ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

# Exotic Options: Pricing a Rainbow Option of Two Assets via Monte Carlo and Variance Reduction Techniques

Computational and Quantitative Finance with C++

Agianogloy Nikolaos-Vasileios:MFT2501
DelRosso Vasileios:MFT2509
Lathiotakis Efthymios:MFT2504
Pattas Panagiotis:MFT2510

Instructor : Englezos Nikolaos —University of Piraeus

- Exotic Options — Rainbow Options. They derive their value from multiple underlying assets (e.g two) and their payoff can have different forms. They theoretical foundation was provided by Margrabe (1978) and later on by Stulz (1982).
- Due to the multi-asset exposure under a single derivative contract they provide a more natural risk diversification,enhanced returns and a better cost efficiency.
- Forms of Rainbow Options are :
  - Best of assets or cash : $max(S_1, S_2, ...S_n, K)$
  - **Call on** max/**min** : $max(max/min(S_1, S_2, ...S_n) - K, 0.0)$
  - Put on max/min : $max(K - max/min(S_1, S_2, ...S_n), 0.0)$
  - Put 2 and Call 1 : $max(S_1 - S_2, 0.0)$

Those options types *follow a Geometric Brownian Motion under the risk-neutral meausure:*

- $dS_i(t) = rS_i(t)dt + \sigma_i S_i(t)dZ_i(t), i = 1, 2$ where $r$ represents the risk-free rate, $\sigma_i$ is the volatility per asset, $Z_i(t)$ are independent Standard Brownian Motions.
- It is given that the correlation between the two assets $Corr(S_1, S_2) \leftrightarrow Corr(dZ_1(t), dZ_2(t)) = 0$
- Also there are no dividends on either asset : $q_1 = q_2 = 0$

# Theoretical Pricing

Below is provided a quick summary of the solution on the Stohastic Differential Equation (SDE): $dS_i(t) = rS_i(t)dt + \sigma_i S_i(t)dZ_i(t), i = 1, 2$

- To solve the above equation we apply Ito's Lemma :
  $dG = (\frac{\partial G}{\partial t} + \frac{\partial G}{\partial S}\mu S + \frac{1}{2}\frac{\partial^2 G}{\partial S^2}\sigma^2 S^2)dt + \frac{\partial G}{\partial S}\sigma S dZ_t$ and by setting $G = ln(S_i)$ we obtain :
  $dG = (r - \frac{1}{2}\sigma^2)dt + \sigma dZ_t$ is a Generalized Wiener Process because now drift and variance rates are constants
- Next step is to integrate $\int$ on both sides for our time interval of $t \in [0, T]$ thus we get: $lnS_{i,T} = lnS_{i,0} + (r - \frac{1}{2}\sigma^2)T + \sigma Z_{i,T}$ or equivallent:
  $S_{i,T} = S_{i,0} \;\; exp((r - \frac{1}{2}\sigma_i^2)T + \sigma_i Z_{i,T}$ with $Z_i = \epsilon_i \sqrt{\Delta t} \;\; with \;\; \epsilon_i \sim N(0, 1)$
  So there is the conclusion that: $lnS_{i,T} \sim N[lnS_{i,0} + (r - \frac{1}{2}\sigma_i^2)T, \sigma_i\sqrt{T}]$ meaning there is a Lognormal distribution followed by each asset's price

# Theoretical Pricing — cont'd

Since the existance of two assets simultaniously under the payoff's structure, with each one having its own mean and volatility, there is the need to define their prices in a **Joint Distribution** which is the key function to express possible dependencies overall.
That's because here : $E[f(S_1, S_2)] \neq f(E[S_1], E[S_2])$
First, lets assume that :

- A joint vector of the prices is $(lnS_1(T), lnS_2(T) = (Y_1, Y_2) \sim N(\mu, \overline{\Sigma})$, where now:
  - $\mu = \begin{pmatrix} lnS_1(0) + (r - 1/2\sigma_1^2)T \\ lnS_2(0) + (r - 1/2\sigma_2^2)T \end{pmatrix}$
  - $\Sigma = \begin{pmatrix} \sigma_1^2 T & 0 \\ 0 & \sigma_2^2 T \end{pmatrix}$ So since the $\Sigma$ is diagonal,then $Y_1$ and $Y_2$ are independent normal variables.

## Theoretical Pricing — Disclaimer

Since we have created a backbone for our theoretical pricing we must report here the following statement:

- By evaluating the Marginal Densities of the Prices per asset we are able to compute the exact closed formula of call-on-min payoff. Thus it is always better to directly calculate it from that formula rather than proceeding with the numerical approximations of Monte Carlo.
- In more detail the Closed-Form Formula was computed by Stulz and is: $C_{min}(0) = S_{1,0}e^{-q_1 T}M(d_1, -d; -\rho_1) + S_{2,0}e^{-q_2 T}M(d_2, d - \Sigma\sqrt{T}; -\rho_2) - Ke^{rT}M(d_1 - \sigma_1\sqrt{T}, d_2 - \sigma_2\sqrt{T}; \rho)$
  $M(a, b; \lambda)$ is the CDF of the Standard Bivariate Normal Distribution with some correlation $\lambda$.

# Monte Carlo

Apart from the Closed-Form Formula of Stulz,we mainly *estimated the fair value of a European Rainbow Option - Call on min* with the method of **Monte Carlo.**

- **Plain Monte Carlo comes along with a higher degree of computational complexity**.Therefore we cannot always take a large sample size $N$ and for that reason we **must find more efficient ways to reduce our errors(improve accuracy)**.
  $Error \sim \frac{\sqrt{Var}}{\sqrt{N}}$
- To achieve a better accuracy we implemented on our model two very known variance reduction techniques named : **Antithetic Variates and Control Variates**

# Monte Carlo-Plain Vanilla

Given the previous equations for the Asset's Pricing Paths we can set now the following :

- $S_{i,T} = S_{i,0}\ exp\{(r - \frac{1}{2}\sigma_i^2)T + \sigma_i\sqrt{T}\epsilon_i\}$ where $\epsilon_i \sim N(0,1)$ Independent Standard Normal Random Variables.
- For each simulation $j = 0 : N$ we first generated two independent variables $U_i, i = 1,2 \sim Unif(0,1)$
- Then take *Inverse CDF function of Normal distribution provided* we created the standard normal variables $\epsilon_{1,j}, \epsilon_{2,j} \sim N(0,1)$.
  Thus we can accumulate all the possible prices per $j = 0 : N$ simulations and extract the payoff as it follows :
- $Payoff_j = max(min(S_{1,j}, S_{2,j}) - K, 0.0)$

# Monte Carlo-Plain Vanilla — cont'd

After the simulation of the payoffs per asset $i = 1, 2$ we should discount them with the continuous compounded factor $e^{-rT}$ to obtain their present value:

- $PV = V_0 \approx e^{-rT} \frac{1}{N} \sum_{j=1}^{N} Payoff_{j,T}$

*The above estimator converges to the true option value as* $N \to \infty$ by the *Law of Large Numbers w.p = 1*.

Hence, the fair estimated value of the option is the average of all payoff simulations:

- $\hat{V}_{plain,MC} = e^{-rT} \ \frac{1}{N} \sum_{j=1}^{N} Payoff_j$

# Monte Carlo Plain Vanilla — C++ Code

```cpp
//Plain Monte Carlo of a Rainbow option call on the minimum
//Two assets with zero correlation (ρ=0)
double MC_Rainbow_Call_on_min(double S1, double S2, double K, double r, double v1, double v2, double T, int num_sims){
    //Asset price at maturity --> Si,j= So * exp[(r-1/2 * σi^2)T + σi * sqrt(T) * εi,j]
    //Drift terms under the risk-neutral measure:
    double nu_T1 = (r - 0.5 * v1 * v1) * T;
    double nu_T2 = (r - 0.5 * v2 * v2) * T;
    //Volatility*sqrt(T)
    double v_T1 = v1 * sqrt(T);
    double v_T2 = v2 * sqrt(T);
    double disc_payoff_sum  = 0;
    double disc_payoff_squared_sum = 0;

    for (int i = 0; i < num_sims; i++) {
        //Independent standard normal shocks (ρ = 0)
        double epsilon1 = Standard_Normal_Rand();
        double epsilon2 = Standard_Normal_Rand();
        //Terminal prices under Black-Scholes formula:
        double S_T1 = S1 * exp(nu_T1 + v_T1 * epsilon1);
        double S_T2 = S2 * exp(nu_T2 + v_T2 * epsilon2);
        //Payoff of a call on the minimum of the two assets
        double minn = min(S_T1, S_T2);
        double disc_payoff = max(minn - K, 0.0) * exp(-r*T);

        disc_payoff_sum += disc_payoff;
        disc_payoff_squared_sum += disc_payoff*disc_payoff;
    }
    //Monte Carlo price estimator
    double disc_payoff_average = disc_payoff_sum / num_sims; // PV = (Σ(Payoffj,T)/num_sims
    //Unbiased sample variance of the discounted payoff
    sample_var = (disc_payoff_squared_sum - num_sims * disc_payoff_average * disc_payoff_average) / (num_sims - 1.0);
    return disc_payoff_average;
}
```

Figure: Plain(Crude) Monte Carlo - Main Function

# Monte Carlo Plain Vanilla — C++ Results

MC_Rainbow_Call_on_min.cpp

```cpp

int main() {
    // First we create the parameter list
    int num_sims = 4000000;   // Number of simulated asset paths
    double S1 = 100.0;        // Stock1 price
    double S2 = 100.0;        // Stock2 price
    double K = 100.0;         // Strike price
    double r = 0.05;          // Risk-free rate (5%)
    double v1 = 0.2;          // Volatility of the underlying (20%)
    double v2 = 0.2;          // Volatility of the underlying (20%)
    double T = 1.0;           // One year until expiration
    double p = 0.0;           //correlation p between S1 AND S2

    // Then we calculate the call value via Monte Carlo
    double call_on_min = MC_Rainbow_Call_on_min(S1, S2, K, r, v1, v2, T, num_sims);

    // Finally we output the parameters and prices

    // Relative error for alpha = 5% (95% confidence)
    double z = inverse_of_normal_cdf(0.975, 0.0, 1.0);
    double rel_error = z * sqrt(sample_var / num_sims) / call_on_min;

    cout << "--- MONTE CARLO METHOD ---"        << "\n" << "\n";
    cout << "Number of Paths: "                 << num_sims << std::endl;
    cout << "Underlying 1: "                    << S1 << "\n";
    cout << "Underlying 2: "                    << S2 << "\n";
    cout << "Strike: "                          << K << "\n";
    cout << "Risk-Free Rate: "                  << r << "\n";
    cout << "Volatility1: "                     << v1 << "\n";
    cout << "Volatility2: "                     << v2 << "\n";
    cout << "Maturity: "                        << T << "\n";
    cout << "Correlation p between S1 AND S2: " << p << "\n";
    cout << "Call Price: "                      << call_on_min << "\n";
    cout << "Relative Error: "                  << rel_error << "\n";

    return 0;
}
```

C:\Users\billn\Desktop\MC_R

```
--- MONTE CARLO METHOD ---

Number of Paths: 4000000
Underlying 1: 100
Underlying 2: 100
Strike: 100
Risk-Free Rate: 0.05
Volatility1: 0.2
Volatility2: 0.2
Maturity: 1
Correlation p between S1 AND S2: 0
Call Price: 3.29734
Relative Error: 0.00208138

--------------------------------
Process exited after 2.899 seconds with return value 0
Press any key to continue . . .
```

Figure: Crude Monte Carlo - Results

# Monte Carlo-Antithetic Variates

Now in order to improve the efficiency and the accuracy of the Crude Monte Carlo, we introduce a variance reduction method named *Antithetic Variates*.

- This method main core is to simulate the asset paths for *both* $\epsilon_{1,j}, \epsilon_{2,j} \sim N(0,1)$ alongside their *antithetic* -opposite sign- $-\epsilon_{1,j}, -\epsilon_{2,j}$.**With this procedure we manage to generate negatively correlated variables**.

*This way we achieve the reduction of the variance of our estimator by creating symmetry, hence balancing possible high and low outcomes that would introduce noise in our results.*

# Monte Carlo-Antithetic Variates — cont'd

The equations used in the c++ code are provided here:

- $S^{+}_{i,j}(T) = S_0\ exp[(r - \frac{1}{2}\sigma_i^2)T + \sigma_i\sqrt{T}\epsilon_j]$
- $S^{-}_{i,j}(T) = S_0\ exp[(r - \frac{1}{2}\sigma_i^2)T + \sigma_i\sqrt{T}\epsilon_j]$ where the $+$ symbol is used for the possitive $\epsilon_j$ while then $-$ symbol represents the antithetic $\epsilon_j$ So we can reset out payoffs in the following manor :
  - $Payoff_j^{+} = max(min(S^{+}_{1,j}, S^{+}_{2,j}) - K, 0.0)$
  - $Payoff_j^{-} = max(min(S^{-}_{1,j}, S^{-}_{2,j}) - K, 0.0)$

  Then we average those for each antithetic pair:
  - $\hat{V}_j = \frac{1}{2}(Payoff_j^{+} + Payoff_j^{-})$ and finally the Antithetic Monter Carlo estimate for the option value is:
  - $\hat{V}_{AV,MC} = e^{-rT}\frac{1}{N}\sum_{j=1}^{N}\hat{V}_j$ with the condition : $Var(V_{A\hat{V},MC}) < Var(V_{\hat{plain}})$

# Monte Carlo-Antithetic Variates — C++ Code

Agianoglou AV MC.cpp

```cpp
// Pricing a European Call Option with the Monte Carlo method
double MC_Call_Price_Rainbow(double S1,double S2,double K,double r,double v1,double v2,double T,double p,int num_sims)
 {
    double nu_T1=(r-0.5*v1*v1)*T;
    double nu_T2=(r-0.5*v2*v2)*T;
    double v_T1=v1*sqrt(T);
    double v_T2=v2*sqrt(T);

    double anti_payoff_sum=0.0;
    double anti_payoff_squared_sum=0.0;

    for (int i=0; i<num_sims; i++){
        double epsilon1=Standard_Normal_Rand();
        double epsilon2=Standard_Normal_Rand();

        double S_T1=S1*exp(nu_T1+v_T1*epsilon1);
        double S_T2=S2*exp(nu_T2+v_T2*epsilon2);
        double disc_payoff=std::max(std::min(S_T1,S_T2)-K,0.0);

        //Monte Carlo with Antithetic Variates
        double S_T1_a=S1*exp(nu_T1+v_T1*(-epsilon1));
        double S_T2_a=S2*exp(nu_T2+v_T2*(-epsilon2));
        double anti_payoff=std::max(std::min(S_T1_a,S_T2_a)-K,0.0);
        double anti_payoff_squared=anti_payoff*anti_payoff;
        double anti_mean=(disc_payoff+anti_payoff)*0.5;
        anti_payoff_sum+=anti_mean;
        anti_payoff_squared_sum+=(anti_mean*anti_mean);
    }

double anti_payoff_average=anti_payoff_sum/num_sims;
double price_anti=(anti_payoff_average)*std::exp(-r*T);
anti_sample_var=((anti_payoff_squared_sum-num_sims*anti_payoff_average*anti_payoff_average)/(num_sims-1))*std::exp(-2.0*r*T);

return price_anti;
}
```

Figure: Monte Carlo Antithetic Variates - Main Function

# Monte Carlo-Antithetic Variates — C++ Results

[*] Agianoglou AV MC.cpp

```cpp
int main() {
    // First we create the parameter list
    int num_sims = 4000000;   // Number of simulated asset paths
    double S1 = 100.0;      // Stock1 price
    double S2 = 100.0;      // Stock2 price
    double K = 100.0;      // Strike price
    double r = 0.05;       // Risk-free rate (5%)
    double v1 = 0.2;        // Volatility of the underlying (20%)
    double v2 = 0.2;        // Volatility of the underlying (20%)
    double T = 1.0;        // One year until expiration
    double p = 0.0;       //correlation p between S1 AND S2

    // Then we calculate the call value via Monte Carlo
    double call = MC_Call_Price_Rainbow(S1, S2, K, r, v1, v2, T, p, num_sims);

    // Finally we output the parameters and prices
    // Relative error for alpha = 5% (95% confidence)
    double z = inverse_of_normal_cdf(0.975, 0.0, 1.0);
    double rel_error = z * std::sqrt(anti_sample_var / num_sims) / call;

    cout << "--- ANTITHETIC VARIATES METHOD ---" << endl;
    cout << "Number of Paths: " << num_sims << std::endl;
    cout << "Underlying 1: " << S1 << "\n";
    cout << "Underlying 2: " << S2 << "\n";
    cout << "Strike: " << K << "\n";
    cout << "Risk-Free Rate: " << r << "\n";
    cout << "Volatility1: " << v1 << "\n";
    cout << "Volatility2: " << v2 << "\n";
    cout << "Maturity: " << T << "\n";
    cout << "Correlation p between S1 AND S2: " << p << "\n";
    cout << "Call Price: " << call << "\n";
    cout << "Relative Error: " << rel_error << "\n";
    return 0;
}
```

C:\Users\nikos\Downloads\Un

```
--- ANTITHETIC VARIATES METHOD ---
Number of Paths: 4000000
Underlying 1: 100
Underlying 2: 100
Strike: 100
Risk-Free Rate: 0.05
Volatility1: 0.2
Volatility2: 0.2
Maturity: 1
Correlation p between S1 AND S2: 0
Call Price: 3.29706
Relative Error: 0.00129898

--------------------------------
Process exited after 1.608 seconds with return value 0
Press any key to continue . . .
```

Figure: Monte Carlo Antithetic Variates - Results

# Monte Carlo-Antithetic Variates — cont'd

The importance of the Antithetic Variates depends in a high degree from the nature of the payoff's function.Highly (increasing )monotonic payoffs,such as rainbow options on minimum that we examine, are having strong negative correlation between the paired outcomes.This statement does not apply though for payoffs that dont exhibit this feauture.Therefore the structure of each payoff is crucial when implementing the Antithetic Variates technique in Monte Carlo approximations.

# Monte Carlo-Control Variates

*Another way to improve accuracy and efficiency of Monte Carlo simulation is to implement the Control Variates method.*

- Here the variance is again reducted by trying to introduce information to our system,from *related random variable/Control Variate*.In our case we have two new variables with the following feature :

- $E[Y_1] = \nu_1 : \ known$
- $E[Y_2] = \nu_2 : \ known$

Now we set up the controlled estimator :$X_c = X + c_1\ (Y_1 - \nu_1) + c_2(Y_2 - \nu_2)$ where the $E[X_c] = \mu$ is known.So we just have to *select the proper(optimal)* $c_i^*$.

# Monte Carlo-Control Variates —cont'd

In order to obtain those $c_i^*$ we need to **minimize** the **Variance of the Control Variate** which is :

- $Var(X_c) = Var(X + c_1\ (Y_1 - \nu_1) + c_2(Y_2 - \nu_2)$

So now we need to find the repsectivly $c_{1,2}^*$ such that : $dVar(X_c)/dc_{1,2} = 0$ and then $d^2 Var(X_c)/dc_{1,2}^2 > 0$.
After some algebraic operations we obtain that the optimal coefficients $c_i^*$ , $i = 1, 2$ are:

- $c_1^* = -\frac{Cov(X,Y_1)}{Var(Y_1)}$
- $c_2^* = -\frac{Cov(X,Y_2)}{Var(Y_2)}$

# Monte Carlo-Control Variates —cont'd

So now we can derive that from the basic control variate estimator form:

- $\hat{V}_{control} = \frac{1}{N}\sum_{j=1}^{N}[X^{(j)} + c_1(Y_1^{(j)} - \nu_1) + c_2(Y_2^{(j)} - \nu_2)]$
  can be transformed into the *optimal variance-minimizing control variate estimator*:
- $\hat{V}_{control} = \frac{1}{N}\sum_{j=1}^{N}[X^{(j)} - \frac{Cov(X,Y_1)}{Var(Y_1)}(Y_1^{(j)} - \nu_1) - \frac{Cov(X,Y_2)}{Var(Y_2)}(Y_2^{(j)} - \nu_2)]$

In practice,these covariances and variances are computed from a pilot simulation.

# Monte Carlo-Control Variates —cont'd

Therefore the main advantage for implying the Control Variates technique originates from the state in which the exact analytical expectation of the underlying asset is known.
Having that in mind it is straightforward that **the best control variates are the individual asset prices :** $S_1(T), S_2(T)$ since their risk-neutral expectations are known analytically. So on the C++ code we setted :

- $Y_1 = S_1$
- $Y_2 = S_2$

Finally, the Control Variates method allows the user to fully adjust the estimator in a way that captures and removes parts of the variance,which occur due to any possible fluctuations between the exotic rainbow payoff and the simpler,analytical traced control variables.

# Monte Carlo - Control Variates — C++ Code - Slide 1

```cpp
double sample_var = 0.0;
// Constructing the Monte Carlo Pricing Function
// for Rainbow Option

double MC_Call_Price_Rainbow_CV(double S1,double S2,double K,double r,double v1,double v2,double T,int num_sims,int num_pilot){

    // The correlation is assumed to be ρ = 0
    // Setting up the variables of the Si(T) = Si(0)* exp( (r-0.5* v_i^2)T + v_i*sqrt(T)*ε_i)

    // Each asset here must have its own drift and volatility under the risk-neutral measure
    double nu_1 = ( r - 0.5 * v1 * v1) * T;
    double nu_2 = ( r - 0.5 * v2 * v2) * T;

    double v_1 = v1*sqrt(T);
    double v_2 = v2*sqrt(T);

    // For the Pilot Simulations we need the : Sum of Y1_samples where Y1 := S1_T
    // Same for Y2_samples , where Y2 := S2_T as control variate
    double S1_T_sum = 0.0;
    double S2_T_sum = 0.0;

    // Sum of X*Y1 from sampling in order to later estimate the Cov(X,Y1)
    double product1_sum = 0.0;
    // Same for the Cov(X,Y2)
    double product2_sum = 0.0;

    double disc_payoff_sum = 0.0;

    // Main loop for the path prices generation
    int i,j ;
    for( i=0; i<num_pilot; i++){

        // Generating each ε_i ~ N(0,1) for asset i
        // Here there are only 2 needed.

        double epsilon1 = Stand_Normal_Rand();
        double epsilon2 = Stand_Normal_Rand();

        double S1_T = S1 * exp(nu_1 + v_1 * epsilon1);
```

Figure: Control Variates - Main Function(1)

# Monte Carlo - Control Variates — C++ Code - Slide 2

```cpp
        double S1_T = S1 * exp(nu_1 + v_1 * epsilon1);
        double S2_T = S2 * exp(nu_2 + v_2 * epsilon2);

        //Setting up the payoff of the 2 assets Rainbow Price
        double minimum1 = min(S1_T,S2_T);

        double disc_payoff = max(minimum1 - K,0.0) * exp( - r * T) ;

        // In general the formula of : Cov(X,Yi) = E[XY_k,i] - E[X]E[Y_k.i] with k=1,2 and i : 0 up to num_pilot
        // So the 1st term is correspondingly per asset the :
        double product1 = S1_T * disc_payoff ;
        double product2 = S2_T * disc_payoff;
        // Accumulate ΣY1_i and ΣY2_i
        S1_T_sum += S1_T;
        S2_T_sum += S2_T;
        // Accumulate Σ(X_i)
        disc_payoff_sum += disc_payoff ;
        // Accumulate Σ(X_i,Y1_i)
        product1_sum += product1;
        // Accumulate Σ(X_i,Y1_i)
        product2_sum += product2;

    }
      // sample covariance estimator for Cov(X,Y) from pilot:
    // Cov(X,Y) ≈ [ Σ(XY) - (ΣX)(ΣY)/N ] / (N-1)  (PDF gives sample covariance formula for pilot)
    double sample1_cov = (product1_sum - S1_T_sum * disc_payoff_sum / num_pilot) / (num_pilot - 1);
    double sample2_cov = (product2_sum - S2_T_sum * disc_payoff_sum / num_pilot) / (num_pilot - 1);

    // For this application the control variate is Yi = Si_T.
    // The PDF states E[S_T] = S0 e^{rT} and Var(S_T) = S0^2 e^{2rT}(e^{σ^2 T} - 1).
    double VarY1 = S1 * S1 * exp(2.0 * r * T) * (exp(v1 * v1 * T) - 1.0); // Var(Yi) known in closed form
    double VarY2 = S2 * S2 * exp(2.0 * r * T) * (exp(v2 * v2 * T) - 1.0); // Var(Y) known in closed form


    double ExpY_1 = S1 * exp( r * T); // E[Y1] is known
    double ExpY_2 = S2 * exp( r * T); // E[Y2] is known

    // Now we need to update the c* per asset where we obtained from minimizing Var(X_c)
    // X_c = X + c1 * ( Y1 - E[Y1]) + c2 * ( Y2 - E[Y2])
    // So : Var(X_c) = Var(X) + 2 * Σ ( c_i * Cov(X,Y_i) + ΣΣ c_i * c_j * Cov(Yi,Yj)
```

Figure: Control Variates - Main Function(2)

# Monte Carlo - Control Variates — C++ Code Slide 3

```cpp
    // Now we need to update the c* per asset where we obtained from minimizing Var(X_c)
    // X_c = X + c1 * ( Y1 - E[Y1]) + c2 * ( Y2 - E[Y2])
    // So : Var(X_c) = Var(X) + 2 * Σ ( c_i * Cov(X,Y_i) + ΣΣ c_i * c_j * Cov(Yi,Yj)

    // Setting up the c1* and c2*
    double c1 = -sample1_cov/VarY1;
    double c2 = -sample2_cov/VarY2;

    double control_var_sum = 0.0;
    double control_var_squared_sum = 0.0;

    // Now we proceed to main looping interations:
    for(int i=0; i<num_sims; i++){

        double epsilon1 = Stand_Normal_Rand();
        double epsilon2 = Stand_Normal_Rand();

        //Create the new S1_T,S2_T:
        double new_S1_T = S1 * exp( nu_1 + v_1*epsilon1 );
        double new_S2_T = S2 * exp( nu_2 + v_2*epsilon2 );

        double new_minimum = min( new_S1_T, new_S2_T);
        double new_disc_payoff = exp( -r * T) * max( new_minimum - K, 0.0);

        double control_var = new_disc_payoff + c1 * (new_S1_T - ExpY_1) + c2 * (new_S2_T - ExpY_2);

        control_var_sum += control_var;
        control_var_squared_sum += control_var * control_var;
    }

    double control_var_average = control_var_sum / num_sims;
    sample_var = (control_var_squared_sum - num_sims*control_var_average*control_var_average) / (num_sims - 1);

    return control_var_average;
}

int main() {
```

Figure: Control Variates - Main Function(3)

# Monte Carlo - Control Variates — C++ Results

```cpp
int main() {
    //unsigned int seed = 123456u; // pick any fixed integer seed
    //srand(seed);
    // Getting the actual parameters :

    int num_sims = 4000000;   // Number of simulated asset paths
    int num_pilot = 10000;
    double S1 = 100.0;        // Stock1 price
    double S2 = 100.0;        // Stock2 price
    double K = 100.0;        // Strike price
    double r = 0.05;         // Risk-free rate (5%)
    double v1 = 0.2;          // Volatility of the underlying (20%)
    double v2 = 0.2;          // Volatility of the underlying (20%)
    double T = 1.0;          // One year until expiration

    // Via the function we compute the Call price of Rainbow Option

    double call_price = MC_Call_Price_Rainbow_CV(S1,S2,K,r,v1,v2,T,num_sims,num_pilot);

    // Relative error for alpha = 5% (95% confidence)
    double z = inverse_of_normal_cdf(0.975, 0.0, 1.0);
    double rel_error = z * std::sqrt(sample_var / num_sims) / call_price;

    // Print out the results
    cout << "Number of Paths for Control Variates: " << num_sims << endl;
    cout << "Number of Pilot sims:" << num_pilot << endl;
    cout << "Underlying 1: " << S1 << "\n";
    cout << "Underlying 2: " << S2 << "\n";
    cout << "Strike: " << K << "\n";
    cout << "Risk-Free Rate: " << r << "\n";
    cout << "Volatility1: " << v1 << "\n";
    cout << "Volatility2: " << v2 << "\n";
    cout << "Maturity: " << T << "\n";
    cout << "Correlation p between S1 AND S2: 0"<< "\n";

    cout << "Call Price: " << call_price << "\n";
    cout << "Relative Error: " << rel_error << "\n";

    return 0;
}
```

```
Number of Paths for Control Variates: 4000000
Number of Pilot sims:300
Underlying 1: 100
Underlying 2: 100
Strike: 100
Risk-Free Rate: 0.05
Volatility1: 0.2
Volatility2: 0.2
Maturity: 1
Correlation p between S1 AND S2: 0
Call Price: 3.2956
Relative Error: 0.00153039

--------------------------------
Process exited after 1.547 seconds with return value 0
Press any key to continue . . .
```

Figure: Control Variates - Results

# Summary of Numerical Results (Rainbow Call on min)

| Method | $\hat{V}$ (Price) | Rel. Error (95%) | Runtime (s) |
|---|---|---|---|
| Crude MC | 3.297 300 | 0.002 081 | 3.150 000 |
| Antithetic Variates (AV) | 3.297 000 | 0.001 299 | 2.400 000 |
| Control Variates (CV) | 3.295 600 | 0.001 530 | 1.550 000 |

| Method | Notes / Interpretation |
|---|---|
| Crude MC | Baseline Monte Carlo; largest uncertainty for the same $N$. |
| Antithetic Variates (AV) | Best variance cancellation here due to payoff monotonicity (paired negatives). |
| Control Variates (CV) | Strong variance reduction and fastest runtime; min-payoff limits correlation and pilot-estimation adds some noise. |

**Consistency check:** all estimates within $\mathcal{O}(10^{-3})$ of each other $\Rightarrow$ no evidence of bias; differences consistent with MC sampling error.

Initial parameters: $N = 4 \times 10^6$, $S_1 = S_2 = 100$, $K = 100$, $r = 0.05$, $\sigma_1 = \sigma_2 = 0.2$, $T = 1$, $\rho = 0$.