Course Summary of Computational Methods of Financial Mathematics

by
Jessica Lee Copp

A Thesis
Submitted to the Faculty
of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Financial Mathematics

May 2009
ADVISOR: Marcel Blais

Table of Contents

Introduction

Most realistic financial derivatives models are too complex to allow explicit analytic solutions. The computational techniques used to implement those models fall into two broad categories: finite difference methods for the solution of partial differential equations (PDEs) and Monte Carlo simulation. Accordingly, the course consists of two sections.

The first half of the course focuses on finite difference methods. The following topics are discussed; Parabolic PDEs, Black-Scholes PDE for European and American options; binomial and trinomial trees; explicit, implicit and Crank- Nicholson finite difference methods; far boundary conditions, convergence, stability, variance bias; early exercise and free boundary conditions; parabolic PDEs arising from fixed income derivatives; implied trees for exotic derivatives, adapted trees for interest rate derivatives.

The second half of the course focuses on Monte Carlo. The following topics are discussed; Random number generation and testing; evaluation of expected payoff by Monte Carlo simulation; variance reduction techniquesantithetic variables, importance sampling, martingale control variables; stratification, low-discrepancy sequences and quasi-Monte Carlo methods; efficient evaluation of sensitivity measures; methods suitable for multifactor and term-structure dependent models.

Computational Methods of Financial Mathematics is taught by Marcel Blais, a professor at Worcester Polytechnic Institute.

Background

Options

European Option:
A Euroean Option is a financial contract between two parties, the holder and the writer. At a fixed expiry time T, the holder receives a payoff, and no payoff occurs before time T. Further, the contract is uniquely described by its payoff fuction.

European Call Option:
A Euroean Call Option is a financial contract with the following conditions:

1. at a prescribed time in the future, the expiry date, the holder of the option may purchase a prescribed asset, known as the underlying asset, for a prescribed amount, called the strike price
2. the holder has a right not an obligation
3. seller potentially has an obligation
4. has value

European Put Option:
A Euroean Put Option is a financial contract with the same conditions as a European call, except the holder has the right to sell the underlying to the writer a expiry for the strike price.

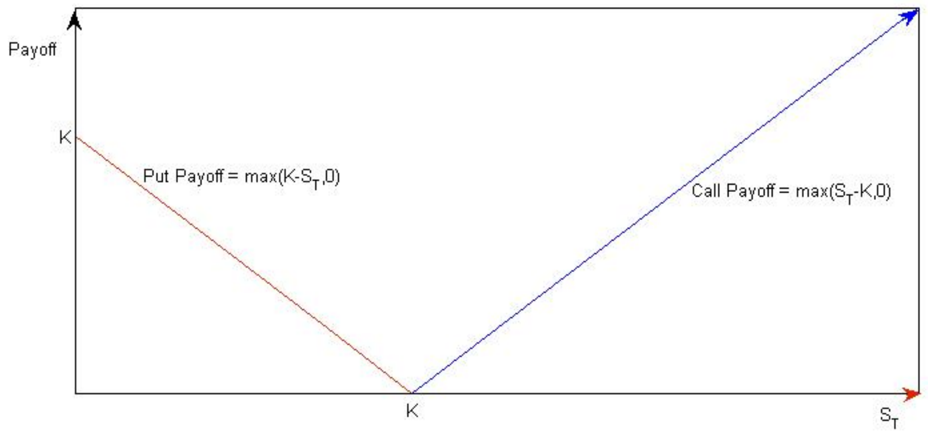

Figure 1: European Options

American Option:
An American Option is a financial contract between two parties, the holder and the writer, with expirty time T. At any time t, $0 < t \leq T$, the holder may exercise the option and receive payoff $g(t, S_t)$, where S_t is the time-t value of the underlying.

Path Dependent Options:
The option payoff can depend on the history of the underlying price path. An example of a path dependent option in and Asian call option. It's defined by its payoff as follows:

$$Payoff = max(A_t - K, 0) \tag{1}$$

$$A_t = avg(S_t : 0 \leq t \leq T) = \frac{1}{T}\int_0^T S_t dt \tag{2}$$

Stochastic Processes and Brownian Motion

Stochastic Process:
A stochasitic process X is a colletion of random variables such that:

$$(X_t, t \in [0, \tau]) = (X_t(\omega), t \in [0, \tau], \omega \in \Omega) \tag{3}$$

where Ω is our sample space and $(\Omega, \mathcal{F}, P)$ is our probability space.

Brownian Motion:
A stochastic process

$$W = (W_t, t \in [0, \infty]) \tag{4}$$

is called a standard Brownian Motion if the following are satisfied:

1. $W_0 = 0$
2. for $0 \leq s \leq t, W_t - W_s \sim N(0, t - s)$
3. independence of increments for $0 \leq s \leq t < u \leq v$,
   $(W_t - W_s) \perp (W_u - W_v)$
4. W has continuous sample paths

A Brownian Motion is nowhere differentiable with probability one.

Examples of Brownian Motion:

- Brownian motion with drift: $(E[W_t] = 0)$

$$X_t = \mu t + \sigma W_t \tag{5}$$

$$E[X_t] = E[\mu t + \sigma W_t] = \mu t + \sigma E[W_t] = \mu t \tag{6}$$

- Geometric Brownian Motion:

$$X_t = \exp(\mu t + \sigma W_t) \tag{7}$$

Stochastic Differential Equations:
Consider a small time interval, dt, during which an asset price changes from S to S+dS.
We decompose it into two parts:

1. One part comes from a fixed rate of return over dt, written μdt, where μ is called the drift.

2. The random component is given by a random sample drawn from a normal distribution with mean 0 and variance dt.

This gives us the following equation:

$$\frac{dS}{S} = \mu dt + \sigma dW_t \tag{8}$$

Ito's Formula

Ito Process:
X_t is an Ito Process if it is a stochastic process that can be written

$$dX_t = u(t)dt + v(t)dW_t \tag{9}$$

Ito's Formula:
Suppose X_t is an Ito process and $g(t,x) \in C^2([0,\infty) \times \mathbb{R})$.
Then $Y_t = g(t, X_t)$ is an Ito process and

$$dY_t = g_t(t, X_t)dt + g_x(t, X_t)dX_t + \frac{1}{2}g_{xx}(t, X_t)dX_t dX_t \tag{10}$$

where $dX_t dX_t$ is computed using the following:

- $dt \cdot dt = 0$
- $dt \cdot dW_t = 0$
- $dW_t \cdot dW_t = dt$

Black-Scholes-Merton Partial Differential Equation

Deriving the Black-Scholes-Merton Partial Differential Equation:
Form a portfolio with one option and $-\Delta$ units of the underlying. The time t value of our portfolio is $\pi_t = V_t - \Delta S_t$
It's value changes according to $d\pi_t = dV_t - \Delta dS_t$
Ito's formula can be applied to obtain the Black-Scholes-Merton PDE:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2} + r\frac{\partial V}{\partial S}S_t - rV = 0 \tag{11}$$

This works backward in time becasuse an end condition is known instead of an initial condition when looking at the boundary value problem. Using the transfomation $t = T - t$ will allow moving forward in time.
We can rewrite the Black-Scholes-Merton PDE as:

$$\frac{\partial V}{\partial t} = r\frac{\partial V}{\partial S}S_t + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2} - rV \tag{12}$$

Option Pricing Approaches

There are four option pricing approaches:

1. Fundamental Theorem of Asset Pricing
    - use of probability
    - price options by taking the discounted expected payoff under the risk-neutral measure
    - can get analytic solutions in some cases
2. Option Replication
    - create a synthetic option
    - use positions in different financial instruments that collectively replicate the option value
3. Solving Partial Differential Equations
    - often an option's value can be determined by solving a boundary value problem, which is a partial differential equation and a set of boundary conditions
    - sometimes can find analytic solutions
    - usually approximate the solution numerically using finite difference methods
    - deal with discretizing the continuous models
4. Carlo Methods
    - approximate an option's value by simulation
    - repetition (look at how things are converging)

This paper is going to focus on solving partial differential equations using the finite difference method and Monte Carlo methods.

Finite Difference Methods

Estimation Using Taylor's Theorem

To estimate $\frac{\partial u}{\partial x}$ and $\frac{\partial^2 u}{\partial x^2}$ Taylor's Theorem can be applied to $u(x+h)$ and $u(x-h)$.

- First look at $u(x+h) - u(x-h)$
  Expand to get:

$$u(x+h)-u(x-h) = \begin{array}{c} [u(x) + u'(x)h + \frac{u''(x)h^2}{2} + \frac{u'''(x)h^3}{3!} + ...] \\ -[u(x) - u'(x)h + \frac{u''(x)h^2}{2} - \frac{u'''(x)h^3}{3!} + ...] \end{array} \tag{13}$$

  Rearrange and solve in terms of x and t to get:

$$\frac{\partial u}{\partial x}(x,t) = \frac{u(x+\Delta x, t) - u(x-\Delta x, t)}{2\Delta x} + \mathcal{O}(\Delta x^2) \tag{14}$$

- Now look at $u(x+h) + u(x-h)$
  Expand to get:

$$u(x+h) + u(x-h) = 2u(x) + 2\frac{u''(x)h^2}{2} + 2\frac{u^{(4)}(x)h^4}{4!} + \mathcal{O}(h^6) \tag{15}$$

  Rearrange and solve in terms of x and t to get:

$$\frac{\partial^2 u}{\partial x^2}(x,t) = \frac{u(x+\Delta x, t) - 2u(x) + u(x-\Delta x, t)}{(\Delta x)^2} + \mathcal{O}(\Delta x^2) \tag{16}$$

Note: $\mathcal{O}$ is the order of the approximation for the error term.

Spatial Discretization

Let $u_i = u(x_i)$, then the above equations are of the form:

$$\frac{\partial u}{\partial x}(x_i, t) = \frac{u_{i+1} - ui - 1}{2\Delta x} - \mathcal{O}(\Delta x^2) \tag{17}$$

$$\frac{\partial^2 u}{\partial x^2}(x_i, t) = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} - \mathcal{O}(\Delta x^2) \tag{18}$$

These equations hold for the interior points. To solve for the boundary conditions, linearly extrapolate the values for u_0 and u_I using the two adjacent points. This gives the following two equations.

$$u_0 = 2u_1 - u_2 \tag{19}$$

$$u_I = 2u_{I-1} - u_{I-2} \tag{20}$$

Applying these spatial discretizations we get the following:

$$\frac{\partial u}{\partial t} = r(\frac{u_{i+1} - u_{i-1}}{2\Delta x}) + \frac{1}{2}\sigma^2(\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}) - ru_i + \epsilon \tag{21}$$

Simplify to:

$$\frac{\partial u}{\partial t} = \beta u_{i-1} - \gamma u_i + \alpha u_{i+1} \tag{22}$$

Where:

- $\beta = \frac{-r}{2\Delta x} + \frac{\sigma^2}{2(\Delta x)^2}$
- $\gamma = \frac{\sigma^2}{(\Delta x)^2} + r$
- $\alpha = \frac{r}{2\Delta x} + \frac{\sigma^2}{(\Delta x)^2}$

This can be vectorized to give the following (after dropping the error term):

$$\frac{d\bar{u}}{dt} = A\bar{u} \tag{23}$$

Where: A=$\begin{bmatrix} 1 & -2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ \beta & -\gamma & \alpha & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & \beta & -\gamma & \alpha & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & \beta & -\gamma & \alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & -2 & 1 \end{bmatrix}$

- A is non-singular
- A has rank A
- A has eigenvectors $\bar{x}_m$, m = 1,2,...,m
- A has eigenvalues $\lambda_1, \lambda_2, ...\lambda_m$
- $A\bar{x}_m = \lambda_m \bar{x}_m$

Define the following:

- $X = [\bar{x}_1|\bar{x}_2|...|\bar{x}_m]$
- λ is a diagonal matrix of the eigenvalues of A
- $AX = [A\bar{x}_1|A\bar{x}_2|...|A\bar{x}_m] = [\lambda_1\bar{x}_1|\lambda_2\bar{x}_2|...|\lambda_m\bar{x}_m] = \lambda X$
- $\lambda = X^{-1}AX$
- $\bar{v} = X^{-1}\bar{u}$

This transforms the spatial finite difference scheme into:

$$\frac{d\bar{v}}{dt} = \lambda\bar{v} \tag{24}$$

Time Discretization

Shift Operator:
Let $v^n = v(n\Delta t)$. The shift operator E^i is defined by:

$$E^i v^n = v^{n+i} = v([n+i]\Delta t) \tag{25}$$

Time discretized finite differences can be expressed as polynomials in the shift operator. The polynomial shift operator is denoted by P(E).

Define the homogeneous difference equation by:

$$P(E)v^n = 0 \tag{26}$$

Consider $\frac{d\bar{v}}{dt} = \lambda \bar{v}$ Looking at the j^{th} component, obtain the finite difference expression:

$$\frac{dv_j}{dt}(t_n) = \frac{dv_j}{dt}(n\Delta t) \cong \frac{1}{\Delta t} \sum_{i=0}^{M} c_i v_j^{n+i} \tag{27}$$

for some constants c_i

This gives the solution:

$$v_j^n = \sum_{k=0}^{K} c_{jk} (\Lambda_{jk})^n \tag{28}$$

where the Λ_{jk} are the solutions of $P(\Lambda) = 0$
The set Λ_{jk} are called the amplification errors.

Method Evaluations

1. Consistency: A numerical scheme is consistent if the finite difference scheme converges to the partial differential equation as the space and time steps converge to zero.

2. Stability: A numerical scheme is stable if the difference between the numerical solution and the exact solution remains bounded as the number of steps goes to infinity. If any one $|\Lambda_{jk}| > 1$, the scheme is unstable.

3. Converenge: A numerical scheme converges if the difference between the numerical solution and the exact solution at a fixed point in the domain tends to zero uniformly as the space and time steps tend to zero.

4. Lax Equivalence Theorem: Given a properly posed linear initial value problem and a consistent finite difference scheme, then stability is the only requirement for convergence.

Specific Finite Difference Methods

1. The Explicit Euler Scheme:

   - begin with the spatial discretization $\frac{dv}{dt} = \lambda v$
   - use the explicit approximation $\frac{dv}{dt}|^n \cong \frac{v^{n+1}-v^n}{\Delta t}$
   - combine to get $v^{n+1} = \lambda \Delta t v^n + v^n$
   - rewrite as $P(E)v^n = 0$
     $v^{n+1} - v^n - \lambda \Delta t v^n = 0$
     $Ev^n - v^n \lambda \Delta t v^n = 0$
     $(E - 1 - \lambda \Delta t)v^n = 0$
     $P(E)v^n = 0$
   - solve $P(E) = 0$ to find the amplification error
     $\Lambda - 1 - \lambda \Delta t = 0$
     $\Lambda = 1 + \lambda \Delta t$
   - compare Λ to the expansion of $e^{\lambda \Delta t}$ to find that this method is first order accurate
   - to find stability conditions, look at $|\Lambda|$, which shows if $\lambda > 0$ then it's unstable and if $\lambda < 0$ then $\Delta t \leq \frac{-2}{\lambda}$ which shows restricted stability

2. The Implicit Euler Scheme:

   - begin with the spatial discretization $\frac{dv}{dt} = \lambda v$
   - use the explicit approximation $\frac{dv}{dt}|^{n+1} \cong \frac{v^{n+1}-v^n}{\Delta t}$
   - combine to get $v^n = (1 + \lambda \Delta t)v^{n+1}$
   - rewrite as $P(E)v^n = 0$
     $(1 + \lambda \Delta t)v^{n+1} - v^n = 0$
     $(1 + \lambda \Delta t)Ev^n - v^n = 0$
     $[(1 + \lambda \Delta t)E - 1]v^n = 0$
     $P(E)v^n = 0$
   - solve $P(E) = 0$ to find the amplification error
     $(1 + \lambda \Delta t)\Lambda - 1 = 0$
     $\Lambda = \frac{1}{1-\lambda \Delta t}$

- compare Λ (expressed as a geometric series) to the expansion of $e^{\lambda \Delta t}$ to find that this method is first order accurate
- to find stability conditions, look at $|\Lambda|$, which shows if $\lambda \leq 0$ then it's stable

3. Crank - Nicolson Scheme:

   - begin with the spatial discretization $\frac{dv}{dt} = \lambda v$
   - incoroporates both implicit and explicit features by taking the average of the implicit and explicit Euler schemes
   - use the approximation $\frac{1}{2}[\frac{dv}{dt}|^n + \frac{dv}{dt}|^{n+1}] = \frac{v^{n+1}-v^n}{\Delta t}$
   - combine to get $\frac{1}{2}\Delta t(\lambda v^n + \lambda v^{n+1}) = v^{n+1} - v^n$
     $v^{n+1} = v^n + \frac{1}{2}\lambda \Delta t(v^{n+1} - v^n)$
   - rewrite as $P(E)v^n = 0$
     $v^{n+1}(1 - \frac{1}{2}\lambda\Delta t) - v^n(1 + \frac{1}{2}\lambda\Delta t) = 0$
     $Ev^n(1 - \frac{1}{2}\lambda\Delta t) - v^n(1 + \frac{1}{2}\lambda\Delta t) = 0$
     $[(1 - \frac{1}{2}\lambda\Delta t)E - (1 + \frac{1}{2}\lambda\Delta t)]v^n = 0$
     $P(E)v^n = 0$
   - solve $P(E) = 0$ to find the amplification error
     $\Lambda = \frac{1+\frac{1}{2}\lambda\Delta t}{1-\frac{1}{2}\lambda\Delta t}$
   - compare Λ to the expansion of $e^{\lambda\Delta t}$ to find that this method is second order accurate
   - to find stability conditions, look at $|\Lambda|$, which shows if $\lambda \leq 0$ then it's stable

Implementation of the Time Advancement
Inserting the spatial discretization into the Crank-Nicolson scheme gives:

$$u^{n+1} = u^n + \frac{\Delta t}{2}[Au^n + f^n + Au^{n+1} + f^{n+1}] \tag{29}$$

where f^n and f^{n+1} are vectors that specify spatial boundary conditions

This can be rewritten as the following:

$$(I - \frac{\Delta t}{2}A)u^{n+1} = (I + \frac{\Delta t}{2}A)u^n + \frac{\Delta t}{2}(f^n + f^{n+1}) \tag{30}$$

This can be simplified to:

$$\hat{A}u^{n+1} = b \tag{31}$$

There are two approaches to solving this equation; direct solver and iterative solvers. Direct solvers give a solution in a finite number of steps, but the accuracy can't be controlled. This uses Gaussian Elimination to get a system of linear equations which solves the above equation. Iterative solvers satisfy accuracy criteria. There are two main types; stationary and non-stationary methods.

Types of Iterative Solvers

1. Jacobi Method: sets a stopping criteria and initial values for u
   $u_i^{N+1} = \frac{1}{a_{ii}}[f_i - \sum_{j=1}^{n} a_{ij}u_j^N]$
   where $i \neq j$, i represents the rows, and the superscripts represent the iteration of the method

2. The Gauss-Seidel Method: this is a modification of the Jacobi Method where the updates to the unkowns are incorporated into the scheme as they occur and uses the following equation
   $u_i^{N+1} = \frac{1}{a_{ii}}[f_i - \sum_{j<i} a_{ij}u_j^{N+1} - \sum_{j>i} a_{ij}u_j^N]$

3. Successive Overrelaxation Method: this averages the Gauss-Seidel iterate with the previous iterate which give the following equations
   $u_i^{N+1} = \omega\tilde{u}_i^{N+1} + (1-\omega)u_i^N$
   where $\tilde{u}_i^{N+1} = \frac{1}{a_{ii}}[f_i - \sum_{j<i} a_{ij}u_j^{N+1} - \sum_{j>i} a_{ij}u_j^N]$
   and ω is the overrelaxation parameter

Boundary Conditions Boundary conditions may have infinite domains but in finance the boundaries can be set far enough from the region of interest so in practicality it won't effect to solution.

Example: A European Call option

- Boundary Condition: $C(0, t) = 0$
  Implementation: set a minimum value for S, $S_{min} = 0$
  gives $C(S_{min}, t) = 0 = C(0, t)$
- Boundary Condition: $C(S, t) \sim S$ as $S \to \infty$
  Implementation: set a maximum value for S, S_{max}, large enough so S is highly unlikely to get there
- Boundary Condition: $C(S_{max}, t) = S_{max} - Ke^{-r(T-t)}$
  Implementation: as $S \to \infty, S - Ke^{-r(T-t)} \approx S$
  as $t \to T, Ke^{-r(T-t)}$ is increasing
  thus $S_{max} - Ke^{-r(T-t)}$ is decreasing to the payoff $S_{max} - K$

American Derivatives

- The holder faces an optimal exercise problem.
- In region A it's optimal to hold the option. The option can be treated as a European Option.
- In region B it's optimal to exercise the option. The option has exercise value $f(s, t)$.
- The option is priced via dynamic optimization.

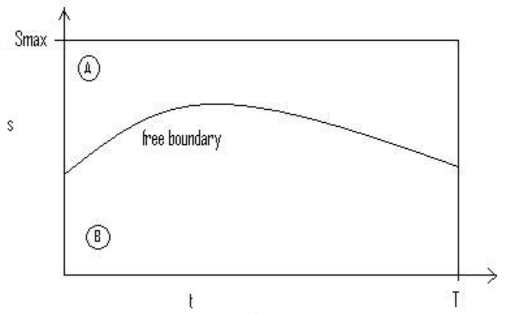

Figure 2: American Derivative

Bellman Principle At a given time, the optimal exercise strategy is the maximum of either the exercise value or the value associated with selecting an optimal strategy later.

Bellman Equation

$$V(S_t) = \max(F(S_t), PV_t[V(S_t + dS_t, t + dt]) \tag{32}$$

- S_t is the underlying
- $F(S)$ is the exercise value which depends only on S
- PV_t is the present value at time t
- this a recursive structure
- starts with the final condition at expiry and works backwards in time
- solves for the option value and the optimal strategy

Pricing Equation

$$V(r,S,t)=\sup_{\tau}\tilde{E}_{s,t}[e^{-r(T-t)}f(S_\tau)] \tag{33}$$

τ are all stopping times conditional on information available at time t.
$\tilde{E}$ is the expectation under the martingame measure.
The pricing equation will hold if the partial differential complementary problem is satisfied.

Partial Differential Complementary Problem

1. $V \geq f$
   the option value can never be below its immediate exercise value
2. $\frac{\partial dV}{\partial dt}+rS\frac{\partial dV}{\partial dS}+\frac{1}{2}\sigma^2S^2\frac{\partial d^2V}{\partial dS^2}\leq rV$
   if the option value is growing more slowly than the money market account, you should exercise
3. $(\frac{\partial dV}{\partial dt}+rS\frac{\partial dV}{\partial dS}+\frac{1}{2}\sigma^2S^2\frac{\partial d^2V}{\partial dS^2}-rV)(V-f)=0$
   complemetary condition; early exercise or the Black-Scholes partial differential equation is satisfied
4. $V(T,S)=f(S)$
   this is the payoff function

Finite Difference Approach to American Options

- Consider a general linear complementary problem, find $\bar{x}$ that satisfies:
  $A\bar{x}\geq\bar{b}$
  $\bar{x}\geq\bar{c}$
  $(\bar{x}-\bar{c})(A\bar{x}-\bar{b})=0$
- Want the partial differential complementary problem to fit this form.
- Consider an option with time-t value $u(S,t)$
- Define differential operator $L=rS\frac{\partial d}{\partial dS}+\frac{1}{2}\sigma^2S^2\frac{\partial d^2}{\partial dS^2}-r$
- $Lu=\frac{\partial du}{\partial dt}$ is the Black-Scholes partial differential equation after the time change $\hat{t}=T-t$

- Set up the partial differential complementary problem with exercise value $F(S,t)$
  1. $u(S,t) \geq F(S,t)$
  2. $\frac{\partial du}{\partial dt} - Lu \geq 0$
  3. $(\frac{\partial du}{\partial dt} - Lu)(u - F) = 0$
  4. $u(S,0) = F(S,0)$
- Use Crank-Nicolson to approzimate Lu to get the following:

$$(I - \frac{1}{2}\Delta t A)u^{n+1} = (I - \frac{1}{2}\Delta t A)u^n + \frac{\Delta t}{2}(f^n + f^{n+1}) \tag{34}$$

  where f^n and f^{n+1} are boundary conditions

- Let $M = I - \frac{1}{2}\Delta t A$ to give $Mu^{n+1} = \bar{b}$
- Let $\bar{F}$ be a discrete approximation to the exercise value F
- Apply the above to the partial differential complementary problem to get:
  1. $u^{n+1} \geq \bar{F}$
  2. $Mu^{n+1} \geq \bar{b}$
  3. $(Mu^{n+1} - \bar{b})^T(u^{n+1} - \bar{F}) = 0$
  4. $u^0 = \bar{F}$
- The above system needs to be solved at each time step
- Simplify with two substitutions:
  1. $\bar{z} = u - \bar{F}$
  2. $\bar{q} = MF - \bar{b}$
- Now the linear complementary problem is:
  1. $\bar{z} \geq \bar{0}$
  2. $\bar{q} + M\bar{z} \geq \bar{0}$

3. $\bar{z}^T(\bar{q} + M\bar{z}) = \bar{0}$

- This has a unique solution if and only if M is a P-matrix.
  i.e. if all its eigenvalues are positive
- $\bar{z}$ is a solution to the linear complementary problem if and only if it satisfies the component wise minimum $min(\bar{z}, \bar{q} + M\bar{z}) = \bar{0}$
- Suppose M=B+C where B is non-singular
- At the kth iteration, if $\bar{z}^k$ is known, consider finding $\bar{z}^{k+1}$ such that:
  $\min(B\bar{z}, \bar{q} + C\bar{z}^k + B\bar{z}^{k+1}) = \bar{0}$
  $\min(0, \bar{q} + C\bar{z}^k) = B\bar{z}^{k+1}$
  $\min(0, \bar{q} + C\bar{z}^k + B\bar{z}^k - B\bar{z}^k) = B\bar{z}^{k+1}$
  $\min(0, \bar{q} + M\bar{z}^k - B\bar{z}^k) = B\bar{z}^{k+1}$
  $\bar{z}^{k+1} = B^{-1}\min(0, \bar{q} + M\bar{z}^k - B\bar{z}^k)$
- There are other choices for B:
  1. The Projected Jacobi Method: set B as the diagonal of M
  2. Projected Successive Overrelaxation Method: set $B = L + \frac{1}{\omega}D$
     L is the strictly lower triangular part of M
     D is the diagonal of M
     ω is the overrelaxation parameter

Monte Carlo Methods

Introduction

Monte Carlo Methods rely on probability and statistics
Have a sample space Ω and an event
Find a set of outcomes in Ω that lead to this event occuring (call this set A)
$A \in \mathcal{F}$
P(A) is the probability of the event occuring

Monte Carlo:

- A different approach to the above calculation
- Randomly sample $\omega \in \Omega$ many times
- For each sampled ω, determine whether or not the event occurs
- P(A) is approximated by the fraction of outcomes that caused the event to occur
- The law of large numbers ensures that this estimate converges to P(A) as the number of draws goes to ∞
- The central limit theorem gives information about the error of our approximation

Weak Law of Large Numbers:
For any $\epsilon > 0$, $\lim P[|\bar{X}_n - \mu| \leq \epsilon] = 1$ as $n \to \infty$ ($\bar{X}_n$ converges in probability). Also written as $\bar{X}_n \to \mu$ in probability

Strong Law of Large Numbers:
$P[\lim \bar{X}_n = \mu] = 1$ as $n \to \infty$
this law implies the weak law
the events for which $\bar{X}_n$ does not converge to μ have probability zero
also written as $\bar{X}_n \to \mu$ almost surely

Central Limit Theorem:
If $Var[X_i] = \sigma^2 < \infty$, then $\forall a \leq b, \lim P[a \leq \frac{(\bar{X}_n - \mu)(\sqrt{n})}{\sigma^2} \leq b] = \Phi(b) - \Phi(a)$

- Φ is the cumulative distribution function of the standard normal distribution
- $\frac{(\bar{X}_n - \mu)(\sqrt{n})}{\sigma^2} \sim N(0,1)$
- $\bar{X}_n \sim N(\mu, \frac{\sigma^2}{\sqrt{n}})$

Monte Carlo Example:

- calculate $\alpha =_0^1 f(x)dx$
- can think of α as an expectation, E[f(u)], where U is uniformly distributed on [0,1]
- sample $U_1, U_2, ...$ are indepently and uniformly distributed from [0,1]
- form $\hat{\alpha_n} = \frac{1}{n}\sum_{i=1}^{n} f(u_i)$
- if f is integrable on [0,1], then the strong law of large numbers implies $\hat{\alpha_n} \rightarrow \alpha$ almost surely
- if f is square integrable on [0,1],
  $\sigma_f^2 = Var[f(u)] = E[(f(u) - E[f(u)])^2] = \int_0^1 [f(x) - \alpha]^2 dx$
- consider the error of our approximation, $\hat{\alpha_n} - \alpha$
- the central limit theorem implies the distribution of $\hat{\alpha_n}$ is approximately $N(\alpha, \frac{\sigma_f^2}{n})$
- our error is thus approximately $N(0, \frac{\sigma_f^2}{n})$
  σ_f^2 is unknown, but the sample standard deviation can be used to estimate it
  $S_f = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[f(u_i) - \hat{\alpha_n}]^2}$
  have error estimate $\frac{\sigma_f}{\sqrt{n}} \approx \frac{S_f}{\sqrt{n}}$

- from $f(u_1), f(u_2)$,... an estimate of α can be found and an error estimate $\frac{S_f}{\sqrt{n}}$
  error is $\mathcal{O}(\frac{1}{\sqrt{n}})$
- compare this to the trapazoid rule:
  $\hat{\alpha_n} = \frac{f(0)+f(1)}{2n} + \frac{1}{n}\sum_{i=1}^{n-1} f(\frac{i}{n})$
  error bound $|\hat{\alpha_n} - \alpha| \leq \frac{k}{12n^2}$, where k is bound on $|f''(x)|$ on [0,1] if it exists
  error is $\mathcal{O}(\frac{1}{n^2})$, which is clearly better than Monte Carlo

Multiple Integrals:

- $\int_{[0,1]^d} f(\bar{x})d\bar{x}, \bar{x} \in \mathbb{R}^d$
- based on n draws from $[0,1]^d$, we get error estimates:
  trapaziod rule: $\mathcal{O}(\frac{1}{n^{2/d}})$
  monte carlo: $\mathcal{O}(\frac{1}{\sqrt{n}})$
- once $d > 4$, monte carlo has a better rate of convergence
- monte carlo is useful for computing multiple integrals

Corollary to Fundamental Theorem of Asset Pricing:
Let V_t be the time-t price of a European-style option. Assume the market has a risk-neutral probability measure $\tilde{P}$. Then the no arbitrage price V_0 of the option is $V_0 = \tilde{E}(D_T V_T)$, where $\tilde{E}$ denotes expectation under the risk-neutral measure $\tilde{P}$ and D_t is the discount factor.
 for fixed interest rate, $D_t = e^{-r(T-t)}$
 for variable interest rate, r_t, $D_t = e^{-\int_t^T r_s ds}$

European Call Option

- strike K, underlying S_t, maturity T, interest rate r (r is drift for stock)
- the underlying asset price evolves accordingly to the stochastic differential equation
  $\frac{dS_t}{S_t} = rdt + \sigma dW_t$
- this has solution $S_T = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma W_T}$
  where $W_T \sim N(0,T)$
- this can be represented by $S_T = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z}$
- take logs of both sides to get a normally distributed random variables
  $ln(S_T) \sim N((r-\frac{1}{2}\sigma^2)T, \sigma^2 T)$
- recall, $V_0 = S_T\Phi(d_1) - e^{-rT}K\Phi(d_2)$
  $\Phi(y)$ is a standard normal cummalative distribution function
  $d_1 = \frac{ln\frac{S}{K}+(r+\frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$
  $d_2 = \frac{ln\frac{S}{K}+(r-\frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$

Monte Carlo Pricing:
$V_0 = \tilde{E}[e^{-rT}(S_T-K)^+]$
Generate a sequence of standard normal random variables $Z_1, Z_2, ...$ and use these to estimate V_0

Algorithm:
```
for i=1:n
    generate $Z_i$
    set $S_i(T) = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z_i}$
    set $c_i = e^{-rT}(S_i(T)-K)^+$
end
set $\hat{c_n} = \frac{1}{n}\sum_{i=1}^{n} c_i$
```

Confidence Intervals:

- To control the error of the approximation, use the sample standard deviation of $c_1, c_2, ..., c_n$
- $S_c = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(c_i - \hat{c_i})^2}$
- Let Z_δ denote the $(1-\delta)$ quantile of the standard normal
  $\Phi(Z_\delta) = 1 - \delta = P[Z \leq z_\delta]$
- By the central limit theorem, $\frac{\hat{c_n} - c_0}{S_c/\sqrt{n}}$ converges to N(0,1)
- $(\hat{c_n} - z_{\delta/2}\frac{S_c}{\sqrt{n}}, \hat{c_n} + z_{\delta/2}\frac{S_c}{\sqrt{n}})$ is an asymptotically valid $(1-\delta)$ confidence interval for c_0

Random Number Generation

- want to simulate randomness - pseudorandom
- generate a sequence of random variables $U_1, U_2, ...$ with two properites:
  1. U_i is uniformly distributed on [0,1]
  2. the U_i are independent
- $[0, 1]$ in the first above property is arbitrary
- mimic randomness: produce finite sequences $u_1, ..., u_k \in [0, 1]$, where k is large
- the u_i constitutes possible outcomes for the independent uniforms $U_1, ..., U_k$
- small (relative to k) segments of this sequence should be difficult to distinguish from the realization of independent uniforms
  i.e. statistical tests for independence should not easily reject segments of $u_1, ..., u_k$

Linear Congruential Generator:
for a, m, c $\in \mathbb{Z}$

- $x_{i+1} = (aX_i + c) \mod m$
- $u_{i+1} = \frac{x_{i+1}}{m}$
- if $c \neq 0$, it's called mixed
- if $c = 0$, it's called pure
- little generatlity is achieved for $c \neq 0$ and scheme is slower, so usually the pure case is used
- a is called the multiplier
- m is the modulus
- an initial seed x_0 is required, where $1 \leq x_0 \leq m-1$

Modulus:
for $y, m \in \mathbb{Z}$, ymodm returns the remainder of y after dividing by m
$y \mod m = y - \left\lfloor \frac{y}{m} \right\rfloor m$

Notes:

- in general, $0 \leq amodm \leq m-1$
- when $c = 0, 0 \leq x_{i+1} \leq m-1$
  therefore $0 \leq u_{i+1} = \frac{X_{i+1}}{m} \leq \frac{m-1}{m} \leq 1$
  thus $u_{i+1} \in [0,1]$

Example:
a=6, m=11, $x_0 = 1$, c=0
$x_1 = 6, x_2 = 3, x_3 = 7, x_4 = 9, x_5 = 10, x_6 = 5, x_7 = 8, x_8 = 4, x_9 = 2, x_{10} = 1$
after this the values start repeating
here, all integers in the interval [1,m-1] appeared

Example:
a=3, m=11, $x_0 = 1$, c=0
$x_1 = 3, x_2 = 9, x_3 = 5, x_4 = 4, x_5 = 1$
here, only five distinct values appears, which shows that a needs to be chosen carefully

Full Period:
A linear congruential generator that produces all m-1 distinct values is said to have a full period. In general, choose m large and a needs to be chosen carefully

Issues for Random Number Generators:

1. Period Length: the longer the better
   the gaps between the u_i have size $\frac{1}{m}$
   the larger the m, the better the approximation to the uniform
2. Reproducibility: the sequences can be reproduced
3. Speed: generators are used many times
4. Portability: an alogrithm should produce the same sequence on any platform
5. Randomness: theoretical properties for construction
   statistical tests to scrutinize results

Theorem:
Suppose $c \neq 0$. For any seed x_0, the linear congruential generator generates m-1 distinct values if:

1. c and m are relatively prime (their only common divisor is 1)
2. every prime number that divides m also divides (a-1)
3. (a-1) is divisible by 4 if m is divisible by 4

Consequence:
The generator has full period if $m = 2^N$, c is odd, and a=4n+1 for some n

Theorem:
Suppose $c = 0$. If m is prime, for any seed $x_0 \neq 0$, the linear congruential generator generates m-1 distinct values if:

1. $a^{m-1} - 1$ is a multiple of m
2. $a^j - 1$ is not a multiple of m for j=1,2,...,m-2

A number satisfying these two properties is called a primative root of m.

Property:
If a is a primitive room of m, then all the x_i are non-zero if $x_0 \neq 0$.
General Sampling Methods:
Assume an available sequence of independent uniformly distributed random variables on [0,1]; $U_1, U_2, ...$

$$P[U_i \leq u] = \begin{array}{l} 0, u < 0 \\ u, 0 \leq u \leq 1 \\ 1, 1 < u \end{array} \tag{35}$$

want to transform these randrom variables into paths of stochastic processes

Inverse Transform Method:

- Random variable X with cummalative distribution function F
- If X has density function f, $F(x) = \int_{-\infty}^{x} f(y)dy$
- If U is uniform $[0, 1]$, it can be interpreted as a probability
- Since F is a cummalative density function, it is monotone increasing
- If F is strictly increasing, it has an inverse
- If F^{-1} denotes the inverse of F, we set $X = F^{-1}(U)$
  if $F(0) \leq u_1 < 1$ then $x_1 \geq 0$

Verification:
Make sure $X = F^{-1}(U)$ actually generates samples from F (or X).
$P[X \leq x] = P[F^{-1}(U) \leq x] = P[U \leq F(x)] = length([0, F(x)]) = F(x)$

Example: Exponential Distribution(θ)

- $F(x) = 1 - e^{-x/\theta}$
- $f(x) = \frac{1}{\theta}e^{-x/\theta}$
- Invert F(x)
- $U = 1 - e^{-x/\theta}$ and solve for x
- therefore $x = -\theta \ln(1 - U)$
- if uniforms are given to the above formula it will produce exponentials
- if $U \sim U[0, 1]$, then so is 1-U, so simplify above to $x = -\theta \ln(U)$

Acceptance-Rejection Method:

- First generate samples from a convenient distribution. Then reject a random subset. The accepted sampes are distibuted according to the target distribution.
- Suppose we have a density function f defined on some subset $X \subseteq \mathbb{R}^d$
- Let g be a density on X from which we can generate samples such that $f(x) \leq cg(x)$, sor some constant $c \geq 1, \forall X$

Method:

- Generate a sample X from g
- Accept the sample with probability $\frac{f(x)}{cg(x)} \leq 1$
- Specifically, the uniform distribution can be used
  - sample U uniformly on $[0, 1]$
  - accept X if $U \leq \frac{f(x)}{cg(x)}$
  - if X is rejected, sample X from g again and sample U again
  - repeat

Verification:

- Suppose Y is returned by our algorithm
- Then Y has the distribution of X conditional on $U \leq \frac{f(x)}{cg(x)}$
- Let $A \subseteq X$. Look at $P[Y \in A]$.
  $P[Y \in A] = P[x \in A | U \leq \frac{f(x)}{cg(x)}]$
- If give X, $P[U \leq \frac{f(x)}{cg(x)}] = \frac{f(x)}{cg(x)}$
- For X, $P[U \leq \frac{f(x)}{cg(x)}] = \int_x \frac{1}{c} f(x) dx = \frac{1}{c}$
- Plug in to get: $P[Y \in A] = P[X \in A, U \leq \frac{f(x)}{cg(x)}]c = \int_A f(x) dx$
  Therefore Y has density f(x)

Notes:

1. The probability of accepting a draw is $\mathcal{O}(\frac{1}{c})$
2. If c is large, it's less likely to be accepted (good to have c close to 1)

Example: Normal from Double Exponential

- A half-normal random variable as the distribution of the absolute value of a normal random variable.
- The double exponential on $(-\infty, \infty)$ has distribution $g(x) = \frac{1}{2}e^{-|x|}$
- Normal density function is $f(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2} \sim N(0,1)$
- Ratio: $\frac{f(x)}{g(x)} = \sqrt{\frac{2}{\pi}}e^{-x^2/2+|x|} \leq \sqrt{\frac{2e}{\pi}} \approx 1.3155 = c$
- To sample a double exponential, draw a standard exponential, $x = -\theta ln(U)$ where $U \sim U[0,1]$, Then randomize the sign
- Rejection Test: $U \geq \frac{f(x)}{cg(x)} = e^{-x^2/2+|x|-1/2} = e^{-(|x|-1)^2/2}$

Algorithm:

1. Generate $U_1, U_2, U_3 \sim U[0,1]$
2. $X \leftarrow -\theta \ln(U_1)$
3. If $U_2 > e^{-(|x|-1)^2/2}$, then go to step 1
4. If $U_3 \leq \frac{1}{2}$, then $X \leftarrow -X$
5. Return X

Normal Random Variables:

- If $Z \sim N(0,1)$, then $\mu + \sigma Z \sim N(\mu, \sigma^2)$
- Thus to generate normal random variables, we need only to generate standard normals
- A d-dimensional normal distribution is characterized by $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{dxd}$

Properties of Σ:

1. Σ is symmetric, $\Sigma = \Sigma^T$
2. Σ is positive semi-definite

Positive Definite:
A matrix $\Sigma \in \mathbb{R}^{dxd}$ is positive definite if $x^T \Sigma x > 0 \forall x \in \mathbb{R}^d$ with $x \neq 0$. Σ is invertible.

Positive Semi-Definite:
A matrix $\Sigma \in \mathbb{R}^{dxd}$ is positive semi-definite if $x^T \Sigma x \geq 0 \forall x \in \mathbb{R}^d$ with $x \neq 0$.

Notes:

- If Σ is positive semi-definite, then it may not be positive definite. There might be an $x \neq 0$ such that $x^T \Sigma x = 0$. If that's the case, Σ is not invertible.
- If Σ is positive definite, $N(\mu, \Sigma)$ has density:

$$\Phi(x) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{\frac{-1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{36}$$

- The standard d-dimensional normal $N(0, I_d)$ has density:

$$\Phi(x) = \frac{1}{(2\pi)^{d/2}} e^{\frac{-1}{2} x^T x} \tag{37}$$

- If $x \sim N(\mu, \Sigma)$ then its i^{th} component X_i has distribution $x_i \sim (\mu_i, \sigma_{ii}^2)$

- Further $Cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$
- The correlation between X_i and X_j is given by $\rho_{ij} = \frac{\Sigma_{ij}}{\sigma_{ii}\sigma jj}$

Box-Muller Method:

- generates a sample from bivariate standard normal, where each component is a standard normal
- consider $Z \sim N(0, I_2)$, two properties:
  1. $R = Z_1^2 + Z_2^2$ is exponentially distributed with $\theta = 2$
  2. given R, the point (Z_1, Z_2) is uniformly distributed on the circle of radius $\sqrt{R}$, centered at the origin
- to generate (Z_1, Z_2):
  1. generate $R : R = -2\ln(U_1), U_1 \sim U[0,1]$
  2. choose a point uniformly from the circle of radius $\sqrt{R}$
     get a new R every time (to insure independence)
     generate a random angle uniformly between 0 and 2π
     $V = 2\pi U_2$
     point on circle: $(\sqrt{R}\cos V, \sqrt{R}\sin V)$

Algorithm:

1. Generate $U_1, U_2 \sim U[0,1]$ independently
2. $R \leftarrow -2\ln(U_1)$
3. $V \leftarrow 2\pi U_2$
4. $Z_1 \leftarrow \sqrt{R}\cos V$
5. $Z_2 \leftarrow \sqrt{R}\sin V$
6. Return Z_1, Z_2

Multivariate Normals:

- $Z \sim N(\mu, \Sigma)$
- Using the correlations $\rho_{ij} = \frac{\sigma_{ij}}{\sigma_{ii}\sigma_{jj}}$, we get $\sigma_{ij} = \rho_{ij}\sigma_{ii}\sigma_{jj}$
- If Σ is positive semi-different, but not positive definite:
  - $\exists x \neq 0$ such that $x^T \Sigma x = 0$ therefore Σ is singular
  - There is no normal density with covariance matrix Σ
  - We can define $N(\mu, \Sigma)$ as the distribution of $x = \mu + AZ$ as long as $Z \sim N(0, I_d)$ for any $A \in \mathbb{R}^{dxd}$ such that $AA^T = \Sigma$. If A has rank k¡d, then one can find k components of x with multivariate density in $\mathbb{R}^k$

Theorem: Linear Transformation Property
Any linear transformation of a normal vector is normal. If $X \sim N(\mu, \Sigma)$, then $AX \sim N(A\mu, A\Sigma A^T)$ for any $\mu \in \mathbb{R}^d$, $\Sigma \in \mathbb{R}^{dxd}$, and $A \in \mathbb{R}^{kxd}$, $\forall$ k.

Generating Multivariate Normals

- Generate independent $Z_1, Z_2, ..., Z_d \sim N(0, 1)$ and put them in a vector $Z \sim N(0, 1)$
- Then $AZ \sim N(0, AA^T)$
- Sampling X from $N(\mu, \Sigma)$ reduces to finding a matrix A with $AA^T = \Sigma$
- There are two cases: positive definite and non-positive definite Σ

Theorem: Cholesky Factorization

- Suppose $\Sigma \in \mathbb{R}^{dxd}$ is positive definite. Then $\exists$ a lower triangular matrix $A \in \mathbb{R}^{dxd}$ such that $\Sigma = AA^T$. A is unique up to changes in sign.
- Consider the computation of $X = \mu + AZ$. The matrix vector product AZ has fewer multiplications than if A was dense (by almost half)

Example (2x2 case):

- Assume Σ is positive definite

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{bmatrix} \tag{38}$$

- Want:

$$\Sigma = AA^T = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \tag{39}$$

- Therefore:

$$\begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} \\ 0 & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11}^2 & A_{21}A_{11} \\ A_{21}A_{11} & A_{21}^2 + A_{22}^2 \end{bmatrix} \tag{40}$$

- Can solve for A to get:

$$A = \begin{bmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sigma_2\sqrt{1-\rho^2} \end{bmatrix} \tag{41}$$

General Case: $\Sigma \in \mathbb{R}^{d\times d}$

- $\Sigma = AA^T$ by the Cholesky Factorization Theorem, which gives:

$$\Sigma = \begin{bmatrix} A_{11} & 0 & 0 & \dots & 0 & 0 \\ A_{21} & A_{22} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{(d-1)1} & A_{(d-1)2} & A_{(d-1)3} & \dots & A_{(d-1)(d-1)} & 0 \\ A_{d1} & A_{d2} & A_{d3} & \dots & A_{d(d-1)} & A_{dd} \end{bmatrix} \tag{42}$$

- Over row 1 of Σ
  $\Sigma_{11} = A_{11}^2$
  $\Sigma_{12} = A_{11}A_{21}$
  $\vdots$
  $\Sigma_{1d} = A_{11}A_{d1}$

- Over row 2 of Σ
  $\Sigma_{21} = A_{21}A_{11}$
  $\Sigma_{22} = A_{21}^2 + A_{22}^2$
  $\vdots$
  $\Sigma_{2d} = A_{21}A_{d1} + A_{22}A_{d2}$
- Over row d of Σ
  $\Sigma_{d1} = A_{11}A_{d1}$
  $\Sigma_{d2} = A_{21}A_{d1} + A_{22}A_{d2}$
  $\vdots$
  $\Sigma_{dd} = A_{d1}^2 + A_{d2}^2 + \ldots + A_{dd}^2$
- While working through the rows and columns of Σ, in each equation, exactly one new entry from A appears. Can solve for A_{ij}

General Solution (when Σ is positive definite):

- $\Sigma_{ij} = \sum_{k=1}^{j} A_{ik}A_{jk}$, for $j \leq i$
- Gives: $A_{ij} = \frac{1}{A_{jj}}(\Sigma_{ij} - \sum_{k=1}^{j-1} A_{ik}A_{jk})$, for $j < i$
  $A_{ii} = (\Sigma_{ii} - \sum_{k=1}^{i-1} A_{ik}^{i-1})^{\frac{1}{2}}$
- If Σ is positive semi-definite and not positive definite, one of these terms will be zero

Algorithm (Cholesky Factorization):
Given Σ is symmetric and positive-definite

```
$A \leftarrow 0 \in \mathbb{R}^{dxd}$
for j=1:d
    for i=j:d
        $v_i = \Sigma_{ij}$
        for k=1:j-1
            $v_i \leftarrow v_i - A_{jk}A_{ik}$
        end
        $A_{ij} \leftarrow v_i/\sqrt{v_j}$
    end
end
return A
```

Semi-Definite Σ (but not positive-definite):

- Σ is singular.
- If $AA^T = \Sigma$, then A is singular.
- Suppose A is lower triangular. Since it's rank deficient, some diagonal element $A_{jj} = 0$.
- Therefore the Cholesky algorithm fails because of a division by 0.
- If $A_{jj} = 0$, we set column j of A to 0.

Algorithm:
Given Σ is symmetric and positive semi-definite but now positive-definite
Same as positive definite case with one update

```
$A \leftarrow 0 \in \mathbb{R}^{dxd}$
for j=1:d
    for i=j:d
        $v_i = \Sigma_{ij}$
        for k=1:j-1
            $v_i \leftarrow v_i - A_{jk}A_{ik}$
        end
        if $v_j > 0$
            $A_{ij} \leftarrow v_i/\sqrt{v_j}$
        end
    end
end
return A
```

Problem:
In practice, if $v_j > 0$ is checking that $v_j \neq 0$, however, if v_j should be 0, it may be positive and very small on a machine.

Problem Reduction:

- $X \sim N(0, \Sigma)$
- Suppose $\text{rank}(\Sigma) = k < d$
- The components of $x \in \mathbb{R}^d$ can be expressed as a linear combination of k components i.e. $\exists$ a subvector $\tilde{x}$ of x and a matrix $D \in \mathbb{R}^{d \times k}$ such that $D\tilde{x} \sim N(0, \Sigma)$ and the covariance matrix of $\tilde{x}$, $\tilde{\Sigma}$ has full rank k
- We can find the Cholesky factorization of $\tilde{\Sigma}$, $\tilde{\Sigma} = \tilde{A}\tilde{A}^T$
- We recover x using $x = D\tilde{A}Z$, $Z \sim N(0, I_d)$
- Situation arises if d variables are generated using $k < d$ sources of uncertainty

Generating Sample Paths

Stochastic Process:
A standard one-dimensional Brownian Motion is a stochastic process, $W_t : 0 \leq t \leq T$ such that:

1. $W_0 = 0$
2. W_t is continuous on $[0, T]$ almost surely
3. W has independent increments
4. $(W_t - W_s) \sim N(0, t - s)$ for any $0 \leq s < t \leq T$

Notes:

- In many applications we need the entire path of an asset price
- For constants $\mu, \sigma > 0$, we call a process X_t a Brownian Motion with drift μ and diffusion coefficient σ^2
  - $X \sim BM(\mu, \sigma^2)$ if $\frac{X_t - \mu t}{\sigma}$ is a standard Brownian Motion

  - given a standard Brownian Motion W_t, a Brownian Motion $X \sim BM(\mu, \sigma^2)$ can be constructed by setting $X_t = \mu t + \sigma W_t$

- Further, X_t solves the stochastic differential equation $dX_t = \mu dt + \sigma dW_t$
- A Brownian Motion can be defined with deterministic drift $\mu(t)$ and diffusion coefficient $\sigma(t)$ through $dX_t = \mu(t)dt + \sigma(t)dW_t$
- Stochastic integration is needed to find the solution:

$$X_t = X_0 + \int_0^t \mu(s)ds + \int_0^t \sigma(s)dW_s \tag{43}$$

- In this case $(X_t - X_s) \sim N(\int_s^t \mu(u)du, \int_s^t \sigma^2(u)du)$

Random Walk Construction:

- focus: simulate Brownian Motion at a fixed set of times $0 < t_1 < t_2 < ... < t_n$
- use the properties that increments are normal and independent
- suppose $Z_1, Z_2, ..., Z_n \sim N(0,1)$ independently
- set $t_0 = 0$ and $W_0 = 0$
- generate a standard Brownian Motion using:

$$W_{t_{i+1}} = W_{t_i} + \sqrt{t_{i+1} - t_i} Z_{i+1}, for i = 0, 1, ..., n-1 \tag{44}$$

- to generate $X \sim BM(\mu, \sigma^2)$ given X_0, then:

$$X_{t_{i+1}} = X_{t_i} + \mu(t_{i+1} - t_i) + \sigma\sqrt{(t_{i+1} - t_i)} Z_{i+1} \tag{45}$$

- for Brownian Motion with time dependent $\mu(t)$ and $\sigma(t)$

$$X_{t_{i+1}} = X_{t_i} + \int_{t_{i+1}}^{t_i} \mu(s)ds + \sqrt{\int_{t_{i+1}}^{t_i} \sigma^2(u)du Z_{i+1}} \tag{46}$$

- These methods are exact, meaning that the joint distribution of the simulated values match that of the true Brownian Motion.
  - for values between t_i and t_{i+1} there is error
  - usually linearly interpolate between the times

Alternative Construction:

- Consider the vector $[W_{t_1}, W_{t_2}, ..., W_{t_n}]^T$
- This is a linear transformation of the increments $[W_{t_1} - W_{t_0}, ..., W_{t_n} - W_{t_{n-1}}]$
  These increments are independent and normally distributed
  Therefore $[W_{t_1}, W_{t_2}, ..., W_{t_n}]$ is multivariate normal
- E$[W_{t_i}]$=0
- $Cov(W_s, W_t) = Cov(W_s, W_s) + Cov(W_s, W_t - W_s) = Var[W_s] = s$
- Let C be the covariance matrix for $[W_{t_1}, W_{t_2}, ..., W_{t_n}]^T$
  Then, $C_{ij} = min(t_i, t_j)$
- This random vector has mean 0
- Since $[W_{t_1}, W_{t_2}, ..., W_{t_n}]^T \sim N(0, C)$, simulate using AZ, where Z is a vector of standard normals and A is the Cholesky factorization of C
- The Cholesky factorization of C gives:

$$A = \begin{bmatrix} \sqrt{t_1} & 0 & 0 & \dots & 0 & 0 \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & \sqrt{t_3 - t_2} & \dots & \sqrt{t_{n-1} - t_{n-2}} & 0 \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & \sqrt{t_3 - t_2} & \dots & \sqrt{t_{n-1} - t_{n-2}} & \sqrt{t_n - t_{n-1}} \end{bmatrix} \tag{47}$$

Standard Brownian Motion:
A process $W_t = [W_1(t), ..., W_d(t)]^T$, is a standard Brownian Motion on $\mathbb{R}^d$ if:

1. $W_0 = 0$
2. W has continuous sample paths almost surely
3. W has independent increments
4. $(W_t - W_s) \sim N(0, (t-s)I)$

Each $W_i(t)$, i=1,...,d is a standard Brownian Motion

Brownian Motion: Suppose $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d\times d}$ which is positive semi-definite. Say X is a Brownian Motion with drift μ and covariance Σ if X has continuous sample paths and independent increments with $(X_t, X_s) \sim N((t-s)\mu, (t-s)\Sigma)$

If $B \in \mathbb{R}^{d\times k}$ is such that $BB^T = \Sigma$ and W is a standard Brownian Motion on $\mathbb{R}^k$, then $X_t = \mu t + BW_t$ is a BM(μ, Σ), where X solves $dX_t = \mu dt + BdW_t$

Simulation:

- Let $Z_1, Z_2, ... \sim N(0,1)$ independent. To simulate W_t, apply the one-dimensional random walk construction to each componenet of W_t.

$$W_j(t_{i+1}) = W_j(t_i) + \sqrt{t_{i+1} - t_i} Z_{i+1} \tag{48}$$

  i=0,1,...,n-1, for each j

- To simulate $X_t \sim BM(\mu, \Sigma)$
  1. find $B \in \mathbb{R}^{dxk}$ such that $BB^T = \Sigma$
  2. set $X_0 = 0$
  3. $X_{t_{i+1}} = X_{t_i} + \mu(t_{i+1} - t_i) + \sqrt{t_{i+1} - t_i} BZ_i$

Geometric Brownian Motion:
A stochastic process S_t is a geometric Brownian Motion if $\ln(S_t)$ is a Brownian Motion with initial value $\ln(S_0)$. To simulate geometric Brownian Motion use exponentiation.

Fundamental Property for Financial Modeling:
If S_t is geometric Brownian Motion, then S_t does not have independent increments. Instead, $\frac{S_{t_2}-S_{t_1}}{S_{t_1}}, \frac{S_{t_3}-S_{t_2}}{S_{t_2}}, ..., \frac{S_{t_n}-S_{t_{n-1}}}{S_{t_{n-1}}}$ are independent.

Stochastic Differential Equation for Geometric Brownian Motion:

- Suppose W is a standard Brownian Motion and X satisfies $dX_t = \mu dt + \sigma dW_t$
- Then $X \sim BM(\mu, \sigma^2)$
- Let $S_t = S_0 e^{x_t} = f(x_t)$, where S_0 is the initial stock price
- $dS = f_t(x_t)dt + f_x(x_t)dX_t + \frac{1}{2}f_{xx}(x_t)dX_t^2$
- Thus $dS_t = 0 + S_0 e^{X_t}(\mu dt + \sigma dW_t) + \frac{1}{2}\sigma S_0 e^{x_t} dt$
  $dS_t = S_t(\mu + \frac{1}{2}\sigma^2)dt + S_t \sigma DW_t$
  $\frac{dS_t}{S_t} = (\mu + \frac{1}{2}\sigma^2)dt + \sigma DW_t$
- This is a differnt stochastic differential equation than what is usually used for geometric Brownian Motion.
- The usual model is: $\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$, where μ is the drift for the geometric Brownian Motion
- If $S_t \sim GBM(\mu, \sigma^2)$, then the solution is $S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t}$
- For $u < t$, $S_t = S_u e^{(\mu - \frac{1}{2}\sigma^2)(t-u) + \sigma(W_t - W_u)}$
- To simulate, use $S_{t_{i+1}} = S_{t_i} e^{(\mu - \frac{1}{2}\sigma^2)(t_{i+1}-t_i) + \sigma\sqrt{t_{i+1}-t_i}Z_{i+1}}$
- The exponential is the randsom walk construction of the brownian motion with drift $\mu - \frac{1}{2}\sigma^2$
- The method is exact. i.e. the resulting vector $[S_{t_1}, ..., S_{t_n}]^T$ has the joint deistribution of $S_t \sim GBM(\mu, \sigma^2)$ at times $t_1, ..., t_n$

Notation:

- Money Market Account: 1 dollar invested at time t=0 has time t value $\beta(t) = e^{rt}$
- Is S pays no dividends, $\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$
- No arbitrage condition: under the risk neutral measure, $\mu = r$, where r is the interest rate
- In the risk neutral world, all assets have the same average rate of return
- Further, under hte risk neutral measure, the discounted stock price, $\frac{S_t}{\beta(t)}$ is a martingale

Path Dependent Payoffs:
path = geoBrownianMotion(μ, σ, N)

Asian Option (with discrete monitoring):
Call: Payoff = $(\bar{S} - K)^+$
Put: Payoff = $(K - \bar{S})^+$
where K is the strike price and $\bar{S} = \frac{1}{n}\sum_{i=1}^{n} S(t_i)$ is the average price of the underlying over monitoring dates $t_1, t_2, ..., t_n$

Asian Option (with continuous monitoring):
$\bar{S} = \frac{1}{t-u}\int_u^t S(\tau)d\tau$
Continuous average of S over [u,t]
More difficult to simulate
Can find analytic solutions in some cases

Barrier Options:
Down-and-out call option has barrier b, strik K, and expiry T
Payoff $= 1_{\{\tau(b)>T\}}(S_T - K)^+$

where $\tau(b) - \inf\{t_i : S_{t_i} < b\}$ is the first time in $t_1, ..., t_n$ that the underlying price drops below b

Example:
Below used discrete monitoing simulate by sampling $S(t_0), S(t_1), ..., S(t_n)$ and keep generating price paths and taking the average
Here the Payoff=0 because it breaks the barrier (Up-and-out), however if the barrier was set at 4, Payoff=$(S_T - K)^+$
In the continuous monitoring case $\tilde{\tau}(b) = \inf t \geq 0 : S_t < b$, and often get analytic solutions

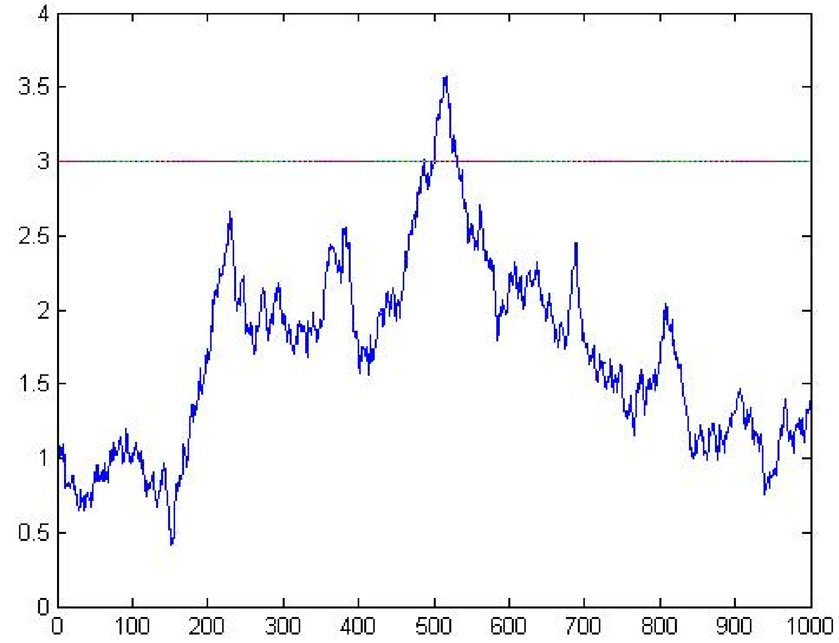

Figure 3: Barrier Option

Lookback Options:
Discrete versions:
Put: Payoff = $\max(S_{t_i}) - S_{t_n}$ for i=1,2,...,n
Call: Payoff = $S_{t_n} - \max(S_{t_i})$ for i=1,2,...,n
Call: profit from buying the underlying at the lowest price over $t_1, ..., t_n$ and selling at the final price
Continuous versions:
Put: Payoff = $\max(S_t) - S_T$ for $t \in [0, t]$
Call: Payoff = $S_T - \max(S_t)$ for $t \in [0, t]$

Incorporate a Term Structure of Interest Rates:

- If we have a constant interest rate r, the time-t price of a zero-coupon bond paying 1 dollar at time $T > t$ is $B(t,T) = e^{-r(T-t)}$
- In reality, r is not constant
- We determine the term structure of interest rates using a collection of bond prices
- Define time-varying interest rate r(u) by $r(u) = -\frac{\partial d}{\partial dt}[B(0,t)]|_{T=u}$
- Solve for B(0,T) to get $B(0,T) = e^{-\int_0^T r(u)du}$
- Under the risk-neutral measure, the dynamics of an asset price are $\frac{dS_t}{S_t} = \mu(t)dt + \sigma dW_t$ with solution $S_t = S_0 e^{\int_0^t r(u)du - \frac{1}{2}\sigma^2 t + \sigma W_t}$
- Can simulate over $0 = t_0 < t_1 < ... < t_n$ using $S_{t_{i+1}} = S_{t_i} e^{\int_{t_i}^{t_{i+1}} r(u)du - \frac{1}{2}\sigma^2(t_{i+1}-t_i) + \sigma\sqrt{t_{i+1}-t_i}Z_{i+1}}$, where $Z_1, ..., Z_n$ are independent N(0,1)
- Suppose bond prices B(0,t) are observed

$$\frac{B(0,t_i)}{B(0,t_{i+1})} = \frac{e^{-\int_0^{t_i} r(u)du}}{e^{-\int_0^{t_{i+1}} r(u)du}} = e^{\int_{t_i}^{t_{i+1}} r(u)du} \tag{49}$$

- The simulation simplifies to $S_{t_{i+1}} = S_{t_i}\frac{B(0,t_i)}{B(0,t_{i+1})}e^{-\frac{1}{2}\sigma^2(t_{i+1}-t_i)+\sigma\sqrt{t_{i+1}-t_i}Z_{i+1}}, i = 0, 1, ..., n-1$

Assets with Dividends:

- Holding a single share of an asset is no longer self-financing, strategy must deal with the dividends.
- If dividends are automatically reinvested into the asset, then the stategy is self-financing. Required neither withdrawls or deposits and the number of shares changes over time.

Model:

- S_t is the underlying asset price
- $\tilde{S}_t$ is the asset price with dividends reinvested
- $\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$
- $\frac{d\tilde{S}_t}{\tilde{S}_t} = \frac{dS_t + dD_t}{S_t}$
- dD_t is the divident payment over dt
- $\tilde{S}_t$ will have continuous paths. If D_t jumps, then S_t jumps in the opposite direction to offset
- In this case $\frac{\tilde{S}_t}{\beta(t)}$ is a martingale under the risk-neutral measure instead of $\frac{S_t}{\beta(t)}$ which shows that it's a very natual thing to reinvest dividends
- Suppose an asset pays a continuous dividend yield at a rate δ , then $dD_t = \delta S_t dt$
- Therefore $\frac{d\tilde{S}_t}{\tilde{S}_t} = \frac{dS_t + \delta S_t dt}{S_t} = \frac{dS_t}{S_t} + \delta dt$
  no jumps because it's continuous dividends
- Therefore $(\mu dt + \sigma dW_t) + \delta dt = (\mu + \delta)dt + \sigma dW_t$
  for no arbitrage, $\mu + \delta = r$
- $\frac{dS_t}{S_t} = \mu dt + \sigma dW_t = (r - \delta)dt + \sigma dW_t$
- Risk neutral dynamics of an asset price with continuous dividend yield δ

- $S_t = S_0 e^{(r-\delta-\frac{1}{2}\sigma^2)t+\sigma W_t}$
  dividend yield reduced the growth rate of the underlying

Applications:

1. Equity Indecies
   Often model an index as a geometric brownian motion
   The index itself does not pay dividends, but the stocks that make up the index might
   There are a wide range of dividends on different dates
   Can approximate with continuous dividend yield

2. Exchange Rates
   S is an exchange rate
   A unit of foriegn currency earns interest at rate r, which can be viewed as a dividend stream
   To model S as geometric brownian motion, $\mu = r - r_f$

3. Commoditites
   Physical commodities like gold and oil
   Cost of storage acts as a negative dividend yield
   Also have the benefit of being able to sell or consume when there's a shortage, quantified with a convenience charge
   Net dividend yield is different between the two

Multiple Dimensions:

- Specify a multidimensional geometric brownian motion through the system
$$\frac{dS_i(t)}{S_i(t)} = \mu_i dt + \sigma_i dX_i(t) \tag{50}$$

- X$_i(t)$ is a standard one-dimensional brownian motion

- X_i and X_j have correlation ρ_{ij}

- Letting $\Sigma_{ij} = \sigma_i\sigma_j\rho_{ij}$ defines $\Sigma \in \mathbb{R}^{dxd}$

- $X(t) \sim BM(0,\Sigma)$ and $S \sim GBM(\mu,\Sigma)$
- Σ is the covariance matrix for X(t), not S, same for μ
- A $BM(0,\Sigma)$ can be represented as AW(t), where W(t) is a d-dimensional standard $BM(0,T)$ amd A is any matrix such that $AA^T = \Sigma$
- Apply to above to get $\frac{dS_i(t)}{S_i(t)} = \mu_i dt + a_i dW(t)$
- Explicitly $\frac{dS_i(t)}{S_i(t)} = \mu_i dt + \Sigma_{j=1}^{d} A_{ij} dW_j(t)$

Simulation:

- Solution is $S_i(t) = S_i(0)e^{(\mu_i - \frac{1}{2}\sigma_i^2)t + \sum_{j=1}^{d} A_{ij}W_j(t)}$
- Can use this form to simulate $GBM(\mu,\Sigma)$
- Simulate at $0 < t_0 < t_1 < ... < t_n$:
  $S_i(t_{k+1}) = S_i(t_k)e^{(\mu_i - \frac{1}{2}\sigma_i^2)(t_{k+1}-t_k) + \sqrt{t_{k+1}-t_k}\sum_{j=1}^{d} A_{ij}Z_{k+1,j}}$
  Choose A as the Cholesky factor of Σ and the number of computations are reduced
- If asset S_i has dividend yield δ_i, set $\mu_i = r - \delta\_i$

Application:

1. Spread Option
   A call option on the spread between two assets, S_1, S_2 with strike K and expiry T
   Payoff $= ([S_1(T) - S_2(T)] - K)^+$
   Example: crack spread: option on the spread between heating oil and crude oil futures
2. Basket Option
   An option on a portfolio of underlying assets
   Example: Payoff $= ([c_1S_1(T) + c_2S_2(T) + ... + c_dS_d(T)] - K)^+$
   Could be related assets such as currencies or stocks in the same economic sector
3. Outperformance Option
   Options on the maximum or minimum of mulitple assets
   Example: Payoff $= (\max[c_1S_1(T) + c_2S_2(T) + ... + c_dS_d(T)] - K)^+$

4. Barrier Option
   Can be knock-in or knock-out and there are many variations
   Example: Down-and-in put option on S_1 that knowck in when S_2 drops below the barrier
   Payoff $= 1_{\{\min S_2(t_i)<b\}}(K - S_1(T))^+$
   S_1 could be a stock and S_2 an index

5. Quantos
   Options that depend on both an underlying asset and an exchange rate
   Example: An option to buy a stock denominated in a foreign currency, with the strike price fixed in the foreign currency, but the payoff is to be made in the domestic currency.
   Payoff $= S_2(T)(S_1(T) - K)^+$
   where S_1 is the stock price and S_2 is the exchange rate

Variance Reduction Techniques:
Goal is to increase the efficiency of Monte Carlo Methods by reducing the variance of simulation estimates, which is often done by exploiting features or specific problems.

Control Variates:

- Exploit information about the error in estimates of known quantities to reduce the error in an estimate of an unknown quantity
- Let $Y_1, ..., Y_n$ be outputs of n runs of a simulation
  Y_i could be the discounted payoff of an option on the i^{th} simulated path
  Assume the Y_i are independent and identically distributed
- Want to estimate $E(Y_i)$
- Estimator: $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$
  This estimator is unbiased and converge to $E(Y_i)$ almost surely
- Suppose on each replication, another output X_i is calculated in addition to Y_i
- Assume the pairs (X_i, Y_i), i=1,...,n are independent and identically distributed and $E(X_i)$ is known

- For any fixed b, calculate $Y_i(b) = Y_i - b[X_i - E(X)]$ for the i^{th} replication

- Calculate the sample mean: $\bar{Y}(b) = \bar{Y} - b[\bar{X} - E(X)] = \frac{1}{n}\sum_{i=1}^{n}[Y_i - b[X_i - E(X)]]$
  Called the control variance estimator
  The observed error, $\bar{X} - E(X)$ is a control in estimating E(Y)

- $E(\bar{Y}(b)) = E[\bar{Y} - b(\bar{X} - E(X))] = E[Y] - b(E[\bar{X}] - E[X]) = E[Y]$
  Therefore $\bar{Y}(b)$ is an unbiased estimator of E[Y]

- As $n \rightarrow \infty$:
  $\lim \bar{Y}(b) = lim \frac{1}{n}\sum_{i=1}^{n}[Y_i - b(X_i - E[X])] = E[Y] - bE[X] + bE[X] = E[Y]$
  Therefore $\bar{Y}(b)$ is a consistency estimator of E[Y]

- $Var[Y_i(b)] = Var[Y_i - b(X_i - E[X])] = E(Y_i^2) - 2bE[Y_i(X_i - E[X])] + b^2E[(X_i - E[X])^2] - (E[Y_i])^2 = \sigma^2(b)$

- The control variate estimator $\bar{Y}(b)$ has variance:
  $Var[\bar{Y}(b)] = Var[\frac{1}{n}\sum_{i=1}^{n} Y_i(b)] = Var[Y_i(b)] = \frac{\sigma^2(b)}{n}$

- The sample mean, $\bar{Y}$, has variance $Var[\bar{Y}] = \frac{\sigma_Y^2}{n} = \frac{\sigma^2(0)}{n}$ (can choose b=0)

- Want a reduction in variance, $Var[\bar{Y}(b)] < Var[\bar{Y}]$

- This holds if $\frac{\sigma^2(b)}{n} < \frac{\sigma_Y^2}{n}$
  if and only if $b^2\sigma_X^2 < 2b\sigma_X\sigma_Y\rho_{XY}$

- To minimize $\sigma^2(b)$, $\frac{d[\sigma^2(b)]}{db} = 0 = -2\sigma_X\sigma_Y\rho_XY + 2b\sigma_X^2$
  Therefore $b = \frac{\sigma_X\sigma_Y\rho_{XY}}{\sigma_X^2} = \frac{Cov(X,Y)}{Var[X]} = b^*$

- $\sigma^2(b^*) = \rho_Y^2 - 2\frac{\sigma_Y\rho_{XY}}{\sigma_X}\sigma_X\sigma_Y\rho_{XY} + (\frac{\sigma_Y\rho_{XY}}{\sigma_X})^2\sigma_X^2 = \sigma_Y^2(1 - \rho_{XY}^2)$

- Compute the ratio of the optimally controlled estimator to that of the uncontrolled estimator:
  $\frac{\sigma^2(b^*)}{\sigma^2(0)} = \frac{\sigma_Y^2(1-\rho_{XY}^2)}{\sigma_Y^2} = 1 - \rho_{XY}^2$

Notes:

- The strength of the correlation between X and Y determines the effectiveness of the control variate
- The variance reduction factor $= \frac{1}{1-\rho_{XY}^2}$
- By using the control variate, this is the variance reduction

Examples:

- If $\rho_{XY} = .95$, then there's a ten-fold reduction
- If $\rho_{XY} = .9$, then there's a five-fold reduction
- If $\rho_{XY} = .7$, then there's a two-fold reduction
- Strong correlation is required to get A benefit
- Often σ_Y and ρ_{XY} are unknown, so estimate B^*. If the parameters are replaced with their sample counterparts, gives: $\hat{b}_n = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y-i-\bar{Y})}{\sum_{i=1}^{n}(X_i-\bar{X})^2}$
- Multiply top and bottom by $\frac{1}{n}$, strong law of large numbers implies $\hat{b}_n \to b^*$ almost surely
- Can use $\bar{Y}(\hat{b}_n)$ as an estimator, $Y_i(\hat{b}_n) = Y_i - \hat{b}_n(X_i - E[X])$, which adds a little bias
- $\hat{b}_n$ is the slope of the least squares regression line, through the points (X_i, Y_i)
- The control variate estimate $\bar{Y}(\hat{b}_n)$ is the value fitten by the line at E[X]

Example: Underlying Assets

- The absence of arbitrage is equivalent to the requirement that discounted asset prices are martingales under the risk-neutral measure.
- If r is the constant interest rate, $e^{-rt}S_t$ is a martingale.
- Given S_0, by above, if S_t is adapted to the filtration $\mathcal{F}_{tt\geq 0}$, then $E[e^{-rt}S_t] = E[e^{-rt}S_t|\mathcal{F}_0] = S_0$
- Suppose want to price an option on S with discounted payoff Y
- Assume Y is a function of the price path
- Form independent price path replications $S_1, ..., S_n$ over $[0, T]$ of S and form the control variate estimator $\frac{1}{n}\sum_{i=1}^{n}[Y_i - b(S_i(T) - e^{rt}S(0))]$
- If pricing a call, $Y = e^{-rt}(S_T - K)^+$
- The correlation between T and S_T depends on K
  If K=0, $\rho = 1$
  If K is large and option is deep out of the money, ρ is small

Appendix

HW 1 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday January 27, 2009

You must compose your assignments independently; however, you may discuss your work with one another at the rough level.

1. (10 Points) Suppose you hold two European call options on the same underlying asset with the same expiry T. The underlying asset has time-t price S_t. The strike prices of the opitions are K_1 and K_2 where $K_1 < K_2$. Carefully sketch the payoff diagram of your portfolio. Make sure to label your diagram and indicate the slopes of the different linear parts of the graph.

2. (10 Points)Consider two European put options on the same underlying asset with the same expiry T. The underlying asset has time-t price S_t. The strike prices of the opitions are K_1 and K_2 where $K_1 < K_2$. Suppose you buy one put option with strike K_2 and you write one put option with strike K_1. Carefully sketch the payoff diagram of your portfolio. This position is called a *Bear Spread*. Why is it given this title?

3. (30 Points) Assuming W_t is a standard Brownian motion, use Ito's formula to derive stochastic differential equations for the following processes:

   (a) $X_t = e^{\frac{1}{2}t}\sin(W_t)$

   (b) $Y_t = e^{\frac{1}{2}t}\cos(W_t)$

   (c) $Z_t = (1 + \frac{1}{3}W_t)^3$. In this case your solution should be simplified to be in terms of Z_t.

HW 1 Solutions - Math 573, Marcel Blais, Spring 2009

1. Payoff = $\begin{array}{ll} y = 0 & , S_T < K_1 < K_2 \\ y = S_T - K_1 & , K_1 < S_T < K_2 \\ y = 2S_T - K_1 - K_2 & , K_1 < K_2 < S_T \end{array}$

2. Payoff = $\begin{array}{ll} y = K_2 - K_1 & , S_T < K_1 < K_2 \\ y = K_2 - S_T & , K_1 < S_T < K_2 \\ y = 0 & , K_1 < K_2 < S_T \end{array}$

   This option is used to hedge risk in a bear market (when the market is going down), hence the name.

3. (a) $X_t = e^{1/2t}\sin(W_t)$
   We set $g(t,x) = e^{1/2t}\sin(x)$. Thus $g_t = \frac{1}{2}g$, $g_x = e^{1/2t}\cos(x)$, and $g_{xx} = -g$.
   Using Ito's formula, we get
   $dX_t = \frac{1}{2}gdt + e^{1/2t}\cos(W_t)dW_t + \frac{1}{2}(-g)dt$
   and $dX_t = e^{1/2t}\cos(W_t)dW_t$

   (b) $Y_t = e^{1/2t}\cos(W_t)$
   We set $g(t,x) = e^{1/2t}\cos(x)$. Thus $g_t = \frac{1}{2}g$, $g_x = -e^{1/2t}\sin(x)$, and $g_{xx} = -g$.
   Using Ito's formula, we get
   $dX_t = \frac{1}{2}gdt - e^{1/2t}\sin(W_t)dW_t + \frac{1}{2}(-g)dt$
   and $dX_t = -e^{1/2t}\sin(W_t)dW_t$

   (c) $Z_t = (1 + \frac{1}{3}W_t)^3$. In this case your solution should be simplified to be in terms of Z_t.
   We set $g(t,x) = (1 + \frac{1}{3}x)^3$. Thus $g_t = 0$, $g_x = (1 + \frac{1}{3}x)^2 = g^{2/3}$, $g_{xx} = \frac{2}{3}(1 + \frac{1}{3}x) = \frac{2}{3}g^{1/3}$.

   Using Ito's formula gives
   $dZ_t = 0dt + g^{2/3}dW_t + \frac{1}{3}g^{1/3}dt$
   and $dZ_t = \frac{1}{3}Z_t^{1/3}dt + Z_t^{2/3}dW_t$

HW 2 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday February 3, 2009

You must compose your assignments independently; however, you may discuss your work with one another at the rough level.

1. (10 Points) Suppose a stock price is governed by $\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$. Consider a European digital call option written on this stock with strike price K and maturity time T. At time T, this option pays \$1 if $S_T > K$ and \$0 if $S_T \leq K$. What is the boundary value problem that determines the time-t value of this option?

2. (a) (5 Points) Show that

$$\lim_{h \to 0} \frac{4u(x+h,t) - u(x+2h,t) - 3u(x,t)}{2h} \tag{51}$$

is a valid definition of $\frac{\partial u}{\partial x}$.

(b) (10 Points) Show that if we use

$$\frac{\partial u}{\partial x} \approx \frac{4u(x+\Delta x,t) - u(x+2\Delta x,t) - 3u(x,t)}{2\Delta x} \tag{52}$$

that the error of our approximation is $O([\Delta x]^2)$.

Hint: Use Taylor series to show that

$$\frac{\partial u}{\partial x} = \frac{4u(x+\Delta x,t) - u(x+2\Delta x,t) - 3u(x,t)}{2\Delta x} + c_2\Delta x^2 + c_3\Delta x^3 + c_4\Delta x^4 \dots. \tag{53}$$

3. (10 points) Consider the approximation

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+2\Delta x,t) - 2u(x+\Delta x,t) + u(x,t)}{(\Delta x)^2} \tag{54}$$

.

Determine the order of error of this approximation. Is this approximation to $\frac{\partial^2 u}{\partial x^2}$ more or less accurate than the approximation we derived in class? Why?

HW 2 Solutions - Math 573, Marcel Blais, Spring 2009

1. We denote the option value by V(S,t). The boundary value problem consists of

   - V must satisfy the Black-Scholes-Merton PDE,

   $$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV \tag{55}$$

   - The final condition is given by the payoff,

   $$V(S,t) = \begin{array}{ll} 1 & if S_T \geq K \\ 0 & if S_T \leq K \end{array} \tag{56}$$

   - If the stock price is zero, it never escapes from zero, and thus the option will have a payoff of zero at maturity. We thus set

   $$V(0,t) = 0 \tag{57}$$

   - As $S \to \infty$, it becomes more and more likely that the option will be exercised. Thus we set

   $$V \to 1^- \text{ as } S \to \infty \tag{58}$$

2. (a) $\lim_{h\to 0} \frac{4u(x+h,t)-u(x+2h,t)-3u(x,t)}{2h}$
   $= lim_{h\to 0}[4u_x(x+h,t) - u_x(x+2h,t) - 3u_x(x,t)\cdot 0]\cdot\frac{1}{2}$
   $= 2u_x(lim_{h\to 0}[x+h],t) - u_x(lim_{h\to 0}[x+2h],t)$
   $= 2u_x(x,t) - u_x(x,t) = u_x(x,t)$

   (b) We use the Taylor series expansions

   $$\begin{array}{c} u(x+h) = u(x) + u'(x)h + \frac{1}{2}u''(x)h^2 + \frac{1}{3!}u'''(x)h^3 + ... \\ u(x+2h) = u(x) + u'(x)2h + \frac{1}{2}u''(x)(2h)^2 + \frac{1}{3!}u'''(x)(2h)^3 + ... \end{array} \tag{59}$$

   Setting $h = \Delta x$ in above and supressing the notation for dependency on t gives

   $$\frac{4u(x+\Delta x,t) - u(x+2\Delta x,t) - 3u(x,t)}{2\Delta x} = u'(x) - \frac{2}{3}u'''(x)(\Delta x)^2 + \mathcal{O}([\Delta x]^3) \tag{60}$$

   Thus our approximation to $\frac{\partial u}{\partial x}$ has an $\mathcal{O}([\Delta x]^2)$ error.

3. Setting $h = \Delta x$ in (59) and supressing the notation for dependency on t gives

$$\frac{(x+2\Delta x,t) - 2u(x+\Delta x,t) + u(x,t)}{(\Delta x)^2} = u''(x)+u'''(x)(\Delta x)+\mathcal{O}([\Delta x]^2) \tag{61}$$

Thus the error of our approximation is $\mathcal{O}(\Delta x)$. This error is larger than the error of the approximation that we used in lecture.

HW 3 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday February 10, 2009

You must compose your assignments independently; however, you may discuss your work with one another at the rough level.

1. (20 Points) The Black-Scholes-Merton partial differential equation can be written
$$\frac{\partial u}{\partial t} = r\frac{\partial u}{\partial x}x + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2}x^2 - ru. \tag{62}$$
Using the spatial finite differences
$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O(\Delta x^2) \tag{63}$$
and
$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + O(\Delta x^2) \tag{64}$$
discretize (62) in x to form a system of ordinary differential equations. Express this system of equations in martix-vector form.

2. (10 Points) Find the eigenvalues and corresponding eigenvectors of the matrix
$$M = \begin{bmatrix} 2 & 0 & 0 \\ 1 & -1 & -2 \\ -1 & 0 & 1 \end{bmatrix}.$$

3. (10 Points) Prove that if $A \in \mathbb{R}^{n\times n}$, then AA^T and A^TA have the same eigenvalues.

HW 3 Solutions - Math 573, Marcel Blais, Spring 2009

1. The resulting linear system is

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u} + \epsilon \tag{65}$$

where $\mathbf{u} = [u_0, u_1, \ldots, u_n]^T$,

$$A = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 & \ldots & 0 \\ \beta_1 & -\gamma_1 & \alpha_1 & 0 & 0 & \ldots & 0 \\ 0 & \beta_2 & -\gamma_2 & \alpha_2 & 0 & \ldots & 0 \\ 0 & 0 & \beta_3 & -\gamma_3 & \alpha_3 & \ldots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & 0 & \ldots & 0 & \beta_{n-1} & -\gamma_{n-1} & \alpha_{n-1} \\ 0 & 0 & \ldots & 0 & 1 & -2 & 1 \end{bmatrix} \tag{66}$$

and $\alpha_i = \frac{1}{2}[\frac{r}{\Delta x}x_i + \frac{\sigma^2}{(\Delta x)^2}x_i^2]$,

$\beta_i = \frac{1}{2}[-\frac{r}{\Delta x}x_i + \frac{\sigma^2}{(\Delta x)^2}x_i^2]$,

$\gamma_i = r + \frac{\sigma^2}{(\Delta x)^2}x_i^2$.

2. The characteristic polynomial for M is

$$(\lambda - 2)(\lambda - 1)(\lambda + 1). \tag{67}$$

The eigenvalue-eigenvector pairs are

$$\left(-1, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\right), \left(1, \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}\right), \text{ and } \left(2, \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\right).$$

3. Suppose λ is an eigenvalue of AA^T. Then for some vector $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{x} \neq \mathbf{0}$,
$$(AA^T - \lambda I)\mathbf{x} = \mathbf{0}. \tag{68}$$
We can multiply both sides of (68) by A^T and use the fact that $A^T I = IA^T$ to get
$$(A^T AA^T - \lambda IA^T)\mathbf{x} = \mathbf{0}. \tag{69}$$
We factor A^T out of the expression on the right hand side to get
$$(A^T A - \lambda I)(A^T\mathbf{x}) = \mathbf{0}. \tag{70}$$
Note that $A^T\mathbf{x} \in \mathbb{R}^n$. We have two cases:

   - If $\lambda \neq 0$, by (68) we have
   $$AA^T\mathbf{x} = \lambda\mathbf{x} \neq \mathbf{0}. \tag{71}$$
   If $A^T\mathbf{x} = \mathbf{0}$, then (71) would not hold. Thus $A^T\mathbf{x} \neq \mathbf{0}$, and $A^T\mathbf{x} \in \mathbb{R}^n$ is an eigenvector for $A^T A$ corresponding to the eigenvalue λ by (70).
   - If $\lambda = 0$, then AA^T is singular. This means that A and A^T are both singular, and thus $A^T A$ is also singular. A singular matirx has $\lambda = 0$ as an eigenvalue, thus $\lambda = 0$ is an eigenvalue of $A^T A$.

   In both cases we see that λ is an eigenvalue for $A^T A$.

   We repeat this argument for an eigenvalue λ of $A^T A$ and conclude that λ is also an eigenvalue for AA^T. This shows that $A^T A$ and AA^T have the same eigenvalues.

HW 4 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday February 17, 2009

You must compose your assignments independently; however, you may discuss your work with one another at the rough level.

1. (25 Points) Show that the Crank-Nicolson finite difference scheme for

$$\frac{dv}{dt} = \lambda v \tag{72}$$

is the only possible second-order accurate scheme that can result from a weighted average of the implicit Euler method and the explicit Euler method.

Can you find a weighted average that gives you a first-order accurate finite difference scheme that is different from the Crank-Nicolson scheme?

Hint: Consider the average

$$\theta \frac{dv}{dt}|^{n} + (1-\theta)\frac{dv}{dt}|^{n+1} \tag{73}$$

for $\theta \in [0,1]$.

2. (10 Points) Using Matlab find[1] a factorization of the 6×6 matrix A

$$A = \begin{bmatrix} -25 & -2 & 14 & 0 & 0 & 0 \\ -2 & -22 & 5 & -8 & 0 & 0 \\ 14 & 5 & -54 & 16 & 7 & 0 \\ 0 & -8 & 16 & -26 & 10 & -15 \\ 0 & 0 & 7 & 10 & -20 & -1 \\ 0 & 0 & 0 & -15 & -1 & -35 \end{bmatrix} \tag{74}$$

such that $A = XLX^{-1}$. L and X should have the form

$$L = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_6 \end{bmatrix}, \tag{75}$$

$$X = \left[\begin{array}{c|c|c|c|c|c|} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} & \mathbf{v_6} \end{array} \right] \tag{76}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6$ are the eigenvalues of A, and the columns of X are the corresponding eigenvectors $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_4}, \mathbf{v_5}, \mathbf{v_6}$. Make sure to check that your solution is correct.

Your complete solution should be coded in a script file (all commands should be saved in the m-file) called *number2.m*. For this problem submit the following

- A printout of your m-file,
- an electronic version of your m-file sent to me as an attachment in an email, and
- a printout of a test run of your m-file.

[1]Hint: Check out the Matlab function "eig".

3. (15 Points) Consider the PDE,

$$\frac{\partial u}{\partial t} = r\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2} - ru. \tag{77}$$

Write a Matlab function called *spatialCoeffs* that takes inputs

- the spatial step Δx,
- the number of space grid points N,
- the volatility σ,
- the interest rate r,

and returns the matrix A that results from the spatial discretization of (77) that we covered in lecture 3.

Note that since this is a function, it should neither prompt the user nor print any output. The top line of your m-file *spatialCoeffs.m* should be

function[A] = spatialCoeffs(deltaX,r,N,sigma). (78)

You function returns the resulting matrix as an argument. You can thus use your function at the Matlab prompt in the following manner

>> "myMatrix = spatialCoeffs(.01,.05,100,2)". (79)

Your complete solution should be coded in an m-file called *spatialCoeffs.m.* For this problem submit the following

- A printout of your m-file,
- an electronic version of your m-file sent to me sent to me as an attachment in an email, and
- a printout of two test runs of your m-file.

HW 4 Solutions - Math 573, Marcel Blais, Spring 2009

1. Using the hint, set

$$\theta\frac{dv}{dt}|^{n} + (1-\theta)\frac{dv}{dt}|^{n+1} = \frac{v^{n+1}-v^{n}}{\Delta t} \tag{80}$$

We now use the ODE to set $\frac{dv}{dt}|^{n} = \lambda v^{n}$, and $\frac{dv}{dt}|^{n+1} = \lambda v^{n+1}$. Substitute into above to get

$$\theta\lambda\Delta t \cdot v^{n} + (1-\theta)\lambda\Delta t \cdot v^{n+1} = v^{n+1} - v^{n} \tag{81}$$

We rewrite this as $[1-(1-\theta)\lambda\Delta t]v^{n+1} - [1+\theta\lambda\Delta t]v^{n} = 0$. Introducing the shift operator E, this becomes

$$[1-(1-\theta)\lambda\Delta t]Ev^{n} - [1+\theta\lambda\Delta t]v^{n} = 0 \tag{82}$$

We define P(E) so that above becomes $P(E)v^{n}$ by

$$P(E) = [1-(1-\theta)\lambda\Delta t]E - [1+\theta\lambda\Delta t] \tag{83}$$

Solving $P(\Lambda)=0$ gives us the amplification error of our finite difference scheme,

$$\Lambda = \frac{1+\theta\lambda\Delta t}{1-(1-\theta)\lambda\Delta t} \tag{84}$$

We now consider the function $f(x) = \frac{1}{1-(1-\theta)x}$. We want to write this as a power series and then substitute $x=\lambda\Delta t$ to build a series representation for Λ. Considering f to be the sum of a geometric series with a=1 and $r=(1-\theta)x$ gives

$$f(x) = \sum_{k=0}^{\infty}[(1-\theta)x]^{k} \tag{85}$$

We observe that

$$f(x)(1+\theta x) = \sum_{k=0}^{\infty}([(1-\theta)x]^{k} + \theta x[(1-\theta)x]^{k}) \tag{86}$$

This simplifies to

$$f(x)(1+\theta x) = 1 + x + (1-\theta)x^{2} + \mathcal{O}(x^{3}) \tag{87}$$

The Maclaurin series expansion of e^x is

$$e^x = 1 + x + \frac{1}{2}x^2 + \mathcal{O}(x^3) \tag{88}$$

Noting that $\Lambda = f(\lambda\Delta t)(1 + \theta\lambda\Delta t)$ and that $e^{\lambda\Delta t}$ can be obtained from the above equation, we compare the two series representations to detect the order of accuracy of our method. We see that regardless of our choice of θ, our method is first-order accurate because the constant and x terms match. This is consistent with the results that the implicit Euler method ($\theta = 0$) and the exlicit Euler method ($\theta = 1$) are first-order accurate. However, we see that in order to have the x^2 terms match, we require $\theta = \frac{1}{2}$. This is the only way to get second-order accuracy with this scheme.

2. Matlab Code:
% factor the given A (a 6x6) matrix such that $A = XLX^-1$
% L is a diagonal matrix formed by the eigenvalues of A
% X are the corresponding eigenvectors of A
A=$\begin{bmatrix} -25 & -2 & 14 & 0 & 0 & 0 \\ -2 & -22 & 5 & -8 & 0 & 0 \\ 14 & 5 & -54 & 16 & 7 & 0 \\ 0 & -8 & 16 & -26 & 10 & -15 \\ 0 & 0 & 7 & 10 & -20 & -1 \\ 0 & 0 & 0 & -15 & -1 & -35 \end{bmatrix}$
[X,L]=eig(A)
$B = X * L * inv(X)$
% Can see that A=B

3. Matlab Code:

```
% Takes imputs deltaX, r, N, and sigma to return alpha, gamma, beta
of the
% matrix A as well as the matrix A that results from spatial discretiza-
tion
% of the PDE.

function[A]=spatialCoeffs(deltaX,r,N,sigma)

alpha=(r/(2*deltaX))+((sigma^2)/(2*(deltaX^2)))
gamma=((sigma^2)/(deltaX^2))+r
beta=(-r/(2*deltaX))+((sigma^2)/(2*(deltaX^2)))
B=zeros(N);
c=(-gamma).*ones(N,1);
d=(alpha).*ones(N-1,1);
f=(beta).*ones(N-1,1);
A=diag(c)+diag(d,1)+diag(f,-1);
A(1,1)=1;
A(1,2)=-2;
A(1,3)=1;
A(N,N-2)=1;
A(N,N-1)=-2;
A(N,N)=1;
A;
```

HW 5 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday February 24, 2009

This homework counts as $\frac{1}{3}$ of a regular homework.

You must compose your assignments independently; however, you may discuss your work with one another at the rough level.

1. Consider a European put option with time t price $P(S, t)$, strike price K, and maturity T. Suppose that that interest rate r is constant. The spatial boundary conditions for such an option are

   - $P(S, t) \to 0$ as $S \to \infty$
   - $P(0, t) = Ke^{-r(T-t)}$

   If we are pricing this option using finite difference methods, how should we implement these boundary conditions?

   Hint: Yes, this is a really easy problem.

HW 5 Solutions - Math 573, Marcel Blais, Spring 2009

1. We implement these boundary conditions in the following manner:

   - For the first condition, $P(S,t) \to 0$ as $S \to \infty$, we set S_{max} to be a large positive value. We then set $P(S_{max},t) = 0$ for all t.
   - For the second condition, $P(0,t) = Ke^{-r(T-t)}$, we set S_{min} to be a small value, usually $S_{min} = 0$. We then set $P(S_{min},t) = Ke^{-r(T-t)}$.

HW 6 - Math 573, Marcel Blais, Spring 2009

Due by 5pm on Friday March 27, 2009

1. Using the Monte Carlo methods we covered in class on March 17, write a Matlab program to calculate

$$\int_0^1 x^2 \, dx. \tag{89}$$

   Your program should be written as a function *integralMC* saved in the file *integralMC.m*. It should take n as an input and return both the estimate of the integral and the error of the approximation. For this problem submit the following

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of two test runs of your m-file using different values for n.

2. Using the Monte Carlo methods we covered in class on March 17, write a Matlab program that calculates the time zero no-arbitrage value of a European call option and a European put option with strike price K, underlying asset price S_t, and maturity T. Your program should be written in the m-file *euroOptionMC.m*. It should take inputs

   - the number of draws n,
   - the initial underlying asset price S_0,
   - the interest rate r, and
   - the volatiliy of the underlying σ.

   Your program should output the estimated prices of the put and call, along with 95% and 99% confidence intervals for each estimate. For this problem submit the following

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of several test runs of your m-file using different values for n.

HW 6 Solutions - Math 573, Marcel Blais, Spring 2009

1. Matlab Code:

```
%Using the Monte Carlo methods calculate the integral from 0 to 1 of
x^2 dx
%takes n as an input and returns the estimation of the integral and the
%error of approximation

function[alpha_hat]=integralMC(n);

f=zeros(n,1);
for ii=1:n
f(ii,1)=(rand(1));
end
f;

sum_old=0;
for ii=1:n
sum_new=(f(ii,1)^2)+sum_old;
sum_old=sum_new;
end

sum=sum_new;
alpha_hat=(1/n)*sum;
errSum_old=0;
for ii=1:n
errSum_new=(f(ii,1)-alpha_hat)^2+errSum_old;
errSum_old=errSum_new;
end

errSum=errSum_new;
S_f=sqrt(errSum/(n-1));
errorAprox=S_f/(sqrt(n))
```

2. Matlab Code:

```
%Using the Monte Carlo mthods calculate the time zero no-arbitrage
value of a European call option and a European put option with strike
price K, underlying asset price S_t, and maturity T. Ouputs the prices
of the put and call along with 95% and 99% confidenceintervals

n=10; %number of draws
S_0=5; %initial underlying asset price
r=.05; %interest rate
sigma=.001; %volatility of the underlying
T=6; %maturity
K=10; %strike price

c=zeros(n,1);
for ii=1:n
Z=randn(1);
S_T=(S_0)*exp((r-(.5*(sigma^2)))*T+(sigma*sqrt(T)*Z));
c(ii,1)=exp(-r*T)*max(S_T-K,0);
end
c;
callSum=0;
for ii=1:n
callSum_new=c(ii,1)+callSum;
callSum=callSum_new;
end
callSum;
callPrice_hat=callSum/n
cError=0;
for ii=1:n
cSum_new=(c(ii,1)-callPrice_hat)^2+cError;
cError=cSum_new;
end
cError;
S_c=sqrt(cError/(n-1));

disp( ['A 95% confidence interval for the call option price is
(' , num2str(callPrice_hat-(1.96*(S_c/sqrt(n)))), ', ' ,
num2str(callPrice_hat+(1.96*(S_c/sqrt(n)))) , ').' ] )
disp( ['A 99% confidence interval for the call option price is
(' , num2str(callPrice_hat-(2.575*(S_c/sqrt(n)))), ', ' ,
num2str(callPrice_hat+(2.575*(S_c/sqrt(n)))) , ').' ] )

p=zeros(n,1);
for ii=1:n
Z=randn(1);
S_T=(S_0)*exp((r-(.5*(sigma^2)))*T+(sigma*sqrt(T)*Z));
p(ii,1)=exp(-r*T)*max(K-S_T,0);
end
p;
putSum=0;
for ii=1:n
putSum_new=p(ii,1)+putSum;
putSum=putSum_new;
end
putSum;
putPrice_hat=putSum/n
pError=0;
for ii=1:n
pSum_new=(p(ii,1)-putPrice_hat)^2+pError;
pError=pSum_new;
end
pError;
S_p=sqrt(pError/(n-1));

disp( ['A 95% confidence interval for the put option price is
(' , num2str(putPrice_hat-(1.96*(S_p/sqrt(n)))), ', ' ,
num2str(putPrice_hat+(1.96*(S_p/sqrt(n)))) , ').' ] )
disp( ['A 99% confidence interval for the put option price is
(' , num2str(putPrice_hat-(2.575*(S_p/sqrt(n)))), ', ' ,
num2str(putPrice_hat+(2.575*(S_p/sqrt(n)))) , ').' ] )
```

HW 7 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday March 31, 2009

1. (10 Points) Write a Matlab function the generates uniformly distributed pseudorandom numbers in $[0, 1]$. Use a linear congruential generator

$$\begin{aligned} x_{i+1} &= (a \cdot x_i + c) \mod m \\ u_{i+1} &= \tfrac{x_{i+1}}{m} \end{aligned} \tag{90}$$

Your program should be written as a function *linConGenerator* saved in the file *linConGenerator.m.* It should take modulus m, multiplier a, c, seed x_0, and N as inputs. It should return a vector of length N containing the pseudorandom values.

For the pure case ($c = 0$), you program should check that the generator has full period. If it does not have full period, your program should print a warning to the user before returning its outputs.

For this problem submit the following

- A printout of your m-file,
- an electronic version of your m-file sent to me as an attachment in an email, and
- a printout of a few test runs of your m-file using different inputs.

2. (10 Points) The Burr distribution with parameters $c, k > 0$ has cumulative distribution function

$$F(x) = 1 - (1 + x^c)^{-k} \tag{91}$$

and density function

$$f(x) = ck\frac{x^{c-1}}{(1 + x^c)^{k+1}} \tag{92}$$

for $x > 0$.

Using the inverse transform method, give a formula for generating samples of the Burr distribution from independent uniform random variables on $[0, 1]$, $U_1, U_2, U_3, \ldots$

3. (10 Points) Write a Matlab function the generates non-negative pseudorandom samples from the exponential distribution with parameter θ.

Your program should be written as a function *exponentialGenerator* saved in the file *exponentialGenerator.m*. It should take N and θ as inputs. It should return a vector of length N containing the pseudorandom values.

You can use Matlab's *rand* function or your linear congruential generator from the first problem.

For this problem submit the following

- A printout of your m-file,
- an electronic version of your m-file sent to me as an attachment in an email, and
- a printout of a few test runs of your m-file using different inputs.

HW 7 Solutions - Math 573, Marcel Blais, Spring 2009

1. Matlab Code:

```
%Generates uniformly distributed pseudorandom numbers in [0,1]
%takes inputs m,a,c,x_0, and N
%returns vector of length N containing pseudorandom values

function[u]=linConGenerator(m,a,c,x_0,N);
x=zeros(N,1);
x_old=x_0;
for ii=1:N
x_new=mod((a*x_old+c),m);
x(ii,1)=x_new;
x_old=x_new;
end
x;
u=zeros(N,1);
for ii=1:N
u(ii,1)=x(ii,1)/m;
end
u;
p=zeros(5,1);
if c==0
p(1,1)=0;
else p(1,1)=1;
end
k=1;
for jj=1:m-2
if mod((a^jj)-1,m)==0;
k=0;
end
end
if k==0
p(2,1)=1;
else p(2,1)=0;
end
if mod(((a^(m-1))-1),m)==0
```

```
p(3,1)=0;
else p(3,1)=1;
end
if x_0==0
p(4,1)=1;
else p(4,1)=0;
end
if isprime(m)
p(5,1)=0;
else p(5,1)=1;
end
if c==0
if p==zeros(5,1)
disp('generator has full period')
else disp('generator does not have a full period')
end
end
u;
```

2. Invert F(x)
   $U = 1 - (1 + x^c)^{-k}$
   $U - 1 = -(1 + x^c)^{-k}$
   $1 - U = (1 + x^c)^{-k}$
   $(1 - U)^{1/k} = 1 + x^c$
   $((1 - U)^{1/k} - 1)^{1/c} = x$
   If U is from Unifrom[0,1], then (1-U) is also from Uniform[0,1], so can use $((U)^{1/k} - 1)^{1/c} = x$

3. Matlab Code:
   %Generate non-negative pseudorandom samples from an exponential distribution with mean theta. Input N and theta.

```
function[x]=exponentialGenerator(N,theta)

u=zeros(N,1);
for ii=1:N
u(ii,1)=rand(1);
end
u;

x=zeros(N,1);

if theta¿0
for ii=1:N
x(ii,1)=(-1)*theta*log(u(ii,1));
end
else
for ii=1:N
x(ii,1)=theta*log(u(ii,1));
end
end
x;
```

HW 8 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday April 7, 2009

1. Prove the following:
   (a) (10 Points) Suppose that $\Sigma \in \mathbb{R}^{d\times d}$ is positive definite. Show that Σ is invertible.
   (b) (10 Points) Suppose that $\Sigma \in \mathbb{R}^{d\times d}$ is positive semidefinite but not positive definite. Show that Σ is singular.
   (c) (10 Points) Suppose that $A \in \mathbb{R}^{d\times d}$ is invertible. Show that $A^T A$ is positive definite.

2. (a) (15 Points) Suppose X is a random variable on $[0,1]$ with density function $f(x) = 20x(1-x)^3$. Using the uniform distribution on $[0,1]$ and the acceptance-rejection method, give an algorithm for generating samples from X.
   (b) (10 Points) Write a Matlab function that generates N pseudorandom samples of X. Your program should be written as a function *HW8* saved in the file *HW8.m*. It should take N as an input, and it should return a vector of length N containing the pseudorandom values. For this problem submit the following
      - A printout of your m-file,
      - an electronic version of your m-file sent to me as an attachment in an email, and
      - a printout of a few test runs of your m-file using different inputs.

3. (15 Points) Write a Matlab function that generates N samples from a normally distributed random variable with mean μ and variance σ^2 using the Box-Muller method. Your program should be written as a function *boxMuller* saved in the file *boxMuller.m*. It should take N, μ, and σ as inputs, and it should return a vector of length N containing the pseudorandom values. For this problem submit the following

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of a few test runs of your m-file using different inputs.

4. (20 Points) Write a Matlab function that generates a sample from a bivariate normally distributed random variable with mean $[\mu_1, \mu_2]^T$ and symmetric positive definite covariance matrix Σ. Your program should be written as a function *bivariateNormal* saved in the file *bivariateNormal.m*. It should take $[\mu_1, \mu_2]^T$ and Σ as inputs, and it should return a vector of containing the pseudorandom values.

   Make sure that your program checks that Σ is symmetric and positive definite.[2] Use your Box-Muller program to generate the standard normals in this program.

   For this problem submit the following

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of a few test runs of your m-file using different inputs.

[2]Hint: A 2×2 matrix is positive definite if its top left entry is positive and its determinant is positive.

HW 8 Solutions - Math 573, Marcel Blais, Spring 2009

1. (a) By definition, for $x \in \mathbb{R}^{d\times d}$ with $x \neq 0$, $x^T\Sigma x > 0$. Thus $x^T(\Sigma x) > 0$ and $\Sigma x \neq 0$. This means that the nullspace of Σ consists of only 0, which is equivalent to Σ being non-singular.

   (b) For positive semidefinite $\Sigma \in \mathbb{R}^{d\times d}$, we know there exists a lower-triangular Cholesky factor A such that $\Sigma = AA^T$. From our algorithm in class, since Σ is not positive definite we know that this factor A has at least one zero diagonal entry. Since the determinant of a lower-triangular matrix is the product of its diagonal entries, det(A)=0, and thus
   det(Σ)=det(AA^T)=det(A)·det(A^T)=0·0=0.
   This means that Σ is singular.

   (c) Let $x \in \mathbb{R}^d$ with $x \neq 0$.
   $x^TAA^Tx = (x^TA)(A^Tx) = (A^Tx)^T(A^Tx) = ||A^Tx||_2^2$.
   Since A is non-singular, $Ax \neq 0$ and thus $||A^Tx||_2^2 > 0$, which means that AA^T is positive definite.

2. (a) First we find the maximum value of f(x) on [0,1]. Computing f'(x) and solving f'(x*)=0 gives x*=$\frac{1}{4}$, and $f(\frac{1}{4}) = \frac{135}{64}$, which is the maximum value attained by f on [0,1]. We choose $c = \frac{135}{64}$. Our target distribution has density f, and the uniform distribution on [0,1] has density g(x)=1. Our algorithm is as follows:

   - Generate U_1, U_2 independent from Uniform[0,1]
   - If $U_1 \leq \frac{f(U_2)}{c \cdot g(U_2)} = \frac{64}{135} \cdot 20U_2(1-U_2)^3$, accept U_1
   - If $U_1 > \frac{64}{135} \cdot 20U_2(1-U_2)^3$, reject U_1 and return to the first step

   (b) Matlab Code:

```
% Generates N random samples from the distribution with pdf
% f(x) = 20x(1 − x)^3 using the acceptance-rejection method
% Input N, Number of samples to be generated.

function[X] = HW8(N)

X = zeros(N,1);
accept_sample = false;
for ii = 1:N
accept_sample = false;
while (∼ accept_sample)
x = rand();
u = rand();
if (x ¡= (256/27)*u*(1 − u)^3),
accept_sample = true;
end
end
X(ii) = x;
end
```

3. Matlab Code:

```
function[Z] = boxMuller(mu,sigma,N)
% Generates N samples from a N(mu,sigma^2) distribution using the Box-Muller method
%Input mu, the mean of the normal random sample, sigma, the standard deviation of the standard normal, and N, the numer of samples

function[Z] = boxMuller(mu,sigma,N)

Z = zeros(N,1);

for ii=1:N
u1 = rand();
u2 = rand();
R = -2*log(u1);
V = 2*pi*u2;
z = sqrt(R)*cos(V);
Z(ii) = mu + sigma*z;
end
```

4. Matlab Code:

```
%Generates a bivariate normal random sample w distribution N(mu,Sigma)
%Inputs: mu, mean vector (2x1), Sigma, covariance matrix (2x2). It
must be symmetric positive definite
%Generate two independent standard normals using the Box Muller
method.

function[X] = bivariateNormal(mu, Sigma)

z1 = boxMuller(0,1,1);
z2 = boxMuller(0,1,1);
Z = [z1;z2];

[muRows,muCols] = size(mu);
if (muRows == 1) & (muCols == 2)
mu = mu';
elseif muRows + muCols ~= 3
error('Input vector mu must have two entries');
end

if sum(sum(Sigma == Sigma')) ~= 4
error('The covariance matrix is not symmetric.');
end
if ((Sigma(1,1) ¡= 0) — (det(Sigma) ¡= 0))
error('Sigma is not positive definite.');
end

A = zeros(2);
A(1,1) = sqrt(Sigma(1,1));
A(2,1) = Sigma(1,2)/A(1,1);
A(2,2) = sqrt(Sigma(2,2)-A(2,1)^2);

X = mu + A*Z;
```

HW 9 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday April 14, 2009

Note: The Matlab functions you write should not have any display statements or uses of plot functions. To plot or display infomation from your functions, call your functions in separate m-files.

1. (10 Points) Show that the random walk construction for simulating Brownian Motion is exact. To do this, show that $(X_{t_{i+1}} - X_{t_i})$ has distribution $N(\mu(t_{i+1} - t_i), \sigma^2(t_{i+1} - t_i))$ and that $(X_{t_{i+i}} - X_{t_i}) \perp (X_{t_i} - X_{t_{i-1}})$.

2. (10 Points) Write a Matlab function called *brownianMotion* saved in m-file *brownianMotion.m*. It should take drift μ, volatility σ, time T, and number of time-steps N as inputs ($\Delta t = \frac{T}{N}$). Your function should return a vector of simulated values $[W_{t_0}, W_{t_1}, \ldots, W_{t_N}]^T$ from a Brownian motion W_t that is distributed $BM(\mu, \sigma^2)$. For this problem please submit the following:

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of a few test runs of your m-file using different inputs. Plot your simulation in each case.

3. (10 Points) Write a Matlab function called *multiVarNormal* saved in m-file *multiVarNormal.m*. It should take mean vector $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ as inputs. Your function should return a d-vector of normally distributed random variables according to the distribution $N(\mu, \Sigma)$. Your program should check that Σ is symmetric positive definite. You should compute the Cholesky factorization of Σ

$$AA^T = \Sigma \tag{93}$$

using the algorithm that we studied in class. For this problem submit the following

- A printout of your m-file,
- an electronic version of your m-file sent to me as an attachment in an email, and
- a printout of a few test runs of your program using different inputs.

4. (10 Points) Write a Matlab function called *geoBrownianMotion* saved in m-file *geoBrownianMotion.m*. It should take drift μ, volatility σ, time T, and number of time-steps N as inputs ($\Delta t = \frac{T}{N}$). Your function should return a vector of simulated values $[S_{t_0}, S_{t_1}, \ldots, S_{t_N}]^T$ from a geometric Brownian motion S_t that is distributed $GBM(\mu, \sigma^2)$.

   Hint: Call your function *brownianMotion* in this program. Make sure to be careful with your drift terms.

   For this problem please submit the following:

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of a few test runs of your m-file using different inputs. Plot your simulation in each case.

HW 9 - Math 573, Marcel Blais, Spring 2009

1. By definition,

$$(X_{t_{i+1}} - X_{t_i}) = \mu(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i} \cdot Z_{i+1} \tag{94}$$

where Z_{i+1} is a standard normal random variable.

To compute the distribution of $(X_{t_{i+i}} - X_{t_i})$, we have three steps:

- By (94), $X_{t_{i+1}} - X_{t_i}$ is of the form $a + bZ_{i+1}$ where $a, b \in \mathbb{R}$. Therefore $X_{t_{i+1}} - X_{t_i}$ is normally distributed.
- $E(X_{t_{i+1}} - X_{t_i}) = E[\mu(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i} \cdot Z_{i+1}]$

  $= E[\mu(t_{i+1} - t_i)] + \sigma\sqrt{t_{i+1} - t_i} \cdot E[Z_{i+1}]$

  $= \mu(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i} \cdot 0$

  $= \mu(t_{i+1} - t_i)$
- $Var(X_{t_{i+1}} - X_{t_i}) = Var[\mu(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i} \cdot Z_{i+1}]$

  $= Var[\mu(t_{i+1} - t_i)] + \sigma^2(t_{i+1} - t_i) \cdot Var[Z_{i+1}]$

  $= 0 + \sigma^2(t_{i+1} - t_i) \cdot 1$

Thus $(X_{t_{i+i}} - X_{t_i})$ has distribution $N(\mu(t_{i+1} - t_i), \sigma^2(t_{i+1} - t_i))$.

To show that $(X_{t_{i+i}} - X_{t_i})$ and $(X_{t_i} - X_{t_{i-1}})$ are independent, we look at their constructions:

$$\begin{aligned}(X_{t_{i+1}} - X_{t_i}) &= \mu(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i} \cdot Z_{i+1} \\ (X_{t_i} - X_{t_{i-1}}) &= \mu(t_i - t_{i-1}) + \sigma\sqrt{t_i - t_{i-1}} \cdot Z_i.\end{aligned} \tag{95}$$

The only random variable term in $(X_{t_{i+1}} - X_{t_i})$ is Z_{i+1}, and the only random variable term in $(X_{t_i} - X_{t_{i-1}})$ is Z_i. Since Z_{i+1} and Z_i are independent, $(X_{t_{i+1}} - X_{t_i})$ and $(X_{t_i} - X_{t_{i-1}})$ are also independent.

2. Matlab Code:

```
%Simulates a Brownian motion with timestep T/N
% Inputs: mu, Drift of the Brownian motion, sigma, Volatility of the
Brownian Motion, N, Number of timesteps, T, Final time, and W_0,
Initial value

function [W]=brownianMotion(mu, sigma, N, T, W_0)

deltaT=T/N;
W=zeros(N,1);
W(1)=W_0;
a=mu*deltaT;
b=sigma*sqrt(deltaT);
for i=2:N
W(i)=W(i-1)+a+b*randn(1);
end
```

3. Matlab Code:

```
% Generates a vector of multivariate normal
% Input mu, the mean vector of the normal distribution, Sigma, co-
variance matrix of the normal distribution, and N, number of pseudo-
random values to be returned

function [randValues] = multiVarNormal(mu, Sigma, N)
d=length(mu);
[muRows,muCols] = size(mu);
if muRows == 1
mu = mu';
end
[n,m] = size(Sigma);
if d~=n — d~=m
error('Dimensions must agree.')
end
if Sigma ~= Sigma
error('Sigma must be symmetric.')
end
if min(eig(Sigma))¡= 0
error('Sigma must be positive definite.')
end
A=zeros(d);
for jj=1:d
for ii=jj:d
v(ii)= Sigma(ii,jj);
for kk=1:jj-1
v(ii)=v(ii)-A(jj,kk)*A(ii,kk);
end
A(ii,jj)=v(ii)/sqrt(v(jj));
end
end
randValues = zeros(d,N);
for i=1:N
Z=randn(d,1);
randValues(:,i)= mu + A*Z;
end
```

4. Matlab Code:

```
% Generates a path of a geometric Brownian motion with timestep
T/N
% Input mu, drift of the geometric Brownian motion, sigma, volatility
of the geometric Brownian motion, N, number of timesteps, T, final
time in the simulation, and S_0, initial value

function [S]=geoBrownianMotion(mu, sigma,N,T,S_0)

deltaT=T/N;
S=zeros(N+1,1);
S(1)=S_0;

Z = randn(N+1,1);

for ii = 2:N+1
S(ii) = S(ii-1)*exp( (mu-0.5*sigma^2)*deltaT + sigma*sqrt(deltaT)*Z(ii))
;
end
```

HW 10 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday April 21, 2009

Note: The Matlab functions you write should not have any display statements or uses of plot functions. To plot or display infomation from your functions, call your functions in separate m-files.

1. (10 Points) Write a Matlab function called *asianOption* saved in m-file *asianOption.m* that prices Asian call and put options with discrete monitoring. It should take interest rate r, drift μ, volatility σ, time T, strike K, initial underlying price S_0, number of monitoring dates M, and number of time-steps N for the underlying asset simulation as inputs ($\Delta t = \frac{T}{N}$). Assume that the monitoring dates are equally spaced apart.

   You may force the user to enter a value of N that is an integer multiple of M. *Hint:* Use the *error* function in Matlab for this.

   Use your function to price an Asian call option with monthly monitoring and parameters $K = 10.5$, $S_0 = 10$, $T = 1$, $\mu = r = .05$, and $\sigma = .01$. Give a 95% confidence interval for your option price.

   For this problem please submit the following:

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of your program being used to price the specified option.

2. (10 Points) Write a Matlab function called *barrierOption* saved in m-file *barrierOption.m* that prices barrier call and put options with discrete monitoring. It should take interest rate r, drift μ, volatility σ, time T, strike K, initial underlying price S_0, number of monitoring dates M, and number of time-steps N for the underlying asset simulation as inputs ($\Delta t = \frac{T}{N}$). Assume that the monitoring dates are equally spaced apart.

   Your program should take inputs that allow the user to specify whether the option is down-and-in, down-and-out, up-and-in, or up-and-out. You may force the user to enter a value of N that is an integer multiple of M.

   Use your function to price an up-and-in call option with monthly monitoring and parameters $K = 10.5$, $S_0 = 10$, $b = 11$, $T = 1$, $\mu = r = .05$, and $\sigma = .01$. Give a 95% confidence interval for your option price.

   For this problem please submit the following:

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of your program being used to price the specified option.

3. (15 Points) Write a Matlab function called *multipleGeoBrownianMotion* saved in m-file *multipleGeoBrownianMotion.m.* It should take drift vector $[\mu_1, \mu_2, \ldots, \mu_d]^T$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ for the distribution of the underlying Brownian motion as inputs. It should also take time T and number of time-steps N as inputs. Your function should return a matrix of d simulated geometric Brownian motions

$$\begin{bmatrix} S_1(t_0) & S_1(t_1) & \ldots & S_1(t_N) \\ S_2(t_0) & S_2(t_1) & \ldots & S_2(t_N) \\ S_3(t_0) & S_3(t_1) & \ldots & S_3(t_N) \\ \vdots & \vdots & \vdots & \vdots \\ S_d(t_0) & S_d(t_1) & \ldots & S_d(t_N) \end{bmatrix} \tag{96}$$

from a d-dimensional geometric Brownian motion $\mathbf{S}_t$ that is distributed $GBM([\mu_1, \mu_2, \ldots, \mu_d]^T, \Sigma)$. [3]

For this problem please submit the following:

- A printout of your m-file,
- an electronic version of your m-file sent to me as an attachment in an email, and
- a printout of a two test runs of your m-file using different inputs. Plot your simulation in each case with all the Brownian motions on the same plot.

[3]For this problem we are directly implementing the material from the end of lecture 11. You will have to use this function to price options in the final homework assignment for the course.

HW 10 Solutions - Math 573, Marcel Blais, Spring 2009

1. Matlab Code:

```
% Uses a Monte Carlo method to estimate the time-zero values of the
Asian Put and the Asian Call w discrete monitoring.
% INPUTS:
% n: Number of samples to draw
% S0: Initial price of the underlying
% r: Continuously compounded annual interest rate
% sigma: Volatility of the underlying
% K: Strike price of the option
% T: Maturity of the option
% N: Number of time-steps in the simulation
% M: Number of monitoring dates
% delta: A (1-delta)% convidence interval is computed
% OUTPUTS:
% CallPrice: The Asian call option price
% putPrice: The Asian put option price
% callCI: A (1-delta)% confidence interval for the call price
% putCI: A (1-delta)% confidence interval for the put price

function[callPrice,putPrice,callCI,putCI] = asianOption(n,S0,r,sigma,K,T,N,M,delta)

if mod(N,M) ~= 0,
error('asianOption(n,S0,r,sigma,K,T,N,M): N should be a multiple of
M.');
end

monitorFactor = N/M;
callPayoffs = zeros(n,1);
putPayoffs = zeros(n,1);

for count = 1:n
S = geoBrownianMotion(r,sigma,N,T,S0);
S_at_M_dates = zeros(M+1,1);
for index = 1:M+1
```

```
monitorDate = 1 + monitorFactor*(index-1);
S_at_M_dates = S(monitorDate);
end
callPayoffs(count) = max(0,mean(S_at_M_dates)-K);
putPayoffs(count) = max(0,K-mean(S_at_M_dates));
end

callPrice = exp(-r*T) * mean(callPayoffs);
putPrice = exp(-r*T) * mean(putPayoffs);

zScore = norminv(1-delta/2,0,1);

callSampleStDev = callPayoffs - callPrice;
callSampleStDev = 1/(n-1)* sum( callSampleStDev.^2);
callCI = [callPrice - zScore*callSampleStDev/sqrt(n),callPrice + zS-
core*callSampleStDev/sqrt(n)]

putSampleStDev = putPayoffs - putPrice;
putSampleStDev = 1/(n-1)* sum( putSampleStDev.^2);
putCI = [putPrice - zScore*putSampleStDev/sqrt(n),putPrice + zS-
core*putSampleStDev/sqrt(n)]
```

2. Matlab Code:

```
% Uses a Monte Carlo method to estimate the time-zero values
% of the Barrier Put option and Barrier Call option w discrete moni-
toring.
% The barrier option can be knock-in or knock-out, and the barrier
can be
% an upward or a downward barrier.
% INPUTS:
% n: Number of samples to draw
% S0: Initial price of the underlying
% r: Continuously compounded annual interest rate
% sigma: Volatility of the underlying
% K: Strike price of the option
% b: Barrier of the option
% T: Maturity of the option
% N: Number of time-steps in the simulation
% M: Number of monitoring dates
% upDown: Set to 'U' for an upward barrier, 'D' for a downward
% barrier.
% inOut: Set to 'I' ofr a knock-in option, 'O' for a knock-out
% option.
% delta: A (1-delta)% convidence interval is computed
% OUTPUTS:
% callPrice: The Asian call option price
% putPrice: The Asian put option price
% callCI: A (1-delta)% confidence interval for the call price
% putCI: A (1-delta)% confidence interval for the put price

function[callPrice,putPrice,callCI,putCI] = barrierOption(n,S0,r,sigma,K,b,T,N,M,upDown,i

if (upDown ~= 'D') &(upDown ~= 'd') & (upDown ~= 'U') & (up-
Down ~= 'u')
error('Bad upDown parameter.')
end
if (inOut ~= 'I') &(inOut ~= 'i') & (inOut ~= 'O') & (inOut ~= 'o')
error('Bad upDown parameter.')
end
```

```
if mod(N,M) ~= 0,
error('asianOption(n,S0,r,sigma,K,T,N,M): N should be a multiple of
M.');
end

monitorFactor = N/M;
callPayoffs = zeros(n,1);
putPayoffs = zeros(n,1);

for count = 1:n

S = geoBrownianMotion(r,sigma,N,T,S0);

S_at_M_dates = zeros(M+1,1);
for index = 1:M+1
monitorDate = 1 + monitorFactor*(index-1);
S_at_M_dates = S(monitorDate);
end

callPayoffs(count) = max(0,S(N)-K);
putPayoffs(count) = max(0,K-S(N));

hitBarrier = 0;
if (max(S) ¿= b) & (upDown == 'u' — upDown == 'U')
hitBarrier = 1;
elseif (min(S) ¡= b) & (upDown == 'd' — upDown == 'D')
hitBarrier = 1;
end

if (inOut == 'i') — (inOut == 'I')
if hitBarrier == 1
barrierIndicator = 1;
else
barrierIndicator = 0;
```

```
end
end

if (inOut == 'o') — (inOut == 'O')
if hitBarrier == 1
barrierIndicator = 0;
else
barrierIndicator = 1;
end
end

callPayoffs(count) = callPayoffs(count)*barrierIndicator;
putPayoffs(count) = putPayoffs(count)*barrierIndicator;

end

callPrice = exp(-r*T) * mean(callPayoffs);
putPrice = exp(-r*T) * mean(putPayoffs);

zScore = norminv(1-delta/2,0,1);

callSampleStDev = callPayoffs - callPrice;
callSampleStDev = 1/(n-1)* sum( callSampleStDev.^2);
callCI = [callPrice - zScore*callSampleStDev/sqrt(n),callPrice + zS-
core*callSampleStDev/sqrt(n)];

putSampleStDev = putPayoffs - putPrice;
putSampleStDev = 1/(n-1)* sum( putSampleStDev.^2);
putCI = [putPrice - zScore*putSampleStDev/sqrt(n),putPrice + zS-
core*putSampleStDev/sqrt(n)];
```

3. Matlab Code:

```
% Generates a path of a geometric Brownian motion with timestep
T/N
% INPUTS:
% mu: Drift of the geometric Brownian motion
% sigma: Volatility of the geometric Brownian motion
% N: Number of timesteps
% T: Final time in the simulation
% S_0: Initial value

function [S]=geoBrownianMotion(mu, sigma,N,T,S_0)

deltaT=T/N;
S=zeros(N+1,1);
S(1)=S_0;

Z = randn(N+1,1);

for ii = 2:N+1
S(ii) = S(ii-1)*exp( (mu-0.5*sigma^2)*deltaT + sigma*sqrt(deltaT)*Z(ii))
;
end
```

HW 11 - Math 573, Marcel Blais, Spring 2009

Due before class on Tuesday April 28, 2009

1. Write a Matlab function called *basketOption* saved in m-file *basketOption.m* that prices a basket option on d underlying assets $S_1, \ldots, S_d$ with payoff

$$([c_1 S_1(T) + c_2 S_2(T) + \ldots c_d S_d(T)] - K)^+. \tag{97}$$

   You function should take inputs

   - Drift vector $[\mu_1, \mu_2, \ldots, \mu_d]^T$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ for the distribution of the underlying Brownian motion .
   - Initial underlying price vector $\mathbf{S}(0)$.
   - Maturity time T.
   - Number of time-steps N.
   - Weights $\mathbf{c} = [c_1, \ldots, c_d]^T$.
   - Interest rate r.
   - Strike price K.

   You can use your function the simulates a d-dimensional geometric Brownian motion in this problem. Please electronically submit copies of any of your own functions that you use in this problem.

   For this problem please submit the following:

   - A printout of your m-file,
   - an electronic version of your m-file sent to me as an attachment in an email, and
   - a printout of your program being used to price two options. Also specify your inputs.

HW 11 Solutions - Math 573, Marcel Blais, Spring 2009

1. Matlab Code:

```
%price a basket option
%inputs dirft vector mu (1xd), covariance matrix Sigma (dxd), initial
price vector S_0, maturity time T, number of time steps N, weight vec-
tor c, interest r, strike price K

function[P]=basketOption(mu,Sigma,S_0,T,N,c,r,K)

[m,d]=size(Sigma);

for ii=1:d
mu(ii,1)=r-.5*Sigma(ii,ii)^2;
end
mu;

s=multipleGeoBrownianMotion(mu,Sigma,T,N,S_0);

p=0;
for ii=1:d
Price=c(ii,1)*s(ii,d)+p;
p=Price;
end
p;

P=max(p-K,0);
```

Finite Difference Project

Finite Difference Project Summary
Using the spatialCoeffs function, I discretized the Black-Scholes-Merton partial differential equation in time to get the matrix A. I found the eigenvalues and eigenvectors of the matrix A which allowed me to find the amplification errors. For a call option I used the boundary conditions C(Smax,t)=Smax*K*exp(-r(T-t)) where Smax is very large and C(Smin,t)=0, where Smin=0. For a put option I used the boundary conditions V(Smax,t)=0 for very large Smax and V(Smin,t)=K*exp(-r(T-t)) for Smin=0. Then to discretize the scheme in time, the user has the option in use either the Jacobi Method or direct solver. The Jacobi Method iterates until the error is less than an epsilon specified by the user. A graph of the option price over time is an output as well as the minimum and maximum eigenvalues, the amplification errors and whether or not the scheme is stable. The Matlab code is broken into three programs as follows.

1. %uses the Jacobi method to find a solution for the u vector.

```
function[u]=jacobiMethod(N,F,A,epsilon)

u=ones(N,1);
errorCheck=1;
while errorCheck ¿ epsilon
for t=1:N-1
newU=((F(:,t)+F(:,t+1))-dot(A(t,:),u))/A(t,t);
end
errorCheck = norm(newU-u,2);
u=newU;
end
```

2.
```
% Takes imputs deltaX, r, N, and sigma to return alpha, gamma, beta
of the
% matrix A as well as the matrix A that results from spatial discretiza-
tion
% of the PDE.
% Used to be function[A]=spatialCoeffs(deltaX,r,N,sigma)
function[A]=spatialCoeffs(deltaX,r,N,sigma)


x=0;
d=zeros(N-1,1);
c=zeros(N,1);
f=zeros(N-1,1);
for ii=1:N-1
alpha=((r*x)/(2*deltaX))+(((sigma^2)*(x^2))/(2*(deltaX^2)));
d(ii,1)=alpha;
d;
beta=((-r*x)/(2*deltaX))+(((sigma^2)*(x^2))/(2*(deltaX^2)));
f(ii,1)=beta;
f;
x=x+deltaX;
end
for jj=1:N
gamma=(((sigma^2)*(x^2))/(deltaX^2))+r;
c(ii,1)=-gamma;
c;
x=x+deltaX;
end
B=zeros(N);
A=diag(c)+diag(d,1)+diag(f,-1);
A(1,1)=1;
A(1,3)=1;
A(N,N-2)=1;
A(N,N-1)=-2;
A(N,N)=1;
A(1,2)=-2;
A;
```

3.
```
%optionPricer is a program to price a European option using the
%Crank-Nicolson finite difference method
%The following parameters are to be filled out before running this file
Option=1; %Chose 1 for a call or 2 for a put
sigma=.1; %volatility
r=.05; %interest rate
T=10; %time at maturity
K=6; %strike price
S_T=2 %final price
Jacobi=1; %for the Jacobi method chose 2, for the direct method chose
1
delta_t=1; %time steps
deltaX=1; %spatial steps
epsilon=.00001; %error tolerance

T=T+1;
S_max=5*K;
N=S_max/deltaX; %number of spaces
N=N+1;

%load in the contents of the file spatialCoeffs.m and run the function
A=spatialCoeffs(deltaX,r,N,sigma);
[V,D]=eig(A);
X=V;
minEig=min(D);
maxEig=max(D);
lambda=D;

minEigenvalue=min(minEig)
maxEigenvalue=max(maxEig)

if Option==1
u=zeros(N,1);
x_start=T;
for ii=1:N
u(ii,1)=max((x_start+(ii*deltaX))-K,0);
```

```
u;
end
F=zeros(N,N+1);
for jj=1:N
F(N,jj)=S_max*K*exp(-r*(T-jj));
F;
end
elseif Option==2
u=zeros(N,1);
x_start=T;
for ii=1:N
u(ii,1)=max(K-(x_start+(ii*deltaX)),0);
u;
end
F=zeros(N,N+1);
for jj=1:N
F(1,jj)=K*exp(-r*(T-jj));
F;
end
else disp('Error in Option input')
end

E=ones(N,1);
for ii=1:N
ampError(ii)=(1+(1/2)*lambda(ii,ii)*delta_t)/(1-(1/2)*lambda(ii,ii)*delta_t);
E(ii,1)=ampError(ii);
E;
end
ampErrors=E

stable=1;
for jj=1:N
if abs(ampErrors(jj))¿1
stable=0;
disp(['Amp Error ',num2str(ampErrors(jj)),' causing instability.'])
end
end
```

```
if stable==0
disp('Routine is unstable')
else disp('Routine is stable')
end

[d1,d2]=size(A);
A_hat=eye(d1)-(delta_t/2)*A;

U=zeros(N,N);
U(:,1)=u;
old_u=u;
for jj=2:T
b=(eye(N)+(.5*delta_t*A))*old_u+(delta_t/2)*(F(:,jj)+F(:,jj+1));
if Jacobi==1
new_u=inv(A_hat)*b;
elseif Jacobi==2
new_u=jacobiMethod(N,F,A,epsilon);
else disp('Error in Jacobi input')
end
U(:,jj)=new_u;
U;
old_u=new_u;
end

surf(U)
```

References

1. Tavella, D. & Randall, C. (2000). *Pricing Financial Instruments: The Finite Difference Method*. New York: Jonh Wiley & Sons, Inc.

2. Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering.* New York: Springer Science + Business Media, LLC.