comes to American Options the Black-Scholes equation becomes:

$$\frac{\partial u}{\partial t} + rS\frac{\partial u}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - ru \leq 0$$

With the boundary conditions

$$u(S, T) = \max(K - S, 0) \text{ and } u(S, t) \geq \max(K - S, 0)$$

for an American put. This equation does not have an analytic solution, since the option can be exercised at any time up until expiration. The problem then becomes to find the optimal time to exercise, that is the time when the payoff of immediate exercise is greater than the expected reward of keeping the option.

Since this problem doesn't have an analytical solution a number of numerical methods have been developed, such as finite-difference methods, binomial tree models, Monte Carlo methods, and many more.

We will focus on the least squares Monte Carlo (LSM) method developed by Longstaff and Schwartz.

Geometric Brownian Motion

One of the essential assumptions of the Black-Scholes model is that the underlying asset, most commonly a stock, is modelled as a geometric Brownian motion, and the LSM method is based on the same framework.

First we need to define a standard Brownian motion, or Wiener process.

Definition 1 A random process $\{W(t)\}$ is said to be a standard Brownian motion on $[0, T]$ if it satisfies:

- I. $W(0) = 0$
- II. $\{W(t)\}$ has independent and stationary increments
- III. $W(t) - W(s) \sim N(0, t - s)$ for any $0 \leq s \leq t \leq T$
- IV. $\{W(t)\}$ has almost surely continuous Trajectories

From this definition it follows that:

$$W(t) \sim N(0, t) \text{ for } 0 < t \leq T$$

Which will be important when simulating the Brownian Motion. Next, we need to define a Brownian Motion with drift and Diffusion Coefficient.

Definition 2 A process $\{X(t)\}$ is said to be a Brownian motion with drift $\mu > 0$ and diffusion coefficient $\sigma^2 > 0$ if

$$\frac{X(t) - \mu t}{\sigma}$$

is a standard Brownian motion.

This means that

$$X(t) \sim \mathcal{N}(\mu t, \sigma^2 t) \text{ for } 0 < t \leq T$$

and that we can construct the process $\{X(t)\}$ using a standard Brownian motion $\{W(t)\}$ by putting

$$X(t) = \mu t + \sigma W(t)$$

The process $\{X(t)\}$ also solves the stochastic differential equation (SDE)

$$dX(t) = \mu t + \sigma dW(t)$$

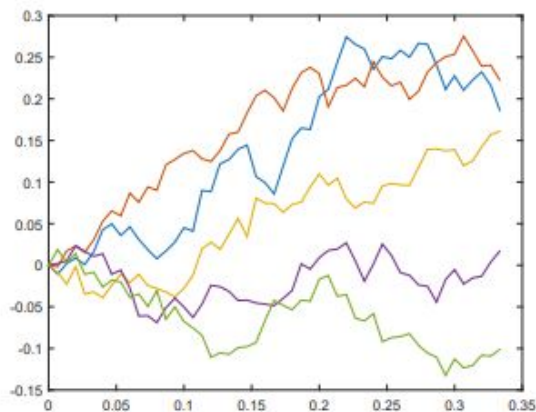
which will be used when we shall construct the geometric Brownian motion.

A Brownian motion is a continuous stochastic process, and as such it cannot be modelled exactly but has to be discretized for a finite number of time steps $t_0 < t_1 < \dots < t_N$. Let $t_0 = 0$, $X(0) = 0$, and Z_1, \dots, Z_N be a set of i.i.d. $N(0, 1)$ random variables. Then the process $\{X(t)\}$ can be simulated as

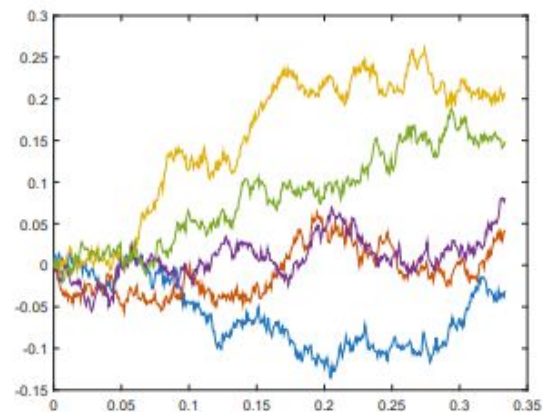
$$X(t_{i+1}) = X(t_i) + \mu(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i}Z_{i+1}$$

for $i = 0, \dots, N - 1$.

Figure 1.1 shows five simulated paths for a Brownian motion with drift $\mu = 0.0488$ and diffusion coefficient $\sigma^2 = 0.04$, in 1.1a with $N = 50$ and in 1.1b with $N = 500$.



(a) 50 time steps



(b) 500 time steps

Figure 1.1: Simulated Brownian motions

A regular Brownian motion cannot be used as a model for an asset price, since it can assume negative values. For this we need the geometric Brownian motion (GBM), which is essentially an exponentiated Brownian motion.

Definition 3 A process $\{S(t)\}$ is said to be a geometric Brownian motion if it solves the stochastic differential equation

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t)$$

To solve this SDE we need to use some Itô calculus, which is a generalised calculus for stochastic processes. More specifically we will use Itô's lemma, which is a generalisation of the chain rule.

Theorem 1 (Itô's lemma). Let $X(t)$ be a Brownian motion with drift μ and diffusion coefficient σ^2 satisfying the SDE

$$dX(t) = \mu dt + \sigma dW(t)$$

and let $f(X, t)$ be a twice differentiable function. Then

$$df(X, t) = \left(\frac{\partial f}{\partial t} + \mu \frac{\partial f}{\partial X} + \frac{1}{2} \sigma^2 \frac{\partial^2 f}{\partial X^2} \right) dt + \sigma \frac{\partial f}{\partial X} dW(t)$$

Applying Itô's lemma to the GBM $\{S(t)\}$ we get

$$df(S, t) = \left(\frac{\partial f}{\partial t} + \mu S \frac{\partial f}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} \right) dt + \sigma S \frac{\partial f}{\partial S} dW(t) \quad (1.3)$$

and if we let $f(S, t) = f(S) = \ln(S)$ we get

$$\frac{\partial f}{\partial t} = 0, \quad \frac{\partial f}{\partial S} = \frac{1}{S}, \quad \frac{\partial^2 f}{\partial S^2} = -\frac{1}{S^2}$$

which means that equation (1.3) becomes

$$\begin{aligned} d \ln(S) &= \left(\mu S \cdot \frac{1}{S} + \frac{1}{2} \sigma^2 S^2 \cdot \left(-\frac{1}{S^2} \right) \right) dt + \sigma S \cdot \frac{1}{S} dW(t) \\ &= \left(\mu - \frac{1}{2} \sigma^2 \right) dt + \sigma dW(t) \end{aligned}$$

Integrating both sides of this equation gives

$$\begin{aligned} \int_{t_0}^{t_1} d \ln(S) &= \int_{t_0}^{t_1} \left(\mu - \frac{1}{2} \sigma^2 \right) dt + \int_{t_0}^{t_1} \sigma dW(t) \\ &\iff \\ \ln(S(t_1)) - \ln(S(t_0)) &= \left(\mu - \frac{1}{2} \sigma^2 \right) (t_1 - t_0) + \sigma (W(t_1) - W(t_0)) \\ &\iff \end{aligned}$$

$$\ln(S(t_1)) = \left(\mu - \frac{1}{2} \sigma^2 \right) (t_1 - t_0) + \sigma (W(t_1) - W(t_0)) + \ln(S(t_0))$$

Exponentiating both sides of this yields the solution

$$S(t_1) = S(t_0) e^{(\mu - \frac{1}{2} \sigma^2)(t_1 - t_0) + \sigma(W(t_1) - W(t_0))} \quad (1.4)$$

Putting $t_0 = 0$ and $t_1 = t$ gives the more common expression

$$S(t) = S(0) e^{(\mu - \frac{1}{2} \sigma^2)t + \sigma W(t)}$$

Since $W(t)$ is a standard Brownian motion, equation (1.4) gives a straight forward way to simulate the GBM $S(t)$ analogue to that of a regular Brownian motion in equation (1.2):

$$S(t_{i+1}) = S(t_i)e^{(\mu - \frac{1}{2}\sigma^2)(t_{i+1}-t_i) + \sigma\sqrt{t_{i+1}-t_i}Z_{i+1}}$$

Monte Carlo Methods

As the name least squares Monte Carlo suggests, the LSM algorithm uses Monte Carlo (MC) to estimate the value of the option.

The general idea of MC methods is to simulate a sample of something you are interested in, and taking the mean to find the "true" value. Mathematically MC is often described as a method to estimate the value of an integral:

$$\mu = \int f(x) dx$$

Factorizing the function $f(x)$ as

$$f(x) = h(x) \cdot p(x)$$

Where $p(x)$ is some density, the integral can be interpreted as the Expected Value

$$\mu = E[h(x)] = \int h(x)p(x) dx$$

By generating an i.i.d sample x_1, x_2, \dots, x_n from $p(x)$, the expected value can be estimated as

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n h(x_i)$$

Given that $h(x)$ is integrable, the law of large numbers states that

$$\hat{\mu} \rightarrow \mu \text{ almost surely as } n \rightarrow \infty$$

And if $h(x)$ is also square integrable and we define

$$\sigma^2 = \text{Var} [h(x)] = \int (h(x) - \mu)^2 p(x) dx$$

It also says that

$$\frac{\sqrt{n} (\hat{\mu} - \mu)}{\sigma} \rightarrow \mathcal{N}(0, 1)$$

This means that the error $\hat{\mu} - \mu$ is normally distributed with the mean 0 and standard deviation σ / \sqrt{n} which gives the convergence rate $O(1 / \sqrt{n})$ for MC Methods. This is both a strength and weakness, as it is comparatively slow convergence but it does not increase with the dimension of the problem as it does for most other numerical methods. This means that MC methods are not very competitive in one dimension, but as dimensions increase so does their competitiveness.

The LSM Algorithm

As mentioned in the above section, the problem with pricing American options is that they can be exercised at all times up until the expiration of the option, unlike a European option that can only be exercised at the expiration time. Here we will use the same notation as in above section, letting

$$\begin{aligned} g(S(t)) &= \max(K - S(t), 0) && \text{for a put option, and} \\ g(S(t)) &= \max(S(t) - K, 0) && \text{for a call option} \end{aligned}$$

be the payoff at time t , and assume that the underlying asset is modelled using the GBM

$$S(t) = S(0)e^{(r - \frac{1}{2}\sigma^2)t + \sigma W(t)}$$

For ease of notation we will only consider put options from now on, but all results are of course applicable to call options as well.

Using the assumptions above, the value at time 0 of a European option can be described as

$$u(S, 0) = E[e^{-rT} g(S(T))]$$

That is, the expected value of the discounted payoff at time T . In a similar way the value of an American option at time 0 is given by

$$u(S, 0) = \sup_{t \in [0, T]} \mathbb{E} \left[e^{-rt} g(S(t)) \right],$$

the expected value of the discounted payoff at the time of exercise that yields the greatest payoff. The reason for this is the assumption of no arbitrage, that is the chance for risk-free reward, which means the option must cost as much as the maximum expected reward from it. This corresponds to the optimization problem of finding the optimal stopping time

$$t^* = \inf \{t \geq 0 \mid S(t) \leq b^*(t)\}$$

for some unknown exercise boundary b^* , as illustrated in Figure 2.1.

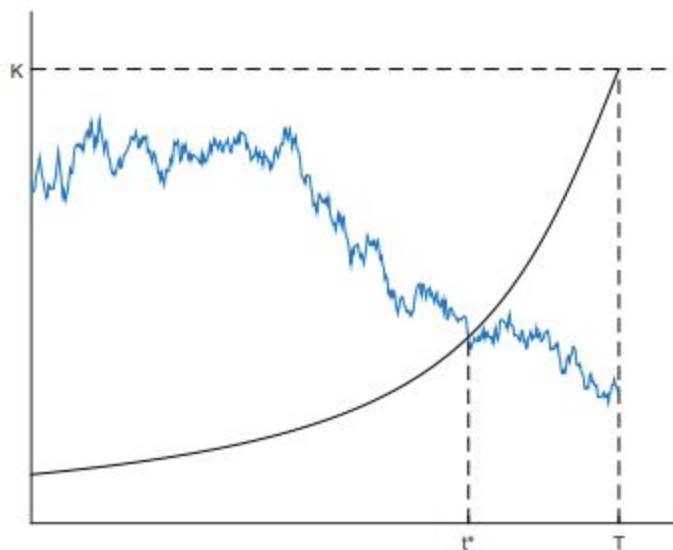


Figure 2.1: Exercise boundary for an American put

So in order to price an American option we need to find the optimal stopping time t^* , and then estimate the expected value

$$u(S, 0) = \mathbb{E} \left[e^{-rt^*} g(S(t^*)) \right]$$

The LSM method, developed by Longstaff and Schwartz in [1], uses a dynamic programming approach to find the optimal stopping time, and Monte Carlo to approximate the expected value. Dynamic programming is a general method for solving optimization problems by dividing it into smaller subproblems and combining their solution to solve the problem. In this case this means that we divide the interval $[0, T]$ into a finite set of time points $\{0, t_1, t_2, \dots, t_N\}$, $t_N = T$, and for each of these decide if it is better to exercise than to hold on to the option. Starting from time T and working backwards to time 0 , we update the stopping time each time we find a time where it is better to exercise until we have found the smallest time where exercise is better.

Let $C(S(t_i))$ denote the value of holding on to the option at time t_i , from now on called the continuation value, and let the value of exercise at time t_i to be the payoff $g(S(t_i))$. Then the dynamic programming algorithm to find the optimal stopping time is given by Algorithm 1.

Algorithm 1 Dynamic programming to find optimal stopping time

```

 $t^* \leftarrow t_N$ 
for  $t$  from  $t_{N-1}$  to  $t_1$  do
  if  $C(S(t)) < g(S(t))$  then
     $t^* \leftarrow t$ 
  else
     $t^* \leftarrow t^*$ 
  end if
end for

```

Using the same arguments as in Equation 2.1, the continuation value at time t_i can be described as the conditional expectation

$$C(S(t_i)) = E \left[e^{-r(t^*-t_i)} g(S(t^*)) \mid S(t_i) \right]$$

where t^* is the optimal stopping time in $\{t_{i+1}, \dots, t_N\}$. Since we are using the method described above in Algorithm 1, such a t^* will always exist. For ease of notation, we define the current payoff $P = P(S(t))$ as:

- For $t = t_N$
 - Let $P = g(S(t))$
- From $t = t_{N-1}$ to $t = t_1$
 - If $C(S(t)) < g(S(t))$, let $P = g(S(t))$

- Otherwise let $P = e^{-r\Delta t}P$

Here $t_{i+1} - t_i$ is denoted by Δt , as we assume that we have equidistant time points. Given this notation, Equation 2.3 becomes

$$C(S(t_i)) = E[e^{-r\Delta t}P \mid S(t_i)] \quad (2.4)$$

To estimate this conditional expectation, the LSM method uses regular least squares regression. This can be done since the conditional expectation is an element in L^2 space, which has an infinite countable orthonormal basis and thus all elements can be represented as a linear combination of a suitable set of basis functions. So to estimate this we need to choose a (finite) set of orthogonal basis functions, and project the discounted payoffs onto the space spanned by these.

In my implementation the basis functions is chosen to be the Laguerre polynomials¹, where the first four are defined as:

$$\begin{aligned} L_0(X) &= 1 & L_2(X) &= \frac{1}{2}(2 - 4X + X^2) \\ L_1(X) &= 1 - X & L_3(X) &= \frac{1}{6}(6 - 18X + 9X^2 - X^3) \end{aligned}$$

Given a set of realised paths $S_i(t)$, $i = 1, \dots, n$ that are in-the-money at time t , i.e. $g(S_i(t)) > 0$, and the payoffs $P_i = P(S_i(t))$, the conditional expectation in Equation 2.4 can be estimated as

$$\hat{C}(S_i(t)) = \sum_{j=0}^k \hat{\beta}_j L_j(S_i(t)) \quad (2.5)$$

where L_0, \dots, L_K are the k first Laguerre polynomials and β_0, \dots, β_k are the estimated regression coefficients. The regression coefficients are obtained by regressing the discounted payoffs $y_i = e^{-r\Delta t}P_i$ against the current values $x_i = S_i(t)$ by regular least squares:

$$(\hat{\beta}_0, \dots, \hat{\beta}_k)^T = (\mathbf{L}^T \mathbf{L})^{-1} \mathbf{L}^T (y_1, \dots, y_n)^T$$

where $L_{ij} = L_j(x_i)$, $i = 1, \dots, n$ and $j = 0, \dots, k$.

By approximating (2.4) with (2.5), we introduce an error in our estimate. In [3] it is shown that

$$\lim_{k \rightarrow \infty} \hat{C}(S(t)) = C(S(t))$$

but not at which rate. In section 3.2, the convergence with respect to the number of basis functions is investigated further.

Now that we have a way to estimate the continuation value, we can simulate a set of M realised paths $S_i(t)$, $t = 0, t_1, t_2, \dots, t_N$, $i = 1, 2, \dots, M$ and use the method from Algorithm 1 to find optimal stopping times t_i^* for all paths, and then estimate the expected value in Equation 2.2 using Monte Carlo:

$$\hat{u} = \frac{1}{M} \sum_{i=1}^M e^{-rt_i^*} g(S(t_i^*)) \quad (2.6)$$

One way to simplify the algorithm is to use the discounted payoffs P_i in the Monte Carlo step instead of the optimal stopping times. Since they are constructed and updated recursively in the same way as the stopping times, by the time we have gone from time $t = t_N$ to $t = t_1$ they will be

$$P_i = e^{-r(t_i^* - t_1)} g(S(t_i^*))$$

which means that

$$e^{-r\Delta t} P_i = e^{-rt_i^*} g(S(t_i^*))$$

and thus Equation 2.6 becomes

$$\hat{u} = \frac{1}{M} \sum_{i=1}^M e^{-r\Delta t} P_i \quad (2.7)$$

The entire LSM algorithm is described in Algorithm 2 below. In each step only the paths that are in-the-money are used, since these are the only ones where the decision to exercise or continue is relevant.

Algorithm 2 LSM Algorithm

Initiate paths $S_i(t), t = 0, t_1, t_2, \dots, t_N, i = 1, 2, \dots, M$
 Put $P_i \leftarrow g(S_i(t_N))$ for all i
for t from t_{N-1} to t_1 **do**
 Find paths $\{i_1, i_2, \dots, i_n\}$ s.t. $g(S_i(t)) > 0$, i.e. that are *in-the-money*
 Let $itm_paths \leftarrow \{i_1, i_2, \dots, i_n\}$
 Let $x_i \leftarrow S_i(t)$ and $y_i \leftarrow e^{-r\Delta t} P_i$ for $i \in itm_paths$
 Perform regression on x, y to obtain coefficients $\hat{\beta}_0, \dots, \hat{\beta}_k$
 Estimate the value of continuation $\hat{C}(S_i(t))$ and calculate the value of
 immediate exercise $g(S_i(t))$ for $i \in itm_paths$
 for i from 1 to M **do**
 if $i \in itm_paths$ **and** $g(S_i(t)) > \hat{C}(S_i(t))$ **then**
 $P_i \leftarrow g(S_i(t))$
 else
 $P_i \leftarrow e^{-r\Delta t} P_i$
 end if
 end for
end for
 $price \leftarrow \frac{1}{M} \sum_{i=1}^M e^{-r\Delta t} P_i$

The Problem

One of the most important problems in option pricing theory is the valuation and optimal exercise of derivatives with American-style exercise features. These types of derivatives are found in all major financial markets including the equity, commodity, foreign exchange, insurance, energy, sovereign, agency, municipal, mortgage, credit, real estate, convertible, swap, and emerging markets. Despite recent advances, however, the valuation and optimal exercise of American options remains one of the most challenging problems in derivatives finance, particularly when more than one factor affects the value of the option. This is primarily because finite difference and binomial techniques become impractical in situations where there are multiple factors.

We discuss the Longstaff and Schwartz approach of calculating the American Put option pricing using historical data that we have available with us of the different stock markets and we further compare this result with different other approaches as well. We use the following additional approaches to compare the results received from the Longstaff Schwartz Method:

1. Geometric Brownian Motion (GBM) Model also known as the Multi-Dimensional Market Model
2. Gaussian Copula Method that uses a Bivariate Gaussian Distribution to calculate the American Call and Put Option price.
3. t-Copula Call and Put Option Price.

We will simulate risk neutral sample paths for equity portfolios and this simulation is done using Monte Carlo Methods.

Overview of the Modeling Framework

The ultimate objective of this example is to compare basket option prices derived from different noise processes. The first noise process is a traditional Brownian motion model whose index portfolio price process is driven by correlated Gaussian random draws. As an alternative, the Brownian motion benchmark is compared to noise processes driven by Gaussian and Student's t copulas, referred to collectively as a *Brownian copula*.

A copula is a multivariate cumulative distribution function (CDF) with uniformly-distributed margins. Although the theoretical foundations were established decades ago, copulas have experienced a tremendous surge in popularity over the last few years, primarily as a technique for modeling non-Gaussian portfolio risks.

Although numerous families exist, all copulas represent a statistical device for modeling the dependence structure between two or more random variables. In addition, important statistics, such as *rank correlation* and *tail dependence* are properties of a given copula and are unchanged by monotonic transforms of their margins.

These copula draws produce dependent random variables, which are subsequently transformed to individual variables (margins). This transformation is achieved with a semi-parametric probability distribution with generalized Pareto tails.


The risk-neutral market model to simulate is

$$dX_t = rX_t dt + \sigma X_t dW_t$$

where the risk-free rate, r , is assumed constant over the life of the option. Because this is a separable multivariate model, the risk-free return is a diagonal matrix in which the same riskless return is applied to all indices. Dividend yields are ignored to simplify the model and its associated data collection.

In contrast, the specification of the exposure matrix, σ , depends on how the driving source of uncertainty is modeled. We can model it directly as a Brownian motion (correlated Gaussian random numbers implicitly mapped to Gaussian margins) or model it as a Brownian copula (correlated Gaussian or t random numbers explicitly mapped to semi-parametric margins).

Because the CDF and inverse CDF (quantile function) of univariate distributions are both monotonic transforms, a copula provides a convenient way to simulate dependent random variables whose margins are dissimilar and arbitrarily distributed. Moreover, because a copula defines a given dependence structure regardless of its margins, copula parameter calibration is typically easier than estimation of the joint distribution function.



Once you have simulated sample paths, options are priced by the least squares regression method of Longstaff & Schwartz (see *Valuing American Options by Simulation: A Simple Least-Squares Approach*, The Review of Financial Studies, Spring 2001). This approach uses least squares to estimate the expected payoff of an option if it is not immediately exercised. It does so by regressing the discounted option cash flows received in the future on the current price of the underlier associated with all in-the-money sample paths. The continuation function is estimated by a simple third-order polynomial, in which all cash flows and prices in the regression are normalized by the option strike price, improving numerical stability.

Code and Model

Import the Supporting Historical Dataset

Load a daily historical dataset of 3-month Euribor, the trading dates spanning the interval 07-Feb-2001 to 24-Apr-2006, and the closing index levels of the following representative large-cap equity indices:

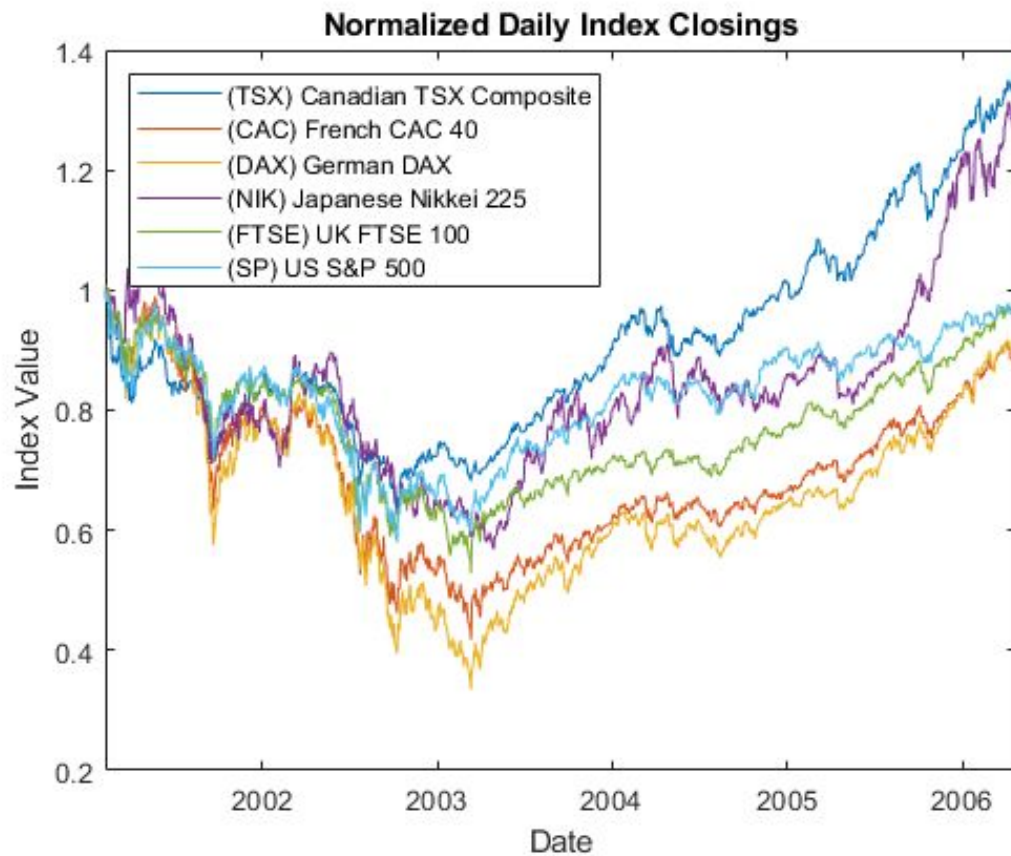
1. TSX Composite (Canada)
2. CAC 40 (France)
3. DAX (Germany)
4. Nikkei 225 (Japan)
5. FTSE 100 (UK)
6. S&P 500 (US)

```
clear; clc; close all;
load Data_GlobalIdx2;
dates = datetime(dates, 'ConvertFrom', 'datenum');
```

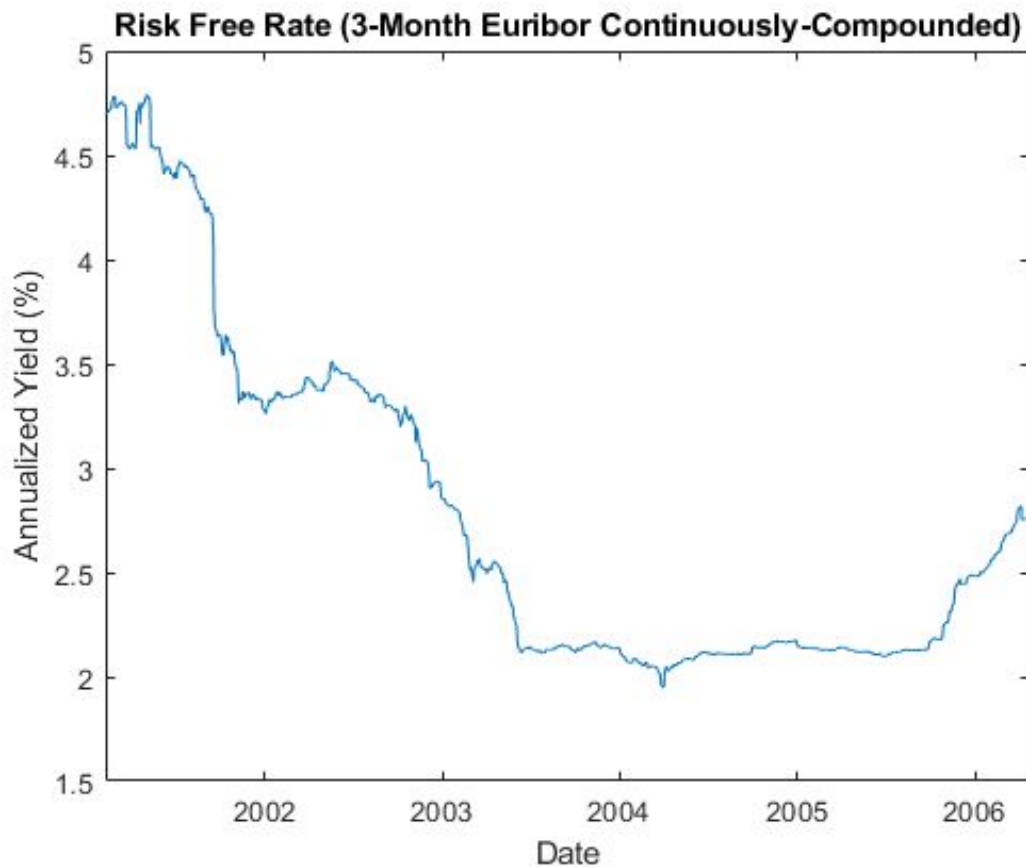
The following plots illustrate this data. Specifically, the plots show the relative price movements of each index and the Euribor risk-free rate proxy. The initial level of each index has been normalized to unity to facilitate the comparison of relative performance over the historical record.

```
nIndices = size(Data,2) - 1;      % # of indices
prices = Data(:,1 : end-1);
yields = Data(:, end);           % daily effective yields
yields = 360 * log(1 + yields);   % continuously-compounded, annualized yield

figure;
plot(dates,
ret2tick(tick2ret(prices, 'Method', 'continuous'), 'Method', 'continuous'))
xlabel('Date')
ylabel('Index Value')
title('Normalized Daily Index Closings')
legend(series{1:end-1}, 'Location', 'NorthWest')
```



```
plot(dates, 100 * yields)
xlabel('Date')
ylabel('Annualized Yield (%)')
title('Risk Free Rate (3-Month Euribor Continuously-Compounded)')
```



Extreme Value Theory & Piecewise Probability Distributions

To prepare for copula modeling, characterize individually the distribution of returns of each index. Although the distribution of each return series may be characterized parametrically, it is useful to fit a semi-parametric model using a piecewise distribution with generalized Pareto tails. This uses Extreme Value Theory to better characterize the behavior in each tail.

The Statistics and Machine Learning Toolbox™ software currently supports two univariate probability distributions related to EVT, a statistical tool for modeling the fat-tailed behavior of financial data such as asset returns and insurance losses:

1. Generalized Extreme Value (GEV) distribution, which uses a modeling technique known as the *block maxima or minima* method. This approach divides a historical dataset into a set of sub-intervals, or blocks, and the largest or smallest observation in each block is recorded and fitted to a GEV distribution.
2. Generalized Pareto (GP) distribution, uses a modeling technique known as the *distribution of exceedances or peaks over threshold* method. This approach sorts a

historical dataset and fits the amount by which those observations that exceed a specified threshold to a GP distribution.

The following analysis highlights the Pareto distribution, which is more widely used in risk management applications.

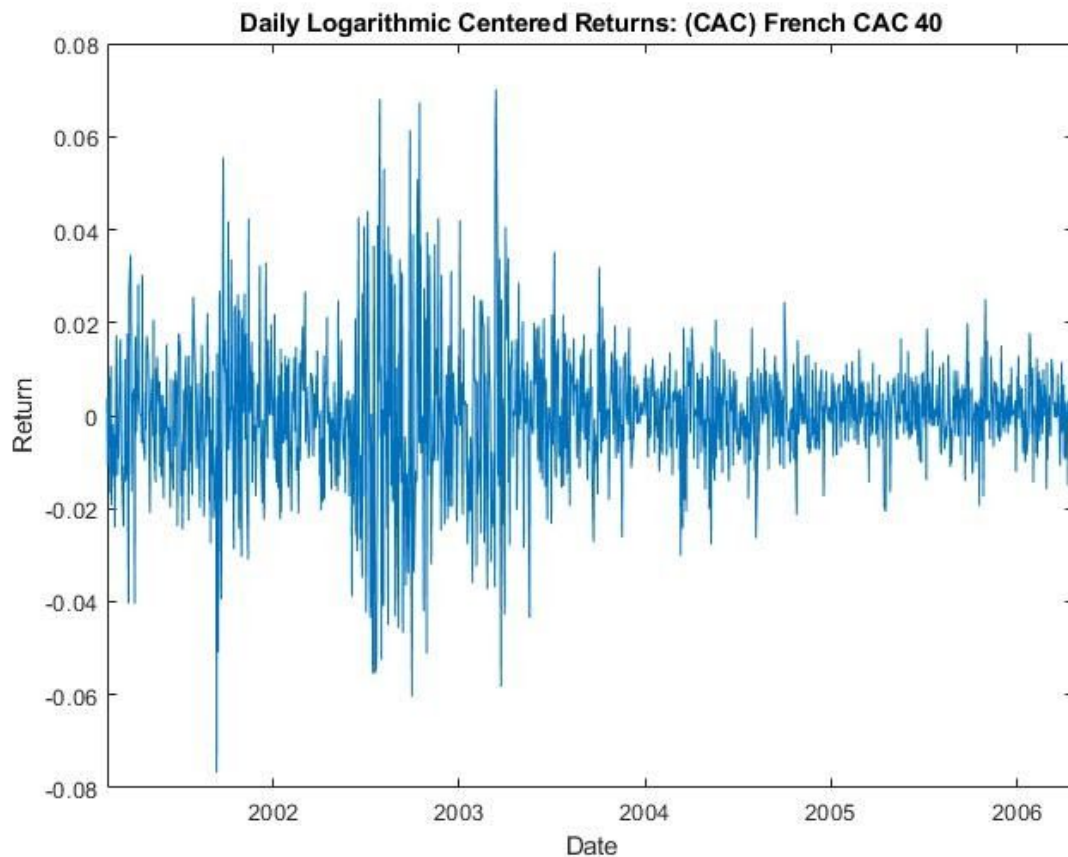
Suppose we want to create a complete statistical description of the probability distribution of daily asset returns of any one of the equity indices. Assume that this description is provided by a piecewise semi-parametric distribution, where the asymptotic behavior in each tail is characterized by a generalized Pareto distribution.

Ultimately, a copula will be used to generate random numbers to drive the simulations. The CDF and inverse CDF transforms will capture the volatility of simulated returns as part of the diffusion term of the SDE. The mean return of each index is governed by the riskless rate and incorporated in the drift term of the SDE. The following code segment centers the returns (that is, extracts the mean) of each index.

Because the following analysis uses extreme value theory to characterize the distribution of each equity index return series, it is helpful to examine details for a particular country:

```
returns = tick2ret(prices, 'Method', 'continuous'); % convert prices to returns
returns = returns - mean(returns); % center the returns
index    = 2; % France stored in column 2

figure;
plot(dates(2:end), returns(:,index))
xlabel('Date')
ylabel('Return')
title(['Daily Logarithmic Centered Returns: ' series{index}])
```



Note that this code segment can be changed to examine details for any country.

Using these centered returns, we estimate the empirical, or non-parametric, CDF of each index with a Gaussian kernel. This smoothes the CDF estimates, eliminating the staircase pattern of unsmoothed sample CDFs. Although non-parametric kernel CDF estimates are well-suited for the interior of the distribution, where most of the data is found, they tend to perform poorly when applied to the upper and lower tails. To better estimate the tails of the distribution, we apply EVT to the returns that fall in each tail.

Specifically, we find the upper and lower thresholds such that 10% of the returns are reserved for each tail. Then we fit the amount by which the extreme returns in each tail fall beyond the associated threshold to a Pareto distribution by maximum likelihood.

The following code segment creates one object of type `paretotails` for each index return series. These Pareto tail objects encapsulate the estimates of the parametric Pareto lower tail, the non-parametric kernel-smoothed interior, and the parametric Pareto upper tail to construct a composite semi-parametric CDF for each index.

```

tailFraction = 0.1; % decimal fraction allocated to each tail
tails = cell(nIndices,1); % cell array of Pareto tail objects
for i = 1:nIndices
    tails{i} = paretotails(returns(:,i), tailFraction, 1 - tailFraction,
'kernel');
end

```

The resulting piecewise distribution object allows interpolation within the interior of the CDF and extrapolation (function evaluation) in each tail. Extrapolation allows estimation of quantiles outside the historical record, which is invaluable for risk management applications.

Pareto tail objects also provide methods to evaluate the CDF and inverse CDF (quantile function), and to query the cumulative probabilities and quantiles of the boundaries between each segment of the piecewise distribution.

Now that three distinct regions of the piecewise distribution have been estimated, we graphically concatenate and display the result.

The following code calls the CDF and inverse CDF methods of the Pareto tails object of interest with data other than that upon which the fit is based. The referenced methods have access to the fitted state. They are now invoked to select and analyze specific regions of the probability curve, acting as a powerful data filtering mechanism.

For reference, the plot also includes a zero-mean Gaussian CDF of the same standard deviation. To a degree, the variation in options prices reflect the extent to which the distribution of each asset differs from this normal curve.

```

minProbability = cdf(tails{index}, (min(returns(:,index))));
maxProbability = cdf(tails{index}, (max(returns(:,index))));

pLowerTail = linspace(minProbability , tailFraction , 200); % lower tail
pUpperTail = linspace(1 - tailFraction, maxProbability , 200); % upper tail
pInterior = linspace(tailFraction , 1 - tailFraction, 200); % interior

figure;
plot(icdf(tails{index}, pLowerTail), pLowerTail, 'red' , 'LineWidth', 2);
hold on;

```



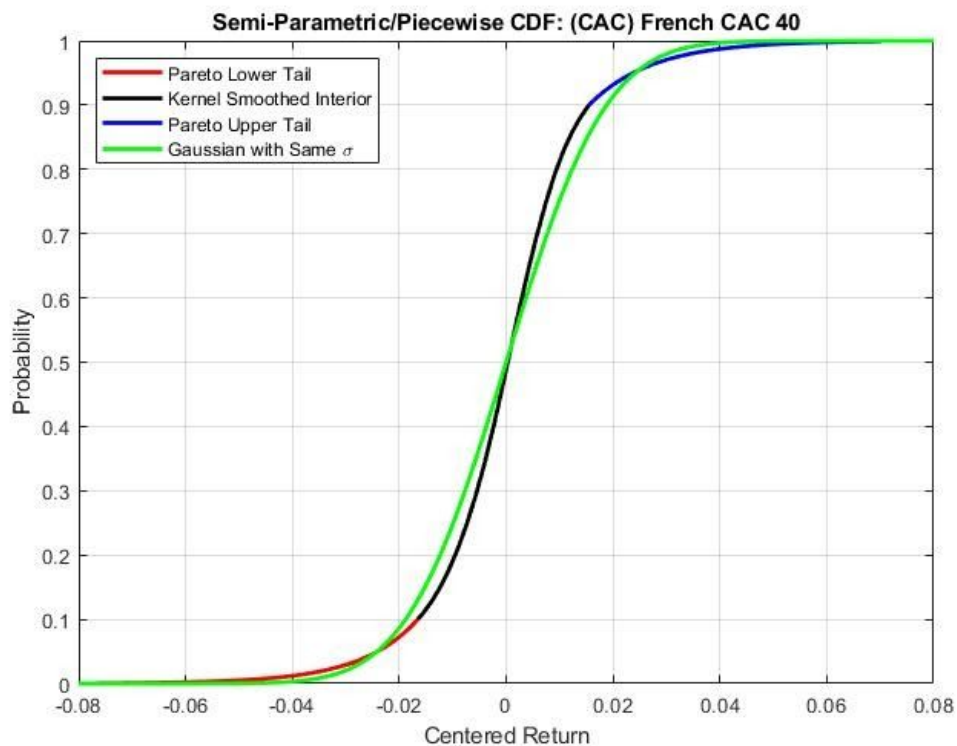
```

grid on;
plot(icdf(tails{index}, pInterior) , pInterior , 'black', 'LineWidth', 2);
plot(icdf(tails{index}, pUpperTail), pUpperTail, 'blue' , 'LineWidth', 2);

limits = axis;
x = linspace(limits(1), limits(2));
plot(x, normcdf(x, 0, std(returns(:,index))), 'green', 'LineWidth', 2);

fig = gcf;
fig.Color = [1 1 1];
hold off;
xlabel('Centered Return');
ylabel('Probability');
title(['Semi-Parametric/Piecewise CDF: ' series{index}]);
legend({'Pareto Lower Tail' 'Kernel Smoothed Interior' ...
        'Pareto Upper Tail' 'Gaussian with Same \sigma'}, 'Location',
'NorthWest');

```



The lower and upper tail regions, displayed in red and blue, respectively, are suitable for extrapolation, while the kernel-smoothed interior, in black, is suitable for interpolation.

Copula Calibration

We will now calibrate and simulate Gaussian and t copulas.

Using the daily index returns, we estimate the parameters of the Gaussian and t copulas using the function `copulafit`. Since a t copula becomes a Gaussian copula as the scalar degrees of freedom parameter (DoF) becomes infinitely large, the two copulas are really of the same family, and therefore share a linear correlation matrix as a fundamental parameter.

Although calibration of the linear correlation matrix of a Gaussian copula is straightforward, the calibration of a t copula is not. For this reason, the Statistics and Machine Learning Toolbox software offers two techniques to calibrate a t copula:

The first technique performs maximum likelihood estimation (MLE) in a two-step process. The inner step maximizes the log-likelihood with respect to the linear correlation matrix, given a fixed value for the degrees of freedom. This conditional maximization is placed within a 1-D maximization with respect to the degrees of freedom, thus maximizing the log-likelihood over all parameters. The function being maximized in this outer step is known as the profile log-likelihood for the degrees of freedom.

The second technique is derived by differentiating the log-likelihood function with respect to the linear correlation matrix, assuming the degrees of freedom is a fixed constant. The resulting expression is a non-linear equation that can be solved iteratively for the correlation matrix. This technique approximates the profile log-likelihood for the degrees of freedom parameter for large sample sizes. This technique is usually significantly faster than the true maximum likelihood technique outlined above; however, you should not use it with small or moderate sample sizes as the estimates and confidence limits may not be accurate.

When the uniform variates are transformed by the empirical CDF of each margin, the calibration method is often known as canonical maximum likelihood (CML). The following code segment first transforms the daily centered returns to uniform variates by the piecewise, semi-parametric CDFs derived above. It then fits the Gaussian and t copulas to the transformed data:

```
U = zeros(size(returns));
for i = 1:nIndices
    U(:,i) = cdf(tails{i}, returns(:,i));    % transform each margin to uniform
end
```

```

options      = statset('Display', 'off', 'TolX', 1e-4);
[rhoT, DoF] = copulafit('t', U, 'Method', 'ApproximateML', 'Options', options);
rhoG        = copulafit('Gaussian', U);

```

The estimated correlation matrices are quite similar but not identical.

```
corrcoef(returns) % linear correlation matrix of daily returns
```

```

1.0000  0.4813  0.5058  0.1854  0.4573  0.6526
0.4813  1.0000  0.8485  0.2261  0.8575  0.5102
0.5058  0.8485  1.0000  0.2001  0.7650  0.6136
0.1854  0.2261  0.2001  1.0000  0.2295  0.1439
0.4573  0.8575  0.7650  0.2295  1.0000  0.4617
0.6526  0.5102  0.6136  0.1439  0.4617  1.0000

```

```
rhoG % linear correlation matrix of the optimized Gaussian copula
```

```

1.0000  0.4745  0.5018  0.1857  0.4721  0.6622
0.4745  1.0000  0.8606  0.2393  0.8459  0.4912
0.5018  0.8606  1.0000  0.2126  0.7608  0.5811
0.1857  0.2393  0.2126  1.0000  0.2396  0.1494
0.4721  0.8459  0.7608  0.2396  1.0000  0.4518
0.6622  0.4912  0.5811  0.1494  0.4518  1.0000

```

```
rhoT % linear correlation matrix of the optimized t copula
```

```
linear correlation matrix of the optimized t copula
```

```

1.0000  0.4671  0.4858  0.1907  0.4734  0.6521
0.4671  1.0000  0.8871  0.2567  0.8500  0.5122
0.4858  0.8871  1.0000  0.2326  0.7723  0.5877
0.1907  0.2567  0.2326  1.0000  0.2503  0.1539
0.4734  0.8500  0.7723  0.2503  1.0000  0.4769
0.6521  0.5122  0.5877  0.1539  0.4769  1.0000

```

```
DoF % scalar degrees of freedom parameter of the optimized t copula
```

```
4.8613
```

Copula Simulation

Now that the copula parameters have been estimated, we will simulate jointly-dependent uniform variates using the function `copularnd`.

Then, by extrapolating the Pareto tails and interpolating the smoothed interior, transform the uniform variates derived from `copularnd` to daily centered returns via the inverse CDF of each index. These simulated centered returns are consistent with those obtained from the historical dataset. The returns are assumed to be independent in time, but at any point in time possess the dependence and rank correlation induced by the given copula.

The following code segment illustrates the dependence structure by simulating centered returns using the t copula. It then plots a 2-D scatter plot with marginal histograms for the French CAC 40 and German DAX using the Statistics and Machine Learning Toolbox `scatterhist` function. The French and German indices were chosen simply because they have the highest correlation of the available data.

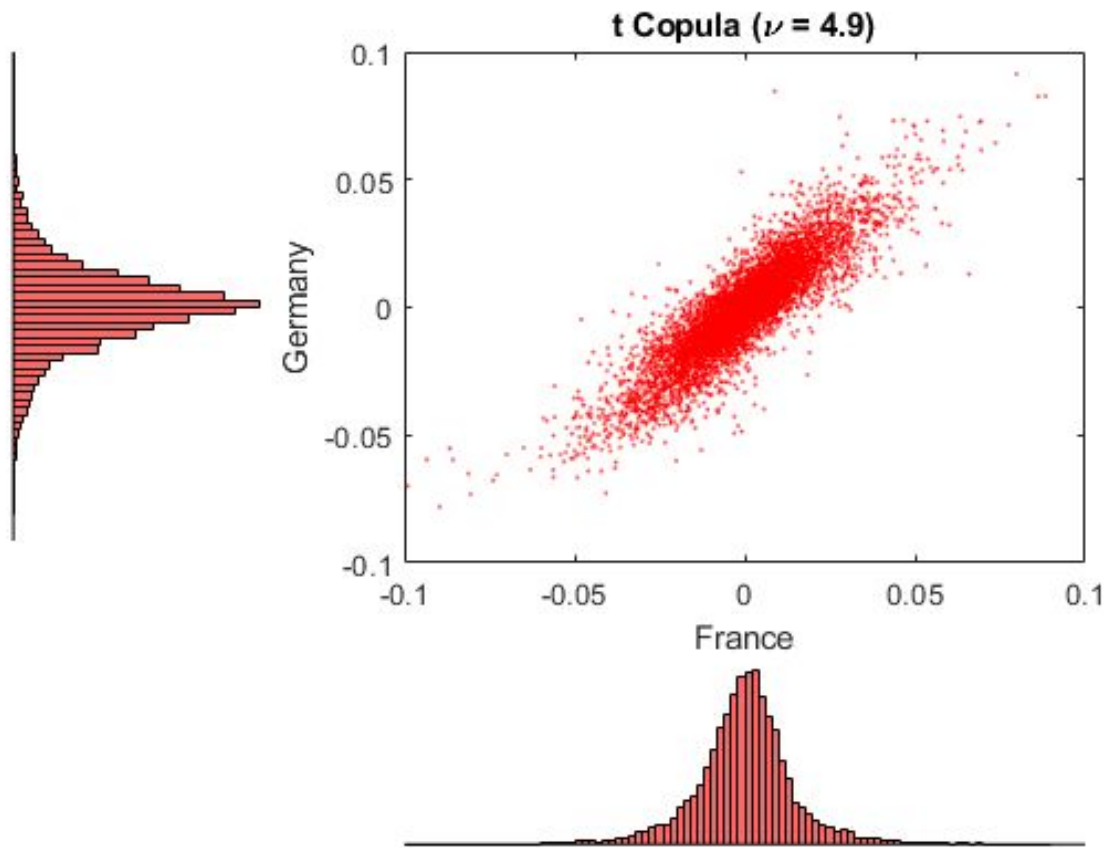
```
nPoints = 10000; % # of simulated observations

s = RandStream.getGlobalStream();
reset(s)

R = zeros(nPoints, nIndices); % pre-allocate simulated returns array
U = copularnd('t', rhoT, DoF, nPoints); % simulate U(0,1) from t copula

for j = 1:nIndices
    R(:,j) = icdf(tails{j}, U(:,j));
end

h = scatterhist(R(:,2), R(:,3), 'Color', 'r', 'Marker', '.', 'MarkerSize', 1);
fig = gcf;
fig.Color = [1 1 1];
y1 = ylim(h(1));
y3 = ylim(h(3));
xlim(h(1), [-.1 .1])
ylim(h(1), [-.1 .1])
xlim(h(2), [-.1 .1])
ylim(h(3), [(y3(1) + (-0.1 - y1(1))) (y3(2) + (0.1 - y1(2)))])
xlabel('France')
ylabel('Germany')
title(['t Copula (\nu = ' num2str(DoF,2) ')'])
```



Simulation Using Gaussian Copula

Now simulate and plot centered returns using the Gaussian copula.

```
reset(s);
R = zeros(nPoints, nIndices); % pre-allocate simulated returns array
U = copularnd('Gaussian', rhoG, nPoints); % simulate U(0,1) from Gaussian
copula

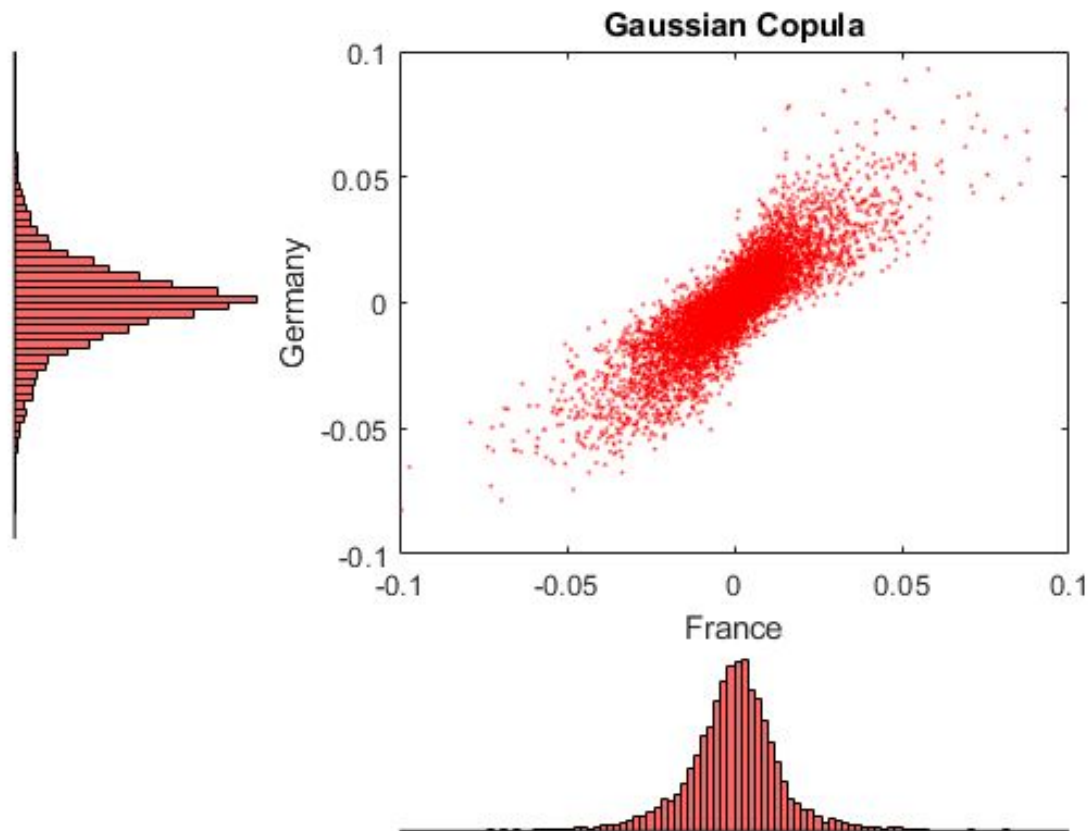
for j = 1:nIndices
    R(:,j) = icdf(tails{j}, U(:,j));
end

h = scatterhist(R(:,2), R(:,3), 'Color', 'r', 'Marker', '.', 'MarkerSize', 1);
fig = gcf;
fig.Color = [1 1 1];
y1 = ylim(h(1));
y3 = ylim(h(3));
xlim(h(1), [-.1 .1])
```

```

ylim(h(1), [-.1 .1])
xlim(h(2), [-.1 .1])
ylim(h(3), [(y3(1) + (-0.1 - y1(1))) (y3(2) + (0.1 - y1(2)))])
xlabel('France')
ylabel('Germany')
title('Gaussian Copula')

```



There is a strong similarity between the miniature histograms on the corresponding axes of each figure. This similarity is not coincidental.

Both copulas simulate uniform random variables, which are then transformed to daily centered returns by the inverse CDF of the piecewise distribution of each index. Therefore, the simulated returns of any given index are identically distributed regardless of the copula.

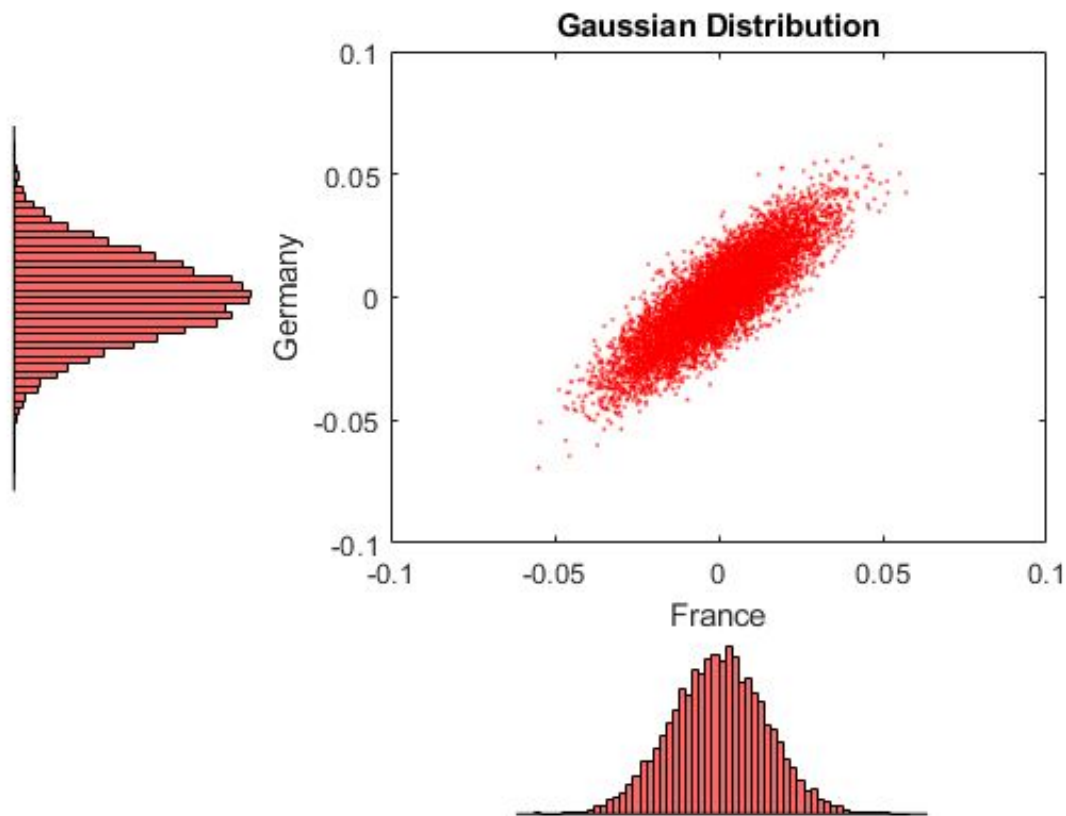
However, the scatter graph of each figure indicates the dependence structure associated with the given copula, and in contrast to the univariate margins shown in the histograms, the scatter graphs are distinct.

Once again, the copula defines a dependence structure regardless of its margins, and therefore offers many features not limited to calibration alone.

Simulation Using Traditional Brownian Motion Model

For reference, simulate and plot centered returns using the Gaussian distribution, which underlies the traditional Brownian motion model.

```
reset(s);
R = mvnrnd(zeros(1,nIndices), cov(returns), nPoints);
h = scatterhist(R(:,2), R(:,3), 'Color','r','Marker','.', 'MarkerSize',1);
fig = gcf;
fig.Color = [1 1 1];
y1 = ylim(h(1));
y3 = ylim(h(3));
xlim(h(1), [-.1 .1])
ylim(h(1), [-.1 .1])
xlim(h(2), [-.1 .1])
ylim(h(3), [(y3(1) + (-0.1 - y1(1))) (y3(2) + (0.1 - y1(2)))])
xlabel('France')
ylabel('Germany')
title('Gaussian Distribution')
```



Longstaff & Schwartz Approach

We create a structure Object called the LongstaffSchwartz Object in MATLAB and add the following fields to it:

1. saveBasketPrices
2. getCallPrice
3. getPutPrice
4. getBasketPrices

We create it as :

```
function f = LongstaffSchwartz(numTimes, numPaths)
% Syntax:
%
% f = LongstaffSchwartz(numTimes, numPaths)
```



```

%
% Description:
%
% End-of-period processing function to price American basket options on a
% portfolio of assets by Monte Carlo simulation, assuming a constant
% risk-free rate. The valuation is based on the technique of Longstaff
% and Schwartz [1], and approximates the continuation function by least
% squares regression using a 3rd-order polynomial:  $y = F(x) = a + b*x +$ 
%  $c*x^2 + d*x^3$ . The function also illustrates how to update, access, and
% share information among several nested functions.
%
% Input Arguments:
%
% numTimes - Number of simulation times steps (numPeriods * numSteps)
% numPaths - Number of independent sample paths (simulation trials)
%
% Output Argument:
%
% f - Structure of nested function handles to price American options.
%
% Reference:
%
% [1] Longstaff F.A., and E.S. Schwartz. "Valuing American Options by
% Simulation: A Simple Least-Squares Approach." The Review of Financial
% Studies. Vol. 14, 2001, pp. 113-147.

prices = zeros(numPaths,numTimes); % Pre-allocate price matrix
times = zeros(numTimes,1); % Pre-allocate sample time vector
iTime = 0; % Counter for the period index
iPath = 1; % Counter for the trial index
tStart = 0; % Initialize starting time
weights = 1; % Initialize portfolio weights

f.LongstaffSchwartz = @saveBasketPrices; % End-of-period processing
f.CallPrice = @getCallPrice; % Call option pricing utility
f.PutPrice = @getPutPrice; % Put option pricing utility
f.Prices = @getBasketPrices; % Portfolio price access utility

function X = saveBasketPrices(t,X) % The actual workhorse, f(t,X)
    if iTime == 0 % Account for initial validation

```

```

iTime = 1;
tStart = t;          % Record time of the initial condition
weights = ones(1,numel(X)); % Assume 1 share of each asset
else % Simulation is live

% The following line forms the portfolio price series by adding the
% current prices of the individual assets, converting an N-D portfolio
% of equities into a 1-D basket upon which a basket option is priced.
% The portfolio price series represents the total purchase price of a
% single share of each individual asset, although the weight vector
% could be anything.

prices(iPath, iTime) = weights * X;

if iPath == 1
    times(iTime) = t; % Save the sample time
end
if iTime < numTimes
    iTime = iTime + 1; % Update the period counter
else % The last period of the current trial
    iTime = 1;        % Re-set the time period counter
    iPath = iPath + 1; % A new trial has begun
end
end
end

function value = getOptionPrice(strike,rate,optionClass)

% Model the continuation function as a 3rd-order polynomial.

F = @(x,a,b,c,d)(a + b*x + c*x.^2 + d*x.^3);

% Pre-allocate option cash flow matrix, and initialize the terminal value
% at expiration to the cash flow of its European counterpart:

V = zeros(numPaths, numTimes); % Option cash flow matrix

if (nargin <= 2) || strcmpi(optionClass,'Call')
    isCallOption = true; % Call option
    V(:,end) = max(prices(:,end)-strike, 0);

```

```

else
    isCallOption = false; % Put option
    V(:,end) = max(strike-prices(:,end),0);
end

% Step backward through time, successively updating the option cash flow
% matrix via OLS regression of in-the-money sample paths:

for iTime = (numTimes-1):-1:1

    % Find all in-the-money sample paths, and format the regression
    % arrays. If no in-the-money paths are found, then assume the option
    % is not exercised:

    if isCallOption
        inTheMoney = find(prices(:,iTime) > strike); % Call option
    else
        inTheMoney = find(prices(:,iTime) < strike); % Put option
    end

    if ~isempty(inTheMoney)

        % For all in-the-money sample paths, identify any positive option
        % cash flows that occur after the current sample time, the sample
        % times at which the cash flows occur, and the column indices of
        % the cash flow matrix associated with them. If there is no
        % subsequent cash flow for a given path, then set the column index
        % to one just so indexing operations do not produce an error.

        iCashFlows = V(inTheMoney,(iTime+1):end) > 0; % Find positive CFs
        tNext = iCashFlows*times((iTime+1):numTimes); % Time of next CF
        iNext = max(iCashFlows*((iTime+1):numTimes)',1); % Index of next CF

        % Format the regression matrices, which include the current prices
        % of the underlier of all in-the-money sample paths (X), and the
        % discounted value of subsequent option cash flows associated with
        % the same paths (Y). If there is no subsequent cash flow for a
        % given path, then set the discount factor to zero just to be

        % clear. The following code segment assumes a constant risk-free

```

% discount rate.

% To improve numerical stability, normalize X and Y by the strike
% price:

```
X = prices(inTheMoney,iTime)/strike; % In-the-money prices
D = exp(-rate*(tNext-times(iTime))); % Discount factors
D(D > 1) = 0; % Allow for no future CFs
Y = (V(sub2ind(size(V),inTheMoney,iNext)).*D)/strike;
```

% Perform the OLS regression:

```
OLS = [ones(numel(X),1) X X.^2 X.^3]\Y;
```

% Determine the intrinsic value of immediate exercise and the
% value of continuation (scaled back to actual, unnormalized CFs):

```
continuationValue = F(X,OLS(1),OLS(2),OLS(3),OLS(4))*strike;
X = X*strike;
```

```
if isCallOption
    intrinsicValue = max(X-strike,0);
else
    intrinsicValue = max(strike-X,0);
end
```

```
iExercise = intrinsicValue > continuationValue;
```

% Update the option cash flow matrix if immediate exercise is more
% valuable than continuation. Note that continuation values are
% not inserted into the option cash flow matrix, but rather
% determine whether or not intrinsic values are. Also, since the
% option can only be exercised once, overwrite all non-zero cash
% flows after the current time.

```
V(inTheMoney(iExercise),iTime) = intrinsicValue(iExercise);
V(inTheMoney(iExercise),(iTime+1):end) = 0;
```

```
end
end
```

```

% Value the option by discounting the cash flows in the option cash flow
% matrix back to the initial time (tStart), assuming a constant risk-free
% discount rate:

stoppingTimes = (V > 0)*times;
value = mean(sum(V,2).*exp(-rate.*(stoppingTimes-tStart)));

end

function value = getCallPrice(strike,rate)
    value = getOptionPrice(strike,rate,'Call');
end

function value = getPutPrice(strike,rate)
    value = getOptionPrice(strike,rate,'Put');
end

function value = getBasketPrices()
    value = permute(prices,[2 3 1]); % Re-order to time-series format
end

end % End of outer/primary function

```

Monte Carlo Simulations

Monte Carlo methods, or **Monte Carlo experiments**, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three problem classes: optimization, numerical integration, and generating draws from a probability distribution.

In physics-related problems, Monte Carlo methods are useful for simulating systems with many coupled degrees of freedom, such as fluids, disordered materials, strongly coupled solids, and cellular structures.

Other examples include modeling phenomena with significant uncertainty in inputs such as the calculation of risk in business and, in mathematics, evaluation of multidimensional definite integrals with complicated boundary conditions. In application to systems engineering problems (space, oil exploration, aircraft design, etc.), Monte Carlo-based

predictions of failure, cost overruns and schedule overruns are routinely better than human intuition or alternative "soft" methods.

In principle, Monte Carlo methods can be used to solve any problem having a probabilistic interpretation. By the law of large numbers, integrals described by the expected value of some random variable can be approximated by taking the empirical mean (a.k.a. the sample mean) of independent samples of the variable. When the probability distribution of the variable is parametrized, mathematicians often use a Markov chain Monte Carlo (MCMC) sampler. The central idea is to design a judicious Markov chain model with a prescribed stationary probability distribution. That is, in the limit, the samples being generated by the MCMC method will be samples from the desired (target) distribution. By the ergodic theorem, the stationary distribution is approximated by the empirical measures of the random states of the MCMC sampler.

In other problems, the objective is generating draws from a sequence of probability distributions satisfying a nonlinear evolution equation. These flows of probability distributions can always be interpreted as the distributions of the random states of a Markov process whose transition probabilities depend on the distributions of the current random states. In other instances we are given a flow of probability distributions with an increasing level of sampling complexity (path spaces models with an increasing time horizon, Boltzmann–Gibbs measures associated with decreasing temperature parameters, and many others). These models can also be seen as the evolution of the law of the random states of a nonlinear Markov chain. A natural way to simulate these sophisticated nonlinear Markov processes is to sample multiple copies of the process, replacing in the evolution equation the unknown distributions of the random states by the sampled empirical measures. In contrast with traditional Monte Carlo and MCMC methodologies these mean field particle techniques rely on sequential interacting samples. The terminology *mean field* reflects the fact that each of the *samples* (a.k.a. particles, individuals, walkers, agents, creatures, or phenotypes) interacts with the empirical measures of the process. When the size of the system tends to infinity, these random empirical measures converge to the deterministic distribution of the random states of the nonlinear Markov chain, so that the statistical interaction between particles vanishes.

American Option Pricing Using the Longstaff & Schwartz Approach

Now that the copulas have been calibrated, we compare the prices of at-the-money American basket options derived from various approaches. To simplify the analysis, we assume that:

1. All indices begin at 100.

2. The portfolio holds a single unit, or share, of each index such that the value of the portfolio at any time is the sum of the values of the individual indices.
3. The option expires in 3 months.
4. The information derived from the daily data is annualized.
5. Each calendar year is composed of 252 trading days.
6. Index levels are simulated daily.
7. The option may be exercised at the end of every trading day and approximates the American option as a Bermudan option.

Now compute the parameters common to all simulation methods:

```
dt      = 1 / 252;           % time increment = 1 day = 1/252 years
yields  = Data(:,end);       % daily effective yields
yields  = 360 * log(1 + yields); % continuously-compounded & annualized
r       = mean(yields);      % historical 3M Euribor average
X       = repmat(100, nIndices, 1); % initial state vector
strike  = sum(X);            % initialize an at-the-money basket

nTrials  = 100;              % # of independent trials
nPeriods = 63;              % # of simulation periods: 63/252 = 0.25 years = 3 months
```

Now we create two separable multi-dimensional market models in which the riskless return and volatility exposure matrices are both diagonal.

While both are diagonal GBM models with identical risk-neutral returns, the first is driven by a correlated Brownian motion and explicitly specifies the sample linear correlation matrix of centered returns. This correlated Brownian motion process is then weighted by a diagonal matrix of annualized index volatilities or standard deviations.

As an alternative, the same model could be driven by an uncorrelated Brownian motion (*standard Brownian motion*) by specifying `correlation` as an identity matrix, or by simply accepting the default value. In this case, the exposure matrix `sigma` is specified as the lower Cholesky factor of the index return covariance matrix. Because the copula-based approaches simulate dependent random numbers, the diagonal exposure form is chosen for consistency.

```
sigma      = std(returns) * sqrt(252); % annualized volatility
correlation = corrcoef(returns);       % correlated Gaussian disturbances
GBM1       = gbm(diag(r(ones(1,nIndices))), diag(sigma), 'StartState', X, ...
                 'Correlation', correlation);
```

Now we create the second model driven by the Brownian copula with an identity matrix `sigma`.

```
GBM2 = gbm(diag(r(ones(1,nIndices))), eye(nIndices), 'StartState', X);
```

We now fit the Gaussian and t copula dependence structures, respectively, and the semi-parametric margins to the centered returns scaled by the square root of the number of trading days per year (252). This scaling does not annualize the daily centered returns. Instead, it scales them such that the volatility remains consistent with the diagonal annualized exposure matrix `sigma` of the traditional Brownian motion model (GBM1) created previously.

We also specify an end-of-period processing function that accepts time followed by state (t, X) , and records the sample times and value of the portfolio as the single-unit weighted average of all indices. This function also shares this information with other functions designed to price American options with a constant riskless rate using the least squares regression approach of Longstaff & Schwartz.

```
f = LongstaffSchwartz(nPeriods, nTrials)
f = struct with fields:

    LongstaffSchwartz: @LongstaffSchwartz/saveBasketPrices
    CallPrice: @LongstaffSchwartz/getCallPrice
    PutPrice: @LongstaffSchwartz/getPutPrice
    Prices: @LongstaffSchwartz/getBasketPrices
```

Now we simulate independent trials of equity index prices over 3 calendar months using the default `simByEuler` method.

```
reset(s);

simByEuler(GBM1, nPeriods, 'nTrials' , nTrials, 'DeltaTime', dt, ...
          'Processes', f.LongstaffSchwartz);

BrownianMotionCallPrice = f.CallPrice(strike, r);
BrownianMotionPutPrice  = f.PutPrice (strike, r);

reset(s);

z = Example_CopulaRNG(returns * sqrt(252), nPeriods, 'Gaussian');
f = Example_LongstaffSchwartz(nPeriods, nTrials);
```



```
simByEuler(GBM2, nPeriods, 'nTrials' , nTrials, 'DeltaTime', dt, ...
           'Processes', f.LongstaffSchwartz, 'Z', z);
```

```
GaussianCopulaCallPrice = f.CallPrice(strike, r);
```

```
GaussianCopulaPutPrice = f.PutPrice (strike, r);
```

Now repeat the copula simulation with the t copula dependence structure. You use the same model object for both copulas; only the random number generator and option pricing functions need to be re-initialized.

```
reset(s);
```

```
z = Example_CopulaRNG(returns * sqrt(252), nPeriods, 't');
```

```
f = Example_LongstaffSchwartz(nPeriods, nTrials);
```

```
simByEuler(GBM2, nPeriods, 'nTrials' , nTrials, 'DeltaTime', dt, ...
           'Processes', f.LongstaffSchwartz, 'Z', z);
```

```
tCopulaCallPrice = f.CallPrice(strike, r);
```

```
tCopulaPutPrice = f.PutPrice (strike, r);
```

Finally, compare the American put and call option prices obtained from all models.

```
disp(' ');
fprintf('                # of Monte Carlo Trials: %8d\n' , nTrials);
fprintf('                # of Time Periods/Trial: %8d\n\n' , nPeriods);
fprintf(' Brownian Motion American Call Basket Price: %8.4f\n' ,
BrownianMotionCallPrice);
fprintf(' Brownian Motion American Put  Basket Price: %8.4f\n\n',
BrownianMotionPutPrice);
fprintf(' Gaussian Copula American Call Basket Price: %8.4f\n' ,
GaussianCopulaCallPrice);
fprintf(' Gaussian Copula American Put  Basket Price: %8.4f\n\n',
GaussianCopulaPutPrice);
fprintf('          t Copula American Call Basket Price: %8.4f\n' ,
tCopulaCallPrice);
fprintf('          t Copula American Put  Basket Price: %8.4f\n' ,
tCopulaPutPrice);
```

```
# of Monte Carlo Trials:      100
```

of Time Periods/Trial: 63

Brownian Motion American Call Basket Price: 25.9456

Brownian Motion American Put Basket Price: 16.4132

Gaussian Copula American Call Basket Price: 24.5711

Gaussian Copula American Put Basket Price: 17.4229

t Copula American Call Basket Price: 22.6220

t Copula American Put Basket Price: 20.9983

Bibliography

1. [Monte Carlo Methods](#) ~Wikipedia
2. [MATLAB](#) ~Mathworks
3. [MATLAB plot](#)
4. [MATLAB Gaussian Distribution](#)
5. [MATLAB Geometric Brownian Motion Model \(gbm\)](#)
6. [MATLAB Simulation](#)
7. [Probability Distribution](#) ~Wikipedia
8. [Valuing American Options by Simulation: A Simple Least Squares Approach](#) ~Francis A. Longstaff and Eduardo S. Schwartz at The University of California Los Angeles (UCLA)
9. [Longstaff Schwartz Pricing of Bermudan Options and Their Greeks](#)
10. [Monte Carlo Methods in Finance](#) ~Wikipedia
11. [Mathematical Modelling](#) ~Universitat Wein