

**BUS 864**  
**Computing Notes**  
**March 8, 2006**  
Anton Theunissen

## Contents

|          |                                                          |          |
|----------|----------------------------------------------------------|----------|
| <b>1</b> | <b>Introduction</b>                                      | <b>1</b> |
| <b>2</b> | <b>Reading</b>                                           | <b>1</b> |
| <b>3</b> | <b>Crank-Nicolson</b>                                    | <b>2</b> |
| 3.1      | Setting the coefficients for the valuation PDE . . . . . | 2        |
| 3.2      | Boundary conditions . . . . .                            | 4        |
| 3.3      | Setting the dimensions of the state space grid . . . . . | 4        |
| 3.4      | Additional conditions . . . . .                          | 4        |
| <b>4</b> | <b>Random Number Generators</b>                          | <b>4</b> |
| 4.1      | Uniform random variables . . . . .                       | 4        |
| 4.2      | Standard Normal Random Variables . . . . .               | 4        |
| <b>5</b> | <b>Simulating Correlated Default Times</b>               | <b>5</b> |
| 5.1      | Gaussian Copula . . . . .                                | 5        |
| 5.1.1    | Cholesky Decomposition of Covariance Matrix . . . . .    | 5        |
| 5.1.2    | Common Factor . . . . .                                  | 6        |
| <b>6</b> | <b>VBA Subroutines</b>                                   | <b>7</b> |
| 6.1      | CNPRICE . . . . .                                        | 7        |
| 6.2      | CNSET and CNSTEP . . . . .                               | 9        |
| 6.3      | Ran2 . . . . .                                           | 12       |
| 6.4      | GasDev . . . . .                                         | 14       |
| 6.5      | Cholesky . . . . .                                       | 15       |

## 1 Introduction

This document contains notes and hints which will (hopefully) help you in completing your BUS 864 assignments. Additions and/or changes may be made to this document from time to time. Please check back regularly for the latest version.

## 2 Reading

Back, Kerry, *A Course in Derivative Securities: Introduction to Theory and Computation*, Springer, 2005, ISBN: 3540253734.  
(Appendix A may be a good substitute for Jackson and Staunton (2001) if you have a little VBA experience).

Haugh, Martin, *The Monte Carlo Framework, Examples from Finance and Generating Correlated Random Variables*, 2004, (Bus 864 web site).

Hull, John C., *Options, Futures and Other Derivatives*, 6<sup>th</sup> ed., Prentice Hall, 2005, ISBN: 0131499084.  
(Two chapters on numerical stuff - chaps 18 and 20 in 5<sup>th</sup>ed).

Jackson, Mary and Mike Staunton, *Advanced Modelling in Finance using Excel and VBA*, John Wiley & Sons, 2001, ISBN: 0471499226.  
(Don't let the title frighten you. This is a good place to start (chaps 1 - 4 ), if you are new to VBA. Some useful VBA subroutines on accompanying CD).

Press, William et al, *Numerical Recipes in C: The Art of Scientific Computing*, 2<sup>nd</sup> ed., Cambridge, 1992, ISBN: 0521431085.  
(Fortran 77, Fortran 90, C versions available at <http://www.nr.com/>. C++ version available in print).

Roman, Steven, *Writing Excel Macros with VBA*, 2<sup>nd</sup> ed., O'Reilly & Associates, 2002, ISBN: 0596003595.  
(Nice treatment of the Excel object model. Good VBA reference book).

Ross, Sheldon, *A First Course in Probability*, 7<sup>th</sup> ed., Prentice Hall, 2005, ISBN: 0131856626.  
(Any edition should do. Your favorite prob/math-stats book may be just as good).

### 3 Crank-Nicolson

You need the following bits of VBA code to implement the CN algorithm in Excel:

- The subroutines CNSET and CNSTEP.
- A collection of VBA functions FNA, FNB, FNC, FMIN, FMAX, which must be available to CNSET.
- A ‘main’ program or subroutine that accepts/reads data (from an Excel sheet or a dialogue box), sets up the appropriate boundary conditions for the securities, and calls the CNSET and CNSTEP subroutines to price the securities.

To get your code to generate reasonably accurate prices, you need to do (think about) the following:

- Set the coefficients for the valuation pde.
- Set the appropriate boundary conditions for the securities to be priced.
- Set the dimensions of the state space grid,  $S \times T$  appropriately.
- Impose additional ‘conditions’ on the value of the security in the interior of  $S \times T$ .

#### 3.1 Setting the coefficients for the valuation PDE

See your course notes, “Valuation by Arbitrage”, for a more thorough and careful exposition of the material covered in this section. Whenever you encounter an undefined symbol here, for example  $k$ , look for the definition in ‘notes1.pdf’ or ‘notes2.pdf’.

We consider (for now) one factor/state variable models only. The stochastic process governing the evolution of the state variable  $s$  is geometric Brownian motion:

$$ds = \alpha s dt + \sigma s dz \quad (1)$$

Consider a (derivative) security,  $U$ , with value or price at time  $t$ ,  $U(s, t)$ . By Ito’s lemma, the price evolves according to:

$$dU = \left( \frac{1}{2} \sigma^2 s^2 U_{ss} + \alpha s U_s + U_t \right) dt + \sigma s U_s dz \quad (2)$$

Risk neutral valuation implies:

$$\frac{E^Q [dU]}{dt} = rU \quad (3)$$

$$\frac{1}{2} \sigma^2 s^2 U_{ss} + (\alpha - \lambda) s U_s + U_t = rU \quad (4)$$

where  $\lambda$  represents the market price of  $s$ -risk,  $Q$  is the risk neutral or martingale measure and  $r$  is the risk-free rate. This is the fundamental valuation pde.

In the CN code we represent the valuation pde as follows:

$$AU_{ss} + BU_s + CU - U_t = 0 \quad (5)$$

so:

$$\begin{aligned} A &= \frac{1}{2}\sigma^2 s^2 \\ B &= (\alpha - \lambda) s \\ C &= -r \end{aligned}$$

Note that we have changed the sign on  $U_t$ . When we employ the CN algorithm to solve the valuation pde we let time run ‘backwards’ from the maturity/exercise date ( $t = T$ ) to the present time ( $t = 0$ ).

Let’s consider a few variations on this theme.

#### **State variable is a traded, non-dividend paying security**

If  $s$  is a traded security, the market price of  $s$ -risk is:

$$\lambda = (\alpha - r) s \quad (6)$$

The risk neutral process for  $s$  is:

$$ds = r s dt + \sigma s dz \quad (7)$$

and the valuation pde is:

$$\frac{1}{2}\sigma^2 s^2 U_{ss} + r s U_s - r U - U_t = 0 \quad (8)$$

so:

$$\begin{aligned} A &= \frac{1}{2}\sigma^2 s^2 \\ B &= r s \\ C &= -r \end{aligned}$$

#### **State variable is a traded security paying a continuous dividend**

Now the risk neutral process for  $s$  is:

$$ds = (r - c) s dt + \sigma s dz \quad (9)$$

where  $c$  is the continuous, proportional dividend paid by the underlying security.

The valuation pde is:

$$\frac{1}{2}\sigma^2 s^2 U_{ss} + (r - c) s U_s - r U - U_t = 0 \quad (10)$$

so:

$$\begin{aligned} A &= \frac{1}{2}\sigma^2 s^2 \\ B &= (r - c) s \\ C &= -r \end{aligned}$$

### 3.2 Boundary conditions

Loosely speaking, boundary conditions are the conditions that we impose on the value of the security,  $U(s, t)$  along the ‘edges’ of  $S \times T$ . The characteristics or state contingent pay-offs of the security will often imply certain boundary conditions. For example, we may know  $U(s, T)$  for all values of  $s(T)$ . We may know  $U(s_{min}, t)$  for all  $t \in [0, T]$  - think of  $U(s, t)$  as the value of a firm’s default risky debt,  $s$  as the market value of the firm’s assets, and  $s_{min}$  as an absorbing ‘bankruptcy’ state. In addition, if the debt is a zero-coupon bond with principal  $B$ , we know that  $U(s, T) = B$  so long as bankruptcy has not occurred in  $[0, T]$ .

#### Calls and Puts

Here’s an easy one. For (European and American) calls and puts the boundary conditions at  $T$  are:

$$\text{Call : } U(s, T) = \max[0, s(T) - X]$$

$$\text{Put : } U(s, T) = \max[0, X - s(T)]$$

### 3.3 Setting the dimensions of the state space grid

The obvious things to think about here are the magnitudes of  $k$  and  $h$ . Also, if we do not have a boundary condition at  $s_{max}$  and/or  $s_{min}$ , relying instead on quadratic extrapolation, we should ensure that the  $s$ -process has lots of ‘room to evolve’ through  $S \times T$  by setting  $s_{max}$  and/or  $s_{min}$  far enough away from  $s(0)$ , the  $s$ -level at which we want to find  $U(s, 0)$ . (*See notes2.pdf*).

### 3.4 Additional conditions

Again, the nature of the security will often impose additional ‘interior’ conditions. Another easy example is the case of American calls and puts. In addition to the boundary condition at  $T$ , we know that everywhere in the interior of  $S \times T$ , the following must hold:

$$\text{Call : } U(s, T) = \max[U(s, t), s(t) - X]$$

$$\text{Put : } U(s, T) = \max[U(s, t), X - s(t)]$$

## 4 Random Number Generators

Functions for generating (pseudo) random numbers are included in module 1 of *RanNum.xls* on the course web site. Module 2 contains a simple subroutine to call the functions.

### 4.1 Uniform random variables

Ran2 generates uniform deviates on the unit interval,  $U \sim \text{Uni}(0, 1)$ . See Press et al (1992) for a discussion of the algorithm. See section(6.3) for the code.

### 4.2 Standard Normal Random Variables

GasDev generates standard normal deviates,  $Z \sim \phi(0, 1)$ . See Press et al (1992) for a discussion of the algorithm. See section(6.4) for the code.

## 5 Simulating Correlated Default Times

### 5.1 Gaussian Copula

$$\begin{aligned}
 F(t_1, \dots, t_n) &= C_{\Sigma}(u_1, \dots, u_n) \\
 &= C_{\Sigma}(F_1(t_1), \dots, F_n(t_n)) \\
 &= \Phi_{\Sigma}^n(\Phi^{-1}(F_1(t_1)), \dots, \Phi^{-1}(F_n(t_n)))
 \end{aligned} \tag{11}$$

Notation:

|                      |                                                                   |
|----------------------|-------------------------------------------------------------------|
| $n$                  | number of reference entities in portfolio                         |
| $\Sigma$             | covariance matrix for $n$ standard normal random variables        |
| $\mathbf{A}$         | Cholesky decomposition of $\Sigma$                                |
| $\mathbf{Z}$         | vector of independent standard normal random variables            |
| $\mathbf{X}$         | vector of dependent (correlated) standard normal random variables |
| $\mathbf{U}$         | vector of dependent (correlated) uniform random variables         |
| $\Phi()$             | standard normal distribution function                             |
| $\phi()$             | standard normal density function                                  |
| $F_i(t)$             | marginal default time distribution function for issuer $i$        |
| $F(t_1, \dots, t_n)$ | a joint default time distribution function                        |
| $\tau_i$             | default time for issuer $i$ .                                     |

#### 5.1.1 Cholesky Decomposition of Covariance Matrix

See Haugh (2004, section 3) for a brief description of Cholesky decomposition. A Cholesky function is included in module 1 of *ExpoCalib.xls* on the course web site. See section (6.5) for the code.

Covariance matrix:

$$\Sigma = \begin{bmatrix} 1 & \rho & \rho \\ \rho & 1 & \rho \\ \rho & \rho & 1 \end{bmatrix}$$

Cholesky decomposition:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ \rho & \sqrt{1-\rho^2} & 0 \\ \rho & \frac{\rho-\rho^2}{\sqrt{1-\rho^2}} & \sqrt{1-\rho^2 - \frac{(\rho-\rho^2)^2}{1-\rho^2}} \end{bmatrix}$$

$$\mathbf{A}^T = \begin{bmatrix} 1 & \rho & \frac{\rho-\rho^2}{\sqrt{1-\rho^2}} \\ 0 & \sqrt{1-\rho^2} & \sqrt{1-\rho^2 - \frac{(\rho-\rho^2)^2}{1-\rho^2}} \\ 0 & 0 & \sqrt{1-\rho^2 - \frac{(\rho-\rho^2)^2}{1-\rho^2}} \end{bmatrix}$$

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 1 & \rho & \rho \\ \rho & 1 & \rho \\ \rho & \rho & 1 \end{bmatrix}$$

Procedure:

- Specify/estimate  $\Sigma$ .

- Determine  $\mathbf{A}$  such that  $\Sigma = \mathbf{A}\mathbf{A}^T$ .
- Generate a vector of independent standard normal deviates,  $\mathbf{z}^T = \{z_1 \dots z_n\}$ .
- Impose the dependence structure by calculating the correlated vector of standard normal deviates,  $\mathbf{x} = \mathbf{A}\mathbf{z}$ .  $\mathbf{X} \sim N(0, \Sigma)$ .
- Set  $u_i = \Phi(x_i)$ . This yields a realization of the uniform random vector,  $\mathbf{U}$ ,  $\mathbf{u}^T = \{u_1 \dots u_n\}$ . The  $U_i$  have a Gaussian Copula dependence structure.
- Find the simulated correlated default time for every issuer,  $\tau_i = F^{-1}(u_i)$ .

### 5.1.2 Common Factor

Notation:

$\rho$  correlation parameter  
 $Z_c$  standard normal random variable, the common factor

Procedure:

- Generate a vector of independent standard normal deviates for each issuer,  $\mathbf{z}^T = \{z_1 \dots z_n\}$ .
- Generate an additional standard normal deviate for the common factor,  $Z_c$ .
- Set  $u_i = \Phi(\rho^{1/2} z_c + (1 - \rho)^{1/2} z_i)$ . This yields a realization of the uniform random vector,  $\mathbf{U}$ , with a Gaussian Copula dependence structure.
- Find the simulated correlated default time for every issuer,  $\tau_i = F^{-1}(u_i)$ .

## 6 VBA Subroutines

### 6.1 CNPRICE

*This code is in module 1 of the CN.xls workbook on the BUS 864 homepage.*

Option Explicit

```
Sub CNPRICE()
'*****
'A simple program to price a european put on a non-dividend paying security.
'Program calls the Crank-Nicholson subroutines CNSET and CNSTEP.
'*****

Dim i As Integer, j As Integer
Dim N As Integer, IMIN As Integer, IMAX As Integer, IFUT As Integer, _
    IFN As Integer, PARM_COUNT As Integer
Dim SMIN As Double, SMAX As Double, K As Double, TMAT As Double, RF As Double, _
    SIGMA As Double, XPRICE As Double, T As Double

'Input form "In_Out" sheet
N = Range("N")
IMIN = Range("IMIN")
IMAX = Range("IMAX")
IFUT = Range("IFUT")
SMIN = Range("SMIN")
SMAX = Range("SMAX")
K = Range("K")
TMAT = Range("TMAT")
RF = Range("RF")
SIGMA = Range("SIGMA")
XPRICE = Range("XPRICE")
PARM_COUNT = Range("PARM_COUNT")

Dim U() As Double, PARM() As Double, S() As Double
ReDim U(N)
ReDim PARM(PARM_COUNT)
ReDim S(N)

'Populate PARM() for passing values to FN functions
PARM(0) = SIGMA
PARM(1) = RF

'Call CNSET - setup to solve pde
Call CNSET(N, SMIN, SMAX, K, IFUT, IMIN, IMAX, PARM)

For i = 0 To N
    S(i) = SMIN + (SMAX - SMIN) / N * i
    U(i) = Application.WorksheetFunction.Max(0#, XPRICE - S(i))
Next i

'Solution loop to solve pde
T = 0#
Do
    Call CNSTEP(T, U)
    T = T + K
    Application.StatusBar = "CNSTEP LOOP " & T
```

```

Loop While T <= TMAT - K / 2

'Output
Range("S_0:U_MAX").ClearContents
For i = 0 To N
    Range("S_0").Cells(i + 1).Value = S(i)
    Range("U_0").Cells(i + 1).Value = U(i)
Next i

End Sub

Public Function FNA(S As Double, PARM() As Double) As Double
FNA = (PARM(0) * S) ^ 2 / 2#
End Function

Public Function FNB(S As Double, PARM() As Double) As Double
FNB = PARM(1) * S
End Function

Public Function FNC(S As Double, PARM() As Double) As Double
FNC = -PARM(1)
End Function

Public Function FMIN(T As Double) As Double
FMIN = 0#
End Function

Public Function FMAX(T As Double) As Double
FMAX = 0#
End Function

```

## 6.2 CNSET and CNSTEP

*This code is in module 2 of the CN.xls workbook on the BUS 864 homepage.*

```

Option Explicit
Private ISMIN As Integer, ISMAX As Integer, N As Integer
Private D() As Double, GAM() As Double, ARR() As Double

Sub CNSET(NN As Integer, SMIN As Double, SMAX As Double, K As Double, _
          IFUT As Integer, IMIN As Integer, IMAX As Integer, PARM() As Double)
'*****
'CNSET sets up coefficient array for Crank-Nicholson algorithm to solve 1 state
'variable plus time partial differential equations. Use in conjunction with subroutine
'CNSTEP.
'The PDE to be solved has the form:
',
',
',          FNA*Uss + FNB*Us + FNC*U - Ut = 0
',
'Arguments:  NN      number of grid intervals in state space S
',           SMIN    min value of S on grid
',           SMAX    max value of S on grid
',           K       size of time step on grid
',           IFUT    flag for futures contract, FNC = 0 if IFUT = 1
',           IMIN    flag for SMIN boundary (0 - quad extrapol, 1 - given value)
',           IMAX    flag for SMAX boundary (0 - quad extrapol, 1 - given value)
',           PARM    array of mdl parameters passed to FNA, FNB, FNC
',           ARR     array of coefficients for CNSTEP
',
'This is a rough 'port' of the FORTRAN code by Jones (1988). Use at your own risk!
'*****

Dim i As Integer, j As Integer
Dim H As Double, FUTURE As Double, S As Double, AX As Double, BX As Double, _
    CX As Double, DENOM As Double, G As Double

'Variables global to this module
N = NN
ISMIN = IMIN
ISMAX = IMAX
ReDim ARR(N, 1 To 4)
ReDim D(N)
ReDim GAM(N)

H = (SMAX - SMIN) / N
FUTURE = 1#
If IFUT = 1 Then FUTURE = 0#

'Interior Coefficients
For i = 1 To N - 1
    S = SMIN + (i * H)
    AX = FNA(S, PARM) * 2# * K
    BX = FNB(S, PARM) * H * K
    CX = FNC(S, PARM) * FUTURE * 2# * H ^ 2 * K
    DENOM = CX - 2# * AX - 4# * H ^ 2
    ARR(i, 1) = (AX - BX) / DENOM
    ARR(i, 2) = 1#
    ARR(i, 3) = (AX + BX) / DENOM

```

```

        ARR(i, 4) = 1# + 8# * H ^ 2 / DENOM
    Next i

'Boundary Coefficients
If ISMIN = 1 Then
    'Known value at SMIN
    ARR(0, 1) = 0#
    ARR(0, 2) = 1#
    ARR(0, 3) = 0#
Else
    'Quadratic extrapolation at SMIN
    G = ARR(1, 3) / (ARR(2, 2) + 3# * ARR(2, 3))
    ARR(0, 1) = 0#
    ARR(0, 2) = G * ARR(2, 3) - ARR(1, 1)
    ARR(0, 3) = G * (ARR(2, 1) - 3# * ARR(2, 3)) - ARR(1, 2)
    ARR(0, 4) = G
End If

If ISMAX = 1 Then
    'Known value at SMAX
    ARR(N, 1) = 0#
    ARR(N, 2) = 1#
    ARR(N, 3) = 0#
Else
    'Quadratic extrapolation at SMAX
    G = ARR(N - 1, 1) / (ARR(N - 2, 2) + 3# * ARR(N - 2, 1))
    ARR(N, 1) = G * (ARR(N - 2, 3) - 3# * ARR(N - 2, 1)) - ARR(N - 1, 2)
    ARR(N, 2) = G * ARR(N - 2, 1) - ARR(N - 1, 3)
    ARR(N, 3) = 0#
    ARR(N, 4) = G
End If

End Sub

Sub CNSTEP(T As Double, U() As Double)
'*****
'CNSTEP takes 1 step in time direction in solving 1 state variable PDE using
'Crank-Nicholson algortihm. T is current time used only for passing to boundary
'value functions FMIN(T), FMAX(T) if ISMIN or ISMAX are set to 1. U(0:N) is array of
'solution so far. ARR(0:N,1:4) is coefficient array from CNSET().
',
'This is a rough 'port' of the FORTRAN code by Jones (1988). Use at your own risk!
'*****

Dim i As Integer, j As Integer
Dim IFN As Integer

'Set up RHS of tridiag system (ABC)U = D
For i = 1 To N - 1
    D(i) = -ARR(i, 1) * U(i - 1) - ARR(i, 4) * U(i) - ARR(i, 3) * U(i + 1)
Next i

If ISMIN = 1 Then 'get solution value at RMIN
    D(0) = FMIN(T)
Else

```

```

      D(0) = D(2) * ARR(0, 4) - D(1)
End If

If ISMAX = 1 Then 'get solution value at RMAX
  D(N) = FMAX(T)
Else
  D(N) = D(N - 2) * ARR(N, 4) - D(N - 1)
End If

Call TRIDAG(U)

End Sub

Sub TRIDAG(X() As Double)
'*****
'Solves for vector X in (ABC)X = D.  A,B,C,D are input vectors and are not modified.
'  A(0:N) = ARR(0:N,1)
'  B(0:N) = ARR(0:N,2)
'  C(0:N) = ARR(0:N,3)
',
'Port' of FORTRAN code by Jones adapted from Press et al(1992),
'Numerical Recipes in FORTRAN 77, second edition, Cambridge University Press, pp. 42-43.
'*****

Dim i As Integer, j As Integer
Dim BET As Double

BET = ARR(0, 2)
X(0) = D(0) / BET

For i = 1 To N
  GAM(i) = ARR(i - 1, 3) / BET
  BET = ARR(i, 2) - ARR(i, 1) * GAM(i)
  X(i) = (D(i) - ARR(i, 1) * X(i - 1)) / BET
Next i

For i = N - 1 To 0 Step -1
  X(i) = X(i) - GAM(i + 1) * X(i + 1)
Next i

End Sub

```

## 6.3 Ran2

Option Explicit

```
Const IM1 = 2147483563
Const IM2 = 2147483399
Const AM = 1 / IM1
Const IMM1 = 2147483562
Const IA1 = 40014
Const IA2 = 40692
Const IQ1 = 53668
Const IQ2 = 52774
Const IR1 = 12211
Const IR2 = 3791
Const NTAB = 32
Const NDIV = (1 + IMM1 / NTAB)
Const EPS = 0.00000012
Const RNMX = 1 - EPS
```

Function ran2(idum As Long) As Double

```
'*****
' Returns uniform random deviate on (0,1)
' Call with idum a negative integer
' This is a rough port of the C version of ran2 by:
' Press et al (1992) "Numerical Recipes in C", second ed., p. 282.
' URL: http://www.nr.com/
'*****
```

```
Dim j As Integer
Dim k As Long
Dim temp As Double
Static iy As Long
Static iv(NTAB) As Long
Static idum2 As Long
```

```
If idum <= 0 Then
    If (-idum) < 1 Then
        idum = 1
    Else
        idum = -idum
    End If
    idum2 = idum
    For j = NTAB + 7 To 0 Step -1
        k = idum / IQ1
        idum = IA1 * (idum - k * IQ1) - k * IR1
        If idum < 0 Then
            idum = idum + IM1
        End If
        If j < NTAB Then
            iv(j) = idum
        End If
    Next
    iy = iv(0)
End If
```

```
k = (idum - idum Mod IQ1) / IQ1
idum = IA1 * (idum - k * IQ1) - k * IR1
```

```

If (idum < 0) Then idum = idum + IM1
k = idum2 / IQ2
idum2 = IA2 * (idum2 - k * IQ2) - k * IR2
If (idum2 < 0) Then idum2 = idum2 + IM2
j = (iy - iy Mod NDIV) / NDIV
iy = iv(j) - idum2
iv(j) = idum
If (iy < 1) Then iy = iy + IMM1
temp = AM * iy
If (temp > RNMX) Then
    ran2 = RNMX
Else
    ran2 = temp
End If
End Function

```

## 6.4 GasDev

```
Function gasdev(idum As Long) As Double
'*****
' Returns a standard normal random deviate
' Call with idum a negative integer
' This is a rough port of the C version of gasdev by:
' Press et al (1992) "Numerical Recipes in C", second ed., pp. 289-290.
' URL: http://www.nr.com/
'*****

Static iset As Integer
Static gset As Double
Dim fac As Double
Dim RSq As Double
Dim v1 As Double
Dim v2 As Double

If iset = 0 Then
    While RSq >= 1 Or RSq = 0
        v1 = 2 * ran2(idum) - 1
        v2 = 2 * ran2(idum) - 1
        RSq = v1 * v1 + v2 * v2
    Wend
    fac = Sqr(-2 * Log(RSq) / RSq)
    gset = v1 * fac
    iset = 1
    gasdev = v2 * fac
Else
    iset = 0
    gasdev = gset
End If
End Function
```

## 6.5 Cholesky

```
Function Cholesky(Mat As Range)
'*****
'Function returns a square matrix L which is Cholsky decomposition of input
'matrix. Input matrix must be square, symmetric, positive definite.
'*****

Dim A, L() As Double, s As Double
Dim n As Integer, M As Integer, i As Integer, j As Integer, k As Integer
A = Mat
n = Mat.Rows.Count
M = Mat.Columns.Count
If n <> M Then
    Cholesky = "?"
    Exit Function
End If

ReDim L(1 To n, 1 To n)
For j = 1 To n
    s = 0
    For k = 1 To j - 1
        s = s + L(j, k) ^ 2
    Next k
    L(j, j) = A(j, j) - s
    If L(j, j) <= 0 Then Exit For
    L(j, j) = Sqr(L(j, j))

    For i = j + 1 To n
        s = 0
        For k = 1 To j - 1
            s = s + L(i, k) * L(j, k)
        Next k
        L(i, j) = (A(i, j) - s) / L(j, j)
    Next i
Next j
Cholesky = L
End Function
```