

R Textbook Companion for
Numerical Methods in Finance and
Economics: A MATLAB-Based Introduction
by Paolo Brandimarte¹

Created by
Bhushan Sahadeo Manjarekar
B.E.
Others
University of Mumbai
Cross-Checked by
R TBC Team

July 18, 2025

¹Funded by a grant from the National Mission on Education through ICT
- <http://spoken-tutorial.org/NMEICT-Intro>. This Textbook Companion and R
codes written in it can be downloaded from the "Textbook Companion Project"
section at the website - <https://r.fossee.in>.

Book Description

Title: Numerical Methods in Finance and Economics: A MATLAB-Based Introduction

Author: Paolo Brandimarte

Publisher: John Wiley & Sons, Inc., Hoboken, New Jersey

Edition: 2

Year: 2006

ISBN: 0471745030

R numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means an R code whose theory is explained in Section 2.3 of the book.

Contents

List of R Codes	4
2 Financial Theory	5
3 Basics of Numerical Analysis	12
4 Numerical Integration Deterministic and Monte Carlo Methods	23
5 Finite Difference Methods for Partial Differential Equations	44
6 Convex Optimization	49
7 Option Pricing by Binomial and Trinomial Lattices	52
8 Option Pricing by Monte Carlo Methods	61
9 Option Pricing by Finite Difference Methods	93
10 Dynamic Programming	98

List of R Codes

Exa 2.7	Basic theory of interest rates	5
Exa 2.8	Basic theory of interest rates	6
Exa 2.9	Basic pricing of fixed income securities	6
Exa 2.10	Interest rate sensitivity and bond portfolio immunization	7
Exa 2.24	Black Scholes model in MATLAB	11
Exa 3.2	Error propagation conditioning and instability	12
Exa 3.4	Vector and matrix norms	12
Exa 3.6	Vector and matrix norms	13
Exa 3.7	Condition number for a matrix	13
Exa 3.8	Condition number for a matrix	13
Exa 3.11	Direct methods for solving systems of linear equations	14
Exa 3.12	Direct methods for solving systems of linear equations	14
Exa 3.13	Iterative methods for solving systems of linear equations	15
Exa 3.14	Iterative methods for solving systems of linear equations	16
Exa 3.15	FUNCTION APPROXIMATION AND INTERPOLATION	17
Exa 3.16	Elementary polynomial interpolation	18
Exa 3.17	Elementary polynomial interpolation	18
Exa 3.18	Interpolation by cubic splines	19
Exa 3.22	Bisection method	20
Exa 3.23	Newtons method	21
Exa 3.24	Optimization based solution of non linear equations .	21
Exa 4.1	Numerical integration in MATLAB	23
Exa 4.2	MONTE CARLO INTEGRATION	23
Exa 4.3	MONTE CARLO INTEGRATION	24
Exa 4.4	Generating pseudorandom numbers	24
Exa 4.5	Generating pseudorandom numbers	25
Exa 4.6	Inverse transform method	26

Exa 4.8	Generating normal variates by the polar approach . . .	26
Exa 4.9	Generating normal variates by the polar approach . . .	27
Exa 4.10	SETTING THE NUMBER OF REPLICATIONS . . .	28
Exa 4.11	Antithetic sampling	29
Exa 4.12	BlsMCAV	30
Exa 4.14	Importance sampling	33
Exa 4.15	Generating Halton low discrepancy sequences	34
Exa 4.16	Generating Halton low discrepancy sequences	36
Exa 4.17	Generating Halton low discrepancy sequences	38
Exa 4.18	Generating Sobol low discrepancy sequences	40
Exa 4.19	Generating Sobol low discrepancy sequences	41
Exa 4.20	Generating Sobol low discrepancy sequences	42
Exa 5.1	Instability in a finite difference scheme	44
Exa 5.3	Solving the heat equation by an explicit method	46
Exa 5.4	Solving the heat equation by a fully implicit method . .	47
Exa 6.1	Finite vs infinite dimensional problems	49
Exa 6.3	Linear vs non linear problems	50
Exa 6.5	Penalty function approach	50
Exa 6.8	Kuhn Tucker conditions	51
Exa 6.12	Geometric and algebraic features of linear programming	51
Exa 7.1	Calibrating a binomial lattice	52
Exa 7.2	Accuracy of the binomial lattice for decreasing deltaT	53
Exa 7.3	Price a pay later option by a binomial lattice	54
Exa 7.4	Pricing an European call by a binomial lattice	55
Exa 7.5	Pricing an American put by a binomial lattice	56
Exa 7.6	Pricing an American spread option by a bidimensional binomial lattice	57
Exa 7.7	Pricing an European call by a trinomial lattice	59
Exa 8.1	Generate asset price paths by Monte Carlo simulation	61
Exa 8.2	Vectorized code to generate asset price paths	62
Exa 8.3	Evaluating the cost of a stop loss hedging strategy . . .	63
Exa 8.4	Vectorized code for the stop loss hedging strategy . . .	65
Exa 8.5	Evaluating the performance of delta hedging	67
Exa 8.6	Implementing and checking path generation for the stan- dard Wiener process by a Brownian bridge	70
Exa 8.7	Code to price an exchange option analytically	71
Exa 8.8	Code to price an exchange option by Monte Carlo sim- ulation	72

Exa 8.9	Crude Monte Carlo simulation for a discrete barrier option	73
Exa 8.10	Conditional Monte Carlo simulation for a discrete barrier option	75
Exa 8.11	Using conditional Monte Carlo and importance sampling for a discrete barrier option	77
Exa 8.12	Monte Carlo simulation for an Asian option	79
Exa 8.13	Monte Carlo simulation with control variates for an Asian option	80
Exa 8.14	Using the geometric average Asian option as a control variate	82
Exa 8.15	Pricing an Asian option by Halton sequences	84
Exa 8.16	Simulating geometric Brownian motion by Halton sequences and the Brownian bridge	87
Exa 8.17	Improving the estimate of the option Delta by Common Random Numbers	90
Exa 8.18	Estimating the option Delta by a pathwise estimator	91
Exa 9.3	price a European vanilla put by a straightforward explicit scheme	93
Exa 9.4	price a vanilla European option by a fully implicit method	94
Exa 9.5	price a down and out put option by the Crank Nicolson method	95
Exa 10.4	Simple asset allocation problem under uncertainty Monte Carlo sampling	98

Chapter 2

Financial Theory

R code Exa 2.7 Basic theory of interest rates

```
1 mypvvar <- function(cf,r) {
2   # get number of periods
3   n = length(cf)
4   #1 get vector of discount factors
5   df<-matrix(0,n)
6   for (i in 0:n-1){
7     df[i+1] = 1/(1+r)^(i)
8   }
9   #compute result
10  pv = cf*df
11 }
12
13 cf<-c(0, 8, 8, 8, 8, 108)
14 pv = mypvvar (cf ,0.08)
15 sum(pv)
16
17 pv = mypvvar (cf ,0.09)
18 sum(pv)
19
20 pv = mypvvar (cf ,0.07)
21 sum(pv)
```

R code Exa 2.8 Basic theory of interest rates

```
1 #install.packages("FinCal")
2 library("FinCal")
3 cf<-c(-100, 8, 8, 8, 8, 108)
4 h=polyroot(cf[length(cf):1])
5 h
6 rho=1/h-1
7 rho
8 index = which((abs(Im(rho)) < 0.001) != 0)
9 rho[index]
10 irr(cf)
```

R code Exa 2.9 Basic pricing of fixed income securities

```
1 mypvvar <- function(cf,r) {
2   # get number of periods
3   n = length(cf)
4   #1 get vector of discount factors
5   df<-matrix(0,n)
6   for (i in 0:n-1){
7     df[i+1] = 1/(1+r)^(i)
8   }
9   #compute result
10  pv = cf*df
11 }
12
13 r1=0.08
14 r2=0.09
15 P1=100/(1+r1)^5
16 P1
```

```

17 P2=100/(1+r2)^5
18 P2
19 (P2-P1)/P1
20
21 P1=100/(1+r1)^20
22 P1
23 P2=100/(1+r2)^20
24 P2
25 (P2-P1)/P1
26
27 cf1<-c(0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 108)
28 cf2<-c(0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 104)
29 P1=mypvvar(cf1,0.08)
30 P1 = sum(P1)
31 P1
32 P2=mypvvar(cf1,0.09)
33 P2 = sum(P2)
34 P2
35 (P2-P1)/P1
36 P1=mypvvar(cf2,0.08)
37 P1 = sum(P1)
38 P1
39 P2=mypvvar(cf2,0.09)
40 P2 = sum(P2)
41 P2
42 (P2-P1)/P1

```

R code Exa 2.10 Interest rate sensitivity and bond portfolio immunization

```

1 library(FinCal)
2 mypvvar <- function(cf,r) {
3   # get number of periods
4   n = length(cf)
5   #1 get vector of discount factors
6   df<-matrix(0,n)

```

```

7   for (i in 0:n-1){
8       df[i+1] = 1/(1+r)^(i)
9   }
10  #compute result
11  pv = cf*df
12 }
13
14 cfdur <- function(cf,yld) {
15     #   CFDUR  Cash flow duration and modified
        duration.
16     #   [D,MD] = CFDUR(CF,YLD) calculates the duration
        D and modified duration
17     #   (volatility) MD of a cash flow given the cash
        flow, CF, and the periodic
18     #   yield, YLD.
19     #
20     #   For example, nine payments of $2.50 and a
        final payment of $102.50 with
21     #   a yield of 2.5% returns a duration of 8.97
        periods and a modified duration
22     #   of 8.75 periods.
23     #
24     #   See also BONDCONV, BONDDUR, CFCONV.
25     #   Copyright 1995–2006 The MathWorks, Inc.
26     x<-dim(cf)
27     rowcf = x[1]
28     colcf = x[2]
29     if (rowcf == 1){
30         cf = t(cf)
31         colcf = 1
32     }
33
34     if (colcf > 1 ){
35         if (length(yld) == 1){
36             yld = yld*matrix(1,colcf)
37         }
38     }
39

```

```

40  pv = matrix(0,colcf)
41  d = pv
42  md = pv
43  m = length(cf[,1])
44  fac = t((1:m))
45  for (loop in 1:colcf){
46      # Compound the yield
47      rates = (matrix(data = 1,ncol = m,nrow = 1)*(1+
        yld[loop]))^fac
48      # find the net present value
49      pv[loop] = sum(mypvvar(c(0,cf[,loop]),yld[loop])
        )
50      # duration
51      d[loop] = sum(cf[,loop]/(rates)*fac)/pv[loop]
52      # modified duration
53      md[loop] = d[loop]/(1+yld[loop])
54  }
55  cat("Duration of cash flow:",d,"\n")
56  cat("Modified Duration (volatility) of cash flow:"
    ,md,"\n")
57  return(list(d=d,md=md))
58 }
59
60 cfconv <- function(cf,yld) {
61     # CFCONV Cash flow convexity.
62     # CX = CFCONV(CF,YLD) calculates the convexity C
        of a cash flow
63     # given the cash flow , CF, and the periodic
        yield , YLD.
64     #
65     # For example, nine payments of $2.50 and a
        final payment of $102.50
66     # with a yield of 2.5% returns a convexity of
        90.45 periods.
67     #
68     # See also BONDCONV, BONDDUR, CFDUR.
69     #
70     # Copyright 1995–2006 The MathWorks, Inc.

```

```

71   x<-dim(cf)
72   rowcf = x[1]
73   colcf = x[2]
74   if (rowcf == 1){
75     cf = t(cf)
76     colcf = 1
77   }
78
79   if (colcf > 1 ){
80     if (length(yld) == 1){
81       yld = yld*matrix(1,colcf)
82     }
83   }
84
85   pv = matrix(0,colcf)
86   cx = pv
87   m = length(cf[,1])
88   fac = t((1:m))
89   for (loop in 1:colcf){
90     #Compound the yield
91     rates = (matrix(data = 1,ncol = m,nrow = 1)*(1+
92       yld[loop]))^fac
93     #find the net present value
94     pv[loop] = sum(mypvvar(c(0,cf[,loop]),yld[loop])
95       )
96     cx[loop] = sum(cf[,loop]/(rates)*fac*(fac+1))/
97       ((1+yld[loop])^2*pv[loop])
98   }
99
100  cat("Cash flow convexity:",cx,"\n")
101  return(cx)
102 }
103
104 cf<-c(10,10,10,10)
105 p1=abs(pv.uneven(0.05,cf))
106 p2=abs(pv.uneven(0.055,cf))
107 (p2-p1)
108 cf<-matrix(cf,nrow = 1,ncol = 4)
109 yld<-matrix(data = c(0.05),nrow = 1,ncol = 1)

```

```

106 x<-cfdur(cf,yld)
107 d<-x$d
108 dm<-x$dm
109 cv=cfconv(cf,yld)
110 -dm*p1*0.005
111 -dm*p1*0.005+0.5*cv*p1*(0.005)^2

```

R code Exa 2.24 Black Scholes model in MATLAB

```

1 #install.packages("OptionPricing")
2 require(OptionPricing)
3 C0 = BS_EC(T = 5/12, K = 50, r = 0.1, sigma = 0.3,
             S0 = 50)
4 dS = 2
5 C1 = BS_EC(T = 5/12, K = 50, r = 0.1, sigma = 0.3,
             S0 = 50+dS)
6 C0[1]+ C0[2]*dS + 0.5*C0[3]*dS-2
7
8 #Answer differs in last line as the value of C0
   gamma is different than the book

```

Chapter 3

Basics of Numerical Analysis

R code Exa 3.2 Error propagation conditioning and instability

```
1 require(rootSolve)
2 fun <- function (x) x^8 - 36*x^7 + 546*x^6 -4536*x^5
   +22449*x^4 -67284*x^3 + 118124*x^2 - 109584*x +
   40320
3 sort(uniroot.all(fun, c(0, 8)),decreasing = TRUE)
4 p2<-c(1, -36.001, 546, -4536, 22449, -67284, 118124,
   -109584, 40320)
5 1/polyroot(p2)
```

R code Exa 3.4 Vector and matrix norms

```
1 v <- matrix(c(2, 4, -1, 3), nrow = 1, ncol = 4,
   byrow = TRUE)
2 norm(v,"I")
3 norm(v,"2")
4 norm(v,"1")
```

R code Exa 3.6 Vector and matrix norms

```
1 A <-matrix(c(2, 4, - 1, 3, 1, 5,-2,3,-1),nrow = 3,
             ncol = 3,byrow = TRUE)
2 norm(A,"1")
3 norm(A,"2")
4 norm(A,"I")
5 norm(A,"F")
6 norm(A,"M")
7 Z <- A %*% t(A)
8 eig<-eigen(Z)
9 sqrt(eig$values)
```

R code Exa 3.7 Condition number for a matrix

```
1 #install.packages("Matrix")
2 require(Matrix)
3 kappa(Hilbert(3),exact = TRUE)
4 kappa(Hilbert(7),exact = TRUE)
5 kappa(Hilbert(10),exact = TRUE)
```

R code Exa 3.8 Condition number for a matrix

```
1 N = 20
2 A<-diag(N)
3
4 for (i in 1:N){
5   for (j in (i+1):N){
6     A[i,j] = -1
7   }
8 }
9
10 b=-rep(1,N)
```



```

11 b[N] = 1
12 solve(A,b)
13
14 b[N] = 1.00001
15 solve(A,b)
16
17 1/rcond(A,'I')
18 2^18
19 0.00001 * 2^18

```

R code Exa 3.11 Direct methods for solving systems of linear equations

```

1 require(Matrix)
2 A<-matrix(c(1, 4, -2, -3,9, 8, 5 ,1,-6),nrow = 3,
           ncol = 3,byrow = TRUE)
3 tmp=expand(lu(Matrix(A)))
4 tmp$L
5 tmp$U
6 tmp$P
7 b <-matrix(c(1,2,3),nrow = 3,ncol = 1,byrow = T)
8 solve(A,b)
9 tmp_P<-matrix(c(0,0,1,0,1,0,1,0,0),nrow = 3,ncol =
               3,byrow = T)
10 x = solve(tmp$U,solve(tmp$L,(tmp_P%*%b)))

```

R code Exa 3.12 Direct methods for solving systems of linear equations

```

1 A<-matrix(c(3, 1, 4, 1, 5, 3, 4, 3,7),nrow = 3,ncol
           = 3,byrow = T)
2 eigen(A)
3 B<-matrix(c(1,2,3),nrow = 3,ncol = 1,byrow = T)
4 U = chol(A)
5 U

```

```
6 solve(U,(solve(t(U),B)))
```

R code Exa 3.13 Iterative methods for solving systems of linear equations

```
1 Jacobi <- function(A,b,x0,eps,MaxIter) {
2   dA = diag(A)
3   C = A - diag(dA)
4   Dinv = diag(1/dA)
5   B = - Dinv %*% C
6   b1 = Dinv %*% b
7   oldx = x0
8
9   for (i in 1:MaxIter){
10     x = B %*% oldx + b1
11     if (norm(x-oldx) < eps*norm(matrix(oldx))){
12       break
13     }
14     oldx = x
15   }
16   cat("Case of Matrix:",A,"\n")
17   cat("Terminated after iterations:",i,"\n")
18   cat("Jacobi:",x,"\n")
19   cat("Exact:",solve(A,b),"\n")
20 }
21
22 A1 <-matrix(c(3, 1, 1, 0, 1, 5, - 1, 2, 1, 0, 3, 1,
23             0, 1, 1, 4),nrow = 4,ncol = 4,byrow = T)
24 b<-matrix(c(1, 4, -2, 1),nrow = 4,ncol = 1,byrow = T)
25
26 Jacobi(A1,b,rep(0,4),1e-08,10000)
27
28 A2 <-matrix(c(2.5, 1, 1, 0, 1, 4.1, -1, 2, 1, 0,
29             2.1, 1, 0, 1, 1, 2.1),nrow = 4,ncol = 4,byrow = T)
30
31 Jacobi(A2,b,rep(0,4),1e-08,10000)
```

```

28
29 A3 <-matrix(c(2, 1, 1, 0, 1, 3.5, -1, 2, 1, 0, 2.1,
    1, 0, 1, 1, 2.1),nrow = 4,ncol = 4,byrow = T)
30 Jacobi(A3,b,rep(0,4),1e-08,10000)

```

R code Exa 3.14 Iterative methods for solving systems of linear equations

```

1 SORGaussSeidel <- function(A, b,x0, omega, eps,
    MaxIter) {
2   oldx = x0
3   x = x0
4   N = length(x0)
5   omega1 = 1 - omega
6   for (k in 1:MaxIter){
7     for (i in 1:(N-1)){
8       z = (b[i] - sum(A[i,(1:i-1)]) * x[1:(i-1)]) -
          sum(A[i,(i+1):N] * x[(i+1):N]) / A[i,i]
9       x[i] = omega * z + omega1 * oldx[i]
10    }
11    if(norm(matrix(x-oldx))<eps*norm(matrix(oldx))){
12      break
13    }
14    oldx = x
15  }
16  result$x = x
17  result$k = k
18  return(result)
19 }
20
21 A2<-matrix(c(2.5, 1, 1, 0, 1, 4.1, -1, 2, 1, 0, 2.1,
    1, 0, 1, 1,2.1),nrow = 4,ncol = 4,byrow = T)
22 b<-matrix(c(1, 4, -2, 1),nrow = 4,ncol = 1,byrow = T
    )
23 omega = seq(0,2,0.1)
24 N = length(omega)

```

```

25 NumIterations = rep(0,N)
26 for (i in 1:N){
27   result = list()
28   result = SORGaussSeidel(A2,b,rep(0,4),omega[i],1e
      -08,1000)
29   NumIterations[i] = result$k
30 }
31 plot(omega,NumIterations)
32 lines(omega,NumIterations)

```

R code Exa 3.15 FUNCTION APPROXIMATION AND INTERPOLATION

```

1 xdata<-c(1, 5, 10, 30, 50)
2 ydata<-log(xdata)
3 p = coef(lm(ydata ~ xdata + I(xdata^2)))
4 p
5
6 x = c(1, 5, 10, 30, 50)
7 y = log(x)
8 plot(x,y)
9 lm2 = lm(y~x+I(x^2))
10 lm4 = lm(y~x+I(x^2)+I(x^3)+I(x^4))
11 xplot=seq(from = 0,to = 50,by = .1)
12 lines(xplot,predict(lm4,newdata=data.frame(x=xplot))
      , col="blue")
13 # and so on
14 lines(xplot,predict(lm2,newdata=data.frame(x=xplot))
      , col="red")
15
16 http://www.utstat.utoronto.ca/reid/sta414/Rsession-polys.pdf

```

R code Exa 3.16 Elementary polynomial interpolation

```
1 x=1:10
2 y<-c(8, 2.5, -2, 0 ,5 ,2 ,4 ,7 ,4.5, 2)
3 lm9 = lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)+I(x^6)+I(x
      ^7)+I(x^8)+I(x^9))
4 xplot=seq(from = 0,to = 10,by = .05)
5 plot(x,y)
6 lines(xplot,predict(lm9,newdata=data.frame(x=xplot))
      , col="blue")
```

R code Exa 3.17 Elementary polynomial interpolation

```
1 runge <- function(x) {
2   1/(1+25*x^2)
3 }
4
5 EquiNodes = -5:5
6 peq = coef(lm(runge(EquiNodes)~EquiNodes+I(EquiNodes
      ^2)+I(EquiNodes^3)+I(EquiNodes^4)+I(EquiNodes^5)+
      I(EquiNodes^6)+I(EquiNodes^7)+I(EquiNodes^8)+I(
      EquiNodes^9)+I(EquiNodes^10)))
7
8 x = -5:5
9 y = runge(x)
10 lm10 = lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)+I(x^6)+I(x
      ^7)+I(x^8)+I(x^9)+I(x^10))
11 xplot=seq(from = -5,to = 5,by = .01)
12 plot(x,y,asp = .33)
13 lines(x,y)
14 grid(10,10)
15 lines(xplot,predict(lm10,newdata=data.frame(x=xplot)
      ), col="blue")
16
17 ChebNodes = 5*cos(pi*(11 - (1:11) + 0.5)/11)
```

```

18 pcheb = coef(lm(runge(ChebNodes)~ChebNodes+I(
    ChebNodes^2)+I(ChebNodes^3)+I(ChebNodes^4)+I(
    ChebNodes^5)+I(ChebNodes^6)+I(ChebNodes^7)+I(
    ChebNodes^8)+I(ChebNodes^9)+I(ChebNodes^10)))
19
20 x = 5*cos(pi*(11 - (1:11) + 0.5)/11)
21 y = runge(x)
22 plot(x,y,asp = 1)
23 lines(x,y)
24 lm10 = lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)+I(x^6)+I(x
    ^7)+I(x^8)+I(x^9)+I(x^10))
25 xplot=seq(from = -5,to = 5,by = .01)
26 lines(xplot,predict(lm10,newdata=data.frame(x=xplot)
    ), col="blue")

```

R code Exa 3.18 Interpolation by cubic splines

```

1 require(pracma)
2
3 x=1: 10
4 y<-c(8, 2.5, -2, 0, 5, 2, 4, 7 ,4.5, 2)
5 plot(x,y)
6 x2<-seq(from = 1, to = 10, by = 0.05)
7 y2=interp1(x,y,x2, 'spline')
8 lines(x2,y2)
9
10 x=1: 10
11 y<-c(8, 2.5, -2, 0, 5, 2, 4, 7 ,4.5, 2)
12 plot(x,y)
13 pp=cubicspline (x, y)
14 x2<-seq(from = 1, to = 10, by = 0.05)
15 y2 = ppval(pp,x2)
16 lines(x2,y2)
17
18 runge <- function(x) {

```

```

19     1/(1+25*x^2)
20 }
21
22 # use 11 equispaced nodes
23 EquiNodes11 = -5:5
24 ppeq11 = cubicspline(EquiNodes11,runge(EquiNodes11))
25 xplot=seq(from = -5, to = 5, by = 0.01)
26
27 # use 20 equispaced nodes
28 EquiNodes20 = linspace(-5,5,20)
29 ppeq20 = cubicspline(EquiNodes20,runge(EquiNodes20))
30 xplot=seq(from = -5, to = 5, by = 0.01)
31
32 # use 21 equispaced nodes
33 EquiNodes21 = linspace(-5,5,21)
34 ppeq21 = cubicspline(EquiNodes21,runge(EquiNodes21))
35 xplot=seq(from = -5, to = 5, by = 0.01)
36
37 par(mfrow=c(3,1))
38 plot(EquiNodes11,runge(EquiNodes11),asp = 1)
39 lines(EquiNodes11,runge(EquiNodes11))
40 lines(xplot,ppval(ppeq11,xplot), col="blue")
41 plot(EquiNodes20,runge(EquiNodes20),asp = 1)
42 lines(EquiNodes20,runge(EquiNodes20))
43 lines(xplot,ppval(ppeq20,xplot), col="blue")
44 plot(EquiNodes21,runge(EquiNodes21),asp = 1)
45 lines(EquiNodes21,runge(EquiNodes21))
46 lines(xplot,ppval(ppeq21,xplot), col="blue")

```

R code Exa 3.22 Bisection method

```

1 f <- function(x) {
2   1/x
3 }
4 ans = uniroot(f,c(-1,1),tol=2^-52)

```

```

5 ans$root
6 ans$f.root
7
8 #z<-curve(f)
9
10 f <- function(x) {
11   x^2
12 }
13 ans2 = uniroot(f,c(-1,1),tol=2^-52)
14 ans2$root
15 ans2$f.root

```

R code Exa 3.23 Newtons method

```

1 require(OptionPricing)
2 c=BS_EC(K=54, r = 0.07, sigma = 0.3, T = 5/12, S0 =
   50)
3 c[1]
4 y <- function(y) (BS_EC(K=54, r = 0.07, sigma = y, T
   = 5/12, S0 = 50)[1]-2.846575)
5 h<-Vectorize(y)
6 uniroot(y,c(-1,1))
7
8 f <- Vectorize(function(y) (BS_EC(K=54, r = 0.07,
   sigma = y, T = 5/12, S0 = 50)[1]-2.846575))
9 curve(f)

```

R code Exa 3.24 Optimization based solution of non linear equations

```

1 require(pracma)
2 f <- function(x) x^3*exp(-x^2)
3 vx<-seq(-4,4,.05)
4 plot(vx,f(vx))

```



```
5 lines(vx,f(vx))
6 fsolve(f,1)
7 fsolve(f,2)
8
9 #fsolve answers defer from matlab \(book\)
```

Chapter 4

Numerical Integration Deterministic and Monte Carlo Methods

R code Exa 4.1 Numerical integration in MATLAB

```
1 #require(pracma)
2 f <- function(x) {
3   exp(-x)*sin(10*x)
4 }
5
6 integrate(f ,0,2*pi)
7
8 integrate(f ,0,2*pi,abs.tol = 10e-6)
9
10 integrate(f ,0,2*pi,abs.tol = 10e-8)
11
12 quadl(f ,0,2*pi)
```

R code Exa 4.2 MONTE CARLO INTEGRATION

```

1 set.seed(12345)
2 mean(exp(runif(10)))
3 mean(exp(runif(10)))
4 mean(exp(runif(10)))
5 mean(exp(runif(1000000)))
6 mean(exp(runif(1000000)))
7 mean(exp(runif(1000000)))

```

R code Exa 4.3 MONTE CARLO INTEGRATION

```

1 BlsMC1 <- function(S0, K , r , T , sigma, NRepl) {
2   nuT = (r - 0.5*sigma**2)*T
3   siT = sigma * sqrt(T)
4   DiscPayoff = exp(-r*T) *pmax(0,(S0*exp(nuT+siT*
5     rnorm(NRepl))-K))
6   Price = mean(DiscPayoff)
7   return(Price)
8 }
9 S0=50
10 K=60
11 r=0.05
12 T=1
13 sigma=0.2
14 set.seed(547)
15 BlsMC1(S0,K,r ,T, sigma, 1000)
16 BlsMC1(S0,K,r ,T, sigma, 1000)
17 BlsMC1(S0,K,r ,T, sigma, 1000)
18 BlsMC1(S0,K,r ,T, sigma, 1000000)
19 BlsMC1(S0,K,r ,T, sigma, 1000000)
20 BlsMC1(S0,K,r ,T, sigma, 1000000)

```

R code Exa 4.4 Generating pseudorandom numbers

```

1 LCG <- function(a,c,m,seed,N) {
2   ZSeq <-matrix(0,N,1)
3   USeq <-matrix(0,N,1)
4   for (i in 1:N){
5     seed = (a*seed+c) %% m
6     ZSeq[i] = seed
7     USeq[i] = seed/m
8   }
9   result<-list(ZSeq,USeq)
10  return(result)
11 }
12
13 a = 5
14 c = 3
15 m = 16
16 seed = 7
17 N = 20
18 LCG(a,c,m,seed,N)

```

R code Exa 4.5 Generating pseudorandom numbers

```

1 LCG <- function(a,c,m,seed,N) {
2   ZSeq <-matrix(0,N,1)
3   USeq <-matrix(0,N,1)
4   for (i in 1:N){
5     seed = (a*seed+c) %% m
6     ZSeq[i] = seed
7     USeq[i] = seed/m
8   }
9   result<-list(ZSeq,USeq)
10  return(result)
11 }
12
13 m = 2048;
14 a = 65;

```

```

15 c = 1;
16 seed = 0;
17 U = LCG(a,c,m,seed, 2048);
18 plot(unlist(U[1])[1:m-1],unlist(U[2])[2:m])
19 plot(unlist(U[1])[1:511],unlist(U[2])[2:512])
20
21 a=1365;
22 c=1 ;
23 U = LCG(a,c,m,seed, 2048)
24 plot(unlist(U[1])[1:m-1],unlist(U[2])[2:m])

```

R code Exa 4.6 Inverse transform method

```

1 set.seed(64657)
2 rexp(1, 1/1)
3
4 EmpiricalDrnd <- function(values, probs, howmany) {
5   cumprobs = cumsum(probs)
6   N = length(probs)
7   samples = matrix(0,howmany, 1)
8   for (k in 1:howmany){
9     loc=sum(runif(1)*cumprobs[N] > cumprobs) + 1;
10    samples[k]=values[loc]
11  }
12  return(samples)
13 }
14
15 values=1:5
16 probs<-c(0.1, 0.2, 0.4, 0.2, 0.1)
17 samples=EmpiricalDrnd(values ,probs ,10000)
18 hist(x = samples)

```

R code Exa 4.8 Generating normal variates by the polar approach

```

1 LCG <- function(a,c,m,seed,N) {
2   ZSeq <-matrix(0,N,1)
3   USeq <-matrix(0,N,1)
4   for (i in 1:N){
5     seed = (a*seed+c) %% m
6     ZSeq[i] = seed
7     USeq[i] = seed/m
8   }
9   result<-list(ZSeq,USeq)
10  return(result)
11 }
12
13 m = 2048;
14 a = 1229;
15 c = 1;
16 N = m-2;
17 seed = 0;
18 U = LCG(a,c,m,seed,N);
19 index<-seq(1,N-1,2)
20 U1 = unlist(U[2])[index]
21 index<-seq(2,N,2)
22 U2 = unlist(U[2])[index]
23 X=sqrt(-2*log(U1))* cos(2*pi*U2);
24 Y=sqrt(-2*log(U1))* sin(2*pi*U2);
25 plot(X,Y)
26 X=sqrt(-2*log(U2))* cos(2*pi*U1);
27 Y=sqrt(-2*log(U2))* sin(2*pi*U1);
28 plot(X,Y)

```

R code Exa 4.9 Generating normal variates by the polar approach

```

1 Sigma<-matrix(c(4, 1, -2, 1, 3, 1, -2, 1, 5),nrow =
2   3,ncol = 3,byrow = TRUE)
3 mu<-matrix(c(8,6,10),nrow = 3,ncol = 1,byrow = TRUE)
4 eigen(Sigma)$values

```

```

4 set.seed(2392)
5
6 MultiNormrnd <- function(mu,sigma,howmany) {
7   n = length(mu)
8   Z = matrix(0,howmany,n)
9   U = chol(sigma)
10  for (i in 1:howmany){
11    Z[i,] = t(mu) + t(matrix(rnorm(n))) %*% U
12  }
13  return(Z)
14 }
15
16 Z = MultiNormrnd(mu,Sigma,10000)
17 mean(Z[1])
18 mean(Z[2])
19 mean(Z[3])
20 cov(Z)

```

R code Exa 4.10 SETTING THE NUMBER OF REPLICATIONS

```

1 #require(fBasics)
2 norm.interval = function(data, variance = var(data),
3   conf.level = 0.95) {
4   z = qnorm((1 - conf.level)/2, lower.tail =
5     FALSE)
6   xbar = mean(data)
7   sdx = sqrt(variance/length(data))
8   c(xbar - z * sdx, xbar + z * sdx)
9 }
10
11 BlsMC2 <- function(S0,K,r,T,sigma,NRepl) {
12   nuT = (r - 0.5*sigma^2)*T;
13   siT = sigma * sqrt(T);
14   DiscPayoff = exp(-r*T)*pmax(0, S0*exp(nuT+siT*
15     rnorm(NRepl))-K);

```

```

13   parameter_estimation<-.normFit(DiscPayoff)
14   ci<-norm.interval(DiscPayoff)
15   final<-list(parameter_estimation,ci)
16   return(final)
17 }
18
19 set.seed(483762)
20 S0=50;
21 K=55;
22 r=0.05;
23 T=5/12;
24 sigma=0.2;
25 answer<-Blsmc2(S0,K,r,T,sigma,50000)
26 answer2<-Blsmc2(S0,K,r,T,sigma,1000000)
27 #(answer[[2]][2] - answer[[2]][1])/1.168224
28 #(answer2[[2]][2] - answer2[[2]][1])/1.173184

```

R code Exa 4.11 Antithetic sampling

```

1 #require(fBasics)
2 norm.interval = function(data, variance = var(data),
3   conf.level = 0.95) {
4   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
5   xbar = mean(data)
6   sdx = sqrt(variance/length(data))
7   c(xbar - z * sdx, xbar + z * sdx)
8 }
9 set.seed(337282)
10 X=exp(runif(100))
11 ci<-norm.interval(X)
12 #from parameter estimation, value of mean =
13   1.7031094
14 (ci[2]-ci[1])/1.7031094

```



```

15 set.seed(3374573)
16 U1=runif(50)
17 U2 = 1 - U1
18 X = 0.5*(exp(U1)+exp(U2))
19 parameter_estimation<-.normFit(X)
20 ci<-norm.interval(X)
21 #from parameter estimation, value of mean =
    1.73413846
22 (ci[2]-ci[1])/1.73413846

```

R code Exa 4.12 BlsMCAV

```

1 suppressMessages(require(fBasics))
2 norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
3   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
4   xbar = mean(data)
5   sdx = sqrt(variance/length(data))
6   c(xbar - z * sdx, xbar + z * sdx)
7 }
8
9 BlsMCAV <- function(S0,K,r,T,sigma,NPairs) {
10   nuT = (r - 0.5*sigma^2)*T;
11   siT = sigma * sqrt(T);
12   Veps = rnorm(NPairs);
13   Payoff1 = pmax( 0 , S0*exp(nuT+siT*Veps) - K);
14   Payoff2 = pmax( 0 , S0*exp(nuT+siT*(-Veps)) - K);
15   DiscPayoff = exp(-r*T) * 0.5 * (Payoff1+Payoff2);
16   parameter_estimation<-.normFit(DiscPayoff)
17   ci<-norm.interval(DiscPayoff)
18   answer<-list(parameter_estimation,ci)
19   return(answer)
20 }
21
22 BlsMC2 <- function(S0,K,r,T,sigma,NRepl) {

```

```

23   nuT = (r - 0.5*sigma^2)*T;
24   siT = sigma * sqrt(T);
25   DiscPayoff = exp(-r*T)*pmax(0, S0*exp(nuT+siT*
      rnorm(NRepl))-K);
26   parameter_estimation<-.normFit(DiscPayoff)
27   ci<-norm.interval(DiscPayoff)
28   final<-list(parameter_estimation,ci)
29   return(final)
30 }
31
32 MCButterfly <- function(S0,r,T,sigma,NRepl,K1,K2,K3)
    {
33   nuT = (r-0.5*sigma^2)*T;
34   siT = sigma*sqrt(T);
35   Veps = rnorm(NRepl);
36   Stocks = S0*exp(nuT + siT*Veps);
37   In1 = which((Stocks > K1) & (Stocks < K2));
38   In2 = which((Stocks >= K2) & (Stocks < K3));
39   Payoff = exp(-r*T)*matrix(c((Stocks[In1]-K1), (K3
      -Stocks[In2]), matrix(0,(NRepl - length(In1) -
      length(In2)),1))),
40   parameter_estimation<-.normFit(Payoff)
41   ci<-norm.interval(Payoff)
42   final<-list(parameter_estimation,ci)
43   return(final)
44 }
45
46 MCAVButterfly <- function(S0,r,T,sigma,NPairs,K1,K2,
    K3) {
47   nuT = (r-0.5*sigma^2)*T;
48   siT = sigma*sqrt(T);
49   Veps = rnorm(NPairs);
50   Stocks1 = S0*exp(nuT + siT*Veps);
51   Stocks2 = S0*exp(nuT - siT*Veps);
52   Payoff1 = matrix(0,NPairs,1);
53   Payoff2 = matrix(0,NPairs,1);
54   In = which((Stocks1 > K1) & (Stocks1 < K2));
55   Payoff1[In] = (Stocks1[In] - K1);

```

```

56   In = which((Stocks1 >= K2) & (Stocks1 < K3));
57   Payoff1[In] = (K3 - Stocks1[In]);
58   In = which((Stocks2 > K1) & (Stocks2 < K2));
59   Payoff2[In] = (Stocks2[In] - K1);
60   In = which((Stocks2 >= K2) & (Stocks2 < K3));
61   Payoff2[In] = (K3 - Stocks2[In]);
62   Payoff = 0.5 * exp(-r*T) * (Payoff1 + Payoff2);
63   parameter_estimation<-.normFit(Payoff)
64   ci<-norm.interval(Payoff)
65   final<-list(parameter_estimation,ci)
66   return(final)
67 }
68
69 set.seed(3374)
70 Y<-BlMC2(50,50 ,0.05,1, 0.4,200000)
71 #from parameter estimation, value of mean = 9.11143
72 (Y[[2]][2]-Y[[2]][1])/9.11143
73
74 Z<-BlMCAV(50,50,0.05,1,0.4,100000)
75 #from parameter estimation, value of mean = 9.020972
76 (Z[[2]][2]-Z[[2]][1])/9.020972
77
78 set.seed(39489378)
79 S0 = 60;
80 K1 = 55;
81 K2 = 60;
82 K3 = 65;
83 T = 5/12;
84 r = 0.1;
85 sigma = 0.4;
86 a<-MCButterfly(S0,r,T,sigma,100000,K1,K2,K3);
87 #from parameter estimation, value of mean =
    0.6104167
88 (a[[2]][2]-a[[2]][1])/0.6104167
89
90 set.seed(72725)
91 b<-MCAVButterfly(S0,r ,T, sigma,50000,K1 ,K2,K3)
92 #from parameter estimation, value of mean =

```

```
0.6154077
93 (b[[2]][2]-b[[2]][1])/0.6154077
```

R code Exa 4.14 Importance sampling

```
1 estpi <- function(m) {
2   z=sqrt(1-runif(m)^2) ;
3   out = 4*sum(z)/m;
4   return(out)
5 }
6 set.seed(483272)
7 estpi(1000)
8 estpi(1000)
9 estpi(1000)
10
11 estpiIS <- function(m,L) {
12   s= seq(0,1-1/L,1/L) + 1/(2*L)
13   hvals = matrix(sqrt(1 - s^2))
14   # get cumulative probabilities
15   cs=apply(hvals,2,cumsum);
16   est = matrix(0,m)
17   for (j in 1:m){
18     # locate sub-interval
19     loc=sum(runif(1)*cs[L] > cs) +1;
20     # sample uniformly within sub-interval
21     x=(loc-1)/L + runif(1)/L;
22     p=hvals[loc]/cs[L];
23     est[j] = sqrt(1 - x^2)/(p*L);
24   }
25   plot(est)
26   z = 4*sum(est)/m;
27   return(z)
28 }
29
30 estpiIS(1000,10)
```

```

31 estpiIS(1000,10)
32 estpiIS(1000,10)
33 estpiIS(1000,100)
34 estpiIS(1000,100)
35 estpiIS(1000,100)

```

R code Exa 4.15 Generating Halton low discrepancy sequences

```

1 #require(fBasics)
2 norm.interval = function(data, variance = var(data),
   conf.level = 0.95) {
3   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
4   xbar = mean(data)
5   sdx = sqrt(variance/length(data))
6   c(xbar - z * sdx, xbar + z * sdx)
7 }
8
9 BlsMCIS <- function(S0,K,r,T,sigma,NRepl) {
10   nuT = (r - 0.5*sigma^2)*T;
11   siT = sigma * sqrt(T);
12   ISnuT = log(K/S0) - 0.5*sigma^2*T;
13   Veps = rnorm(NRepl);
14   VY = ISnuT + siT*Veps;
15   ISRatios = exp( (2*(nuT - ISnuT)*VY - nuT^2 +
   ISnuT^2)/2/siT^2);
16   DiscPayoff = exp(-r*T)*pmax(0, (S0*exp(VY)-K));
17   parameter_estimation<-normFit(DiscPayoff*ISRatios
   )
18   ci<-norm.interval(DiscPayoff*ISRatios)
19   final<-list(parameter_estimation,ci)
20   return(final)
21 }
22
23 Halton <- function(n,b) {
24   n0 = n;

```

```

25   h = 0;
26   f = 1/b;
27   while (n0 > 0){
28     n1 = floor(n0/b);
29     r = n0 - n1*b;
30     h = h+f*r;
31     f = f/b;
32     n0=n1;
33   }
34   return(h)
35 }
36
37 seq = matrix(0,10)
38 for (i in 1:10){
39   seq[i] = Halton(i,2);
40 }
41
42 GetHalton <- function(HowMany, Base) {
43   Seq = matrix(0,HowMany,1)
44   NumBits = 1+round(log(HowMany)/log(Base));
45   VetBase = Base^(-(1:NumBits));
46   WorkVet = matrix(0,1,NumBits);
47   for (i in 1:HowMany){
48     j = 1;
49     ok = 0;
50     while (ok == 0){
51       WorkVet[j] = WorkVet[j]+1;
52       if (WorkVet[j] < Base){
53         ok = 1;
54       }
55       else{
56         WorkVet[j] = 0;
57         j = j+1;
58       }
59     }
60     Seq[i] = sum(WorkVet * VetBase)
61   }
62   return(Seq)

```

```

63 }
64
65
66
67 plot(runif(100) ,runif(100))
68 plot(GetHalton(100,2) ,GetHalton (100,7))
69 plot(GetHalton(100,2), GetHalton(100,4))

```

R code Exa 4.16 Generating Halton low discrepancy sequences

```

1  require(pracma)
2  f <- function(x,y) {
3    exp(-x*y) *(sin(6*pi*x)+cos(8*pi*y))
4  }
5
6  dblquad(f = f,xa = 0,xb = 1,ya = 0,yb = 1)
7
8  n <- seq(0,1,0.01)
9  multiarray = list();
10 multiarray <- meshgrid(n,n)
11 Z<-f(multiarray$X,multiarray$Y)
12
13
14 persp(multiarray$X[1,],multiarray$Y[,1],Z,theta=30,
        phi=30, expand=0.6,col='lightblue', shade=0.75,
        ltheta=120,ticktype='detailed')
15
16 set.seed(4837)
17 mean(f(runif(10000),runif(10000)))
18 mean(f(runif(10000),runif(10000)))
19 mean(f(runif(10000),runif(10000)))
20
21 GetHalton <- function(HowMany, Base) {
22   Seq = matrix(0,HowMany,1)
23   NumBits = 1+round(log(HowMany)/log(Base));

```

```

24 VetBase = Base^(-(1:NumBits));
25 WorkVet = matrix(0,1,NumBits);
26 for (i in 1:HowMany){
27     j = 1;
28     ok = 0;
29     while (ok == 0){
30         WorkVet[j] = WorkVet[j]+1;
31         if (WorkVet[j] < Base){
32             ok = 1;
33         }
34         else{
35             WorkVet[j] = 0;
36             j = j+1;
37         }
38     }
39     Seq[i] = sum(WorkVet * VetBase)
40 }
41 return(Seq)
42 }
43
44 seq2 = GetHalton(10000,2)
45 seq4 = GetHalton(10000,4)
46 seq5 = GetHalton(10000,5)
47 seq7 = GetHalton(10000,7)
48 mean(f(seq2,seq5))
49 mean(f(seq2,seq4))
50 mean(f(seq2,seq7))
51 mean(f(seq5,seq7))
52
53 set.seed(327439)
54 mean(f(runif(100),runif(100)))
55 mean(f(runif(500),runif(500)))
56 mean(f(runif(1000),runif(1000)))
57 mean(f(runif(1500),runif(1500)))
58 mean(f(runif(2000),runif(2000)))
59
60 mean(f(seq2[1:100],seq7[1:100]))
61 mean(f(seq2[1:500],seq7[1:500]))

```



```

62 mean(f(seq2[1:1000], seq7[1:1000]))
63 mean(f(seq2[1:1500], seq7[1:1500]))
64 mean(f(seq2[1:2000], seq7[1:2000]))

```

R code Exa 4.17 Generating Halton low discrepancy sequences

```

1  require(pracma)
2  require(fBasics)
3  norm.interval = function(data, variance = var(data),
   conf.level = 0.95) {
4    z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
5    xbar = mean(data)
6    sdx = sqrt(variance/length(data))
7    c(xbar - z * sdx, xbar + z * sdx)
8  }
9
10 GetHalton <- function(HowMany, Base) {
11   Seq = matrix(0, HowMany, 1)
12   NumBits = 1 + round(log(HowMany)/log(Base));
13   VetBase = Base^(-(1:NumBits));
14   WorkVet = matrix(0, 1, NumBits);
15   for (i in 1:HowMany){
16     j = 1;
17     ok = 0;
18     while (ok == 0){
19       WorkVet[j] = WorkVet[j]+1;
20       if (WorkVet[j] < Base){
21         ok = 1;
22       }
23       else{
24         WorkVet[j] = 0;
25         j = j+1;
26       }
27     }
28     Seq[i] = sum(WorkVet * VetBase)

```

```

29   }
30   return(Seq)
31 }
32
33 BlsHaltonBM <- function(S0,K,r,T,sigma,NPoints,Base1
    ,Base2) {
34   nuT = (r - 0.5*sigma^2)*T;
35   siT = sigma * sqrt(T);
36   H1 = GetHalton(ceiling(NPoints/2),Base1);
37   H2 = GetHalton(ceiling(NPoints/2),Base2);
38   VLog = sqrt(-2*log(H1))
39   Norm1 = VLog * cos(2*pi*H2)
40   Norm2 = VLog * sin(2*pi*H2)
41   Norm = rbind(Norm1,Norm2)
42   DiscPayoff = exp(-r*T) * pmax( 0 , S0*exp(nuT+siT*
    Norm) - K);
43   Price = mean(DiscPayoff);
44   return(Price)
45 }
46
47 BlsHaltonBM(50,52,0.1,5/12,0.4,5000,2,7)
48 BlsHaltonBM(50,52,0.1,5/12,0.4,5000,11,7)
49 BlsHaltonBM(50,52,0.1,5/12,0.4,5000,2,4)
50
51 BlsMC2 <- function(S0,K,r,T,sigma,NRepl) {
52   nuT = (r - 0.5*sigma^2)*T;
53   siT = sigma * sqrt(T);
54   DiscPayoff = exp(-r*T)*pmax(0, S0*exp(nuT+siT*
    rnorm(NRepl))-K);
55   parameter_estimation<-normFit(DiscPayoff)
56   ci<-norm.interval(DiscPayoff)
57   final<-list(parameter_estimation,ci)
58   return(final)
59 }
60
61 set.seed(3726)
62 BlsMC2(50,52,0.1,5/12,0.4,5000)
63 BlsMC2(50,52,0.1,5/12,0.4,5000)

```

```

64 BlsMC2(50,52,0.1,5/12,0.4,5000)
65
66 BlsHaltonINV <- function(S0,X,r,T,sigma,NPoints,Base
    ) {
67     nuT = (r - 0.5*sigma^2)*T;
68     siT = sigma * sqrt(T);
69     # Use inverse transform to generate standard
        normals
70     H = GetHalton(NPoints,Base);
71     Veps = qnorm(H);
72     DiscPayoff = exp(-r*T)*pmax(0,S0*exp(nuT+siT*Veps)
        -X);
73     Price = mean(DiscPayoff);
74     return(Price)
75 }
76
77 BlsHaltonINV(50,52,0.1,5/12,0.4,1000,2)
78 BlsHaltonINV(50,52,0.1,5/12,0.4,2000,2)
79 BlsHaltonINV(50,52,0.1,5/12,0.4,5000,2)
80 BlsHaltonINV(50,52,0.1,5/12,0.4,1000,2)
81 BlsHaltonINV(50,52,0.1,5/12,0.4,10000,2)
82 BlsHaltonINV(50,52,0.1,5/12,0.4,50000,2)
83
84 GetHalton(17,17)
85
86 BlsHaltonINV(50,52,0.1,5/12,0.4,1000,499)
87 BlsHaltonINV(50,52,0.1,5/12,0.4,2000,499)
88 BlsHaltonINV(50,52,0.1,5/12,0.4,5000,499)
89 BlsHaltonINV(50,52,0.1,5/12,0.4,10000,499)
90 BlsHaltonINV(50,52,0.1,5/12,0.4,50000,499)
91
92 plot(GetHalton(1000,109), GetHalton(1000,113))

```

R code Exa 4.18 Generating Sobol low discrepancy sequences

```

1 require(bitops)
2 GetDirNumbers <- function(p,m0,n) {
3   degree = length(p)-1;
4   p = p[2:degree];
5   m = cbind(m0 , matrix(0,1,n-degree))
6   for (i in (degree+1):n){
7     m[i] = bitXor(m[i-degree], 2^degree * m[i-degree
8       ])
9     for (j in 1:(degree-1)){
10      m[i] = bitXor(m[i], 2^j * p[j] * m[i-j]);
11    }
12  }
13  v=m/(2^(1:length(m)))
14  final<-list()
15  final$v<-v
16  final$m<-m
17  return(final)
18 }
19 p <-matrix(c(1,0, 1, 1),nrow = 1,ncol = 4)
20 m0 <-matrix(c(1,3, 7),nrow = 1,ncol = 3)
21 ans<-GetDirNumbers(p,m0, 6)
22 ans$v
23 ans$m

```

R code Exa 4.19 Generating Sobol low discrepancy sequences

```

1 require(bitops)
2 gray<-function(x) bitXor(x,bitShiftR(x,1))
3 codes = matrix(0,16,4);
4 for (i in 1:16){
5   print(intToBits(gray(i-1)) [4:1])
6 }

```

R code Exa 4.20 Generating Sobol low discrepancy sequences

```
1 require(bitops)
2 options(warn=-1)
3
4 GetDirNumbers <- function(p,m0,n) {
5   degree = length(p)-1;
6   p = p[2:degree];
7   m = cbind(m0 , matrix(0,1,n-degree))
8   for (i in (degree+1):n){
9     m[i] = bitXor(m[i-degree], 2^degree * m[i-degree
10      ])
11     for (j in 1:(degree-1)){
12       m[i] = bitXor(m[i], 2^j * p[j] * m[i-j]);
13     }
14   }
15   v=m/(2^(1:length(m)))
16   final<-list()
17   final$v<-v
18   final$m<-m
19   return(final)
20 }
21 p <-matrix(c(1,0, 1, 1),nrow = 1,ncol = 4)
22 m0 <-matrix(c(1,3, 7),nrow = 1,ncol = 3)
23 ans<-GetDirNumbers(p,m0, 6)
24 ans$v
25 ans$m
26
27 GetSobol <- function(GenNumbers, x0, HowMany) {
28   Nbits = 20;
29   factor = 2^Nbits;
30   BitNumbers = GenNumbers * factor;
31   SobSeq = matrix(0,HowMany + 1, 1);
```

```

32  SobSeq[1] = as.integer(x0*factor);
33  for (i in 1:HowMany){
34      c = pmin(which( intToBits(i-1) [1:16] == 0 ));
35      SobSeq[i+1] = bitXor(SobSeq[i], BitNumbers[c]);
36  }
37  SobSeq = SobSeq / factor;
38  return(SobSeq)
39 }
40 GetSobol(ans$v,0,10)
41
42
43 p <-matrix(c(1,0, 1, 1,1,1),nrow = 1,ncol = 6)
44 m0 <-matrix(c(1,3, 5, 9, 11),nrow = 1,ncol = 5)
45 ans<-GetDirNumbers(p,m0, 10)
46
47 GetSobol(ans$v,0.124,10)

```

Chapter 5

Finite Difference Methods for Partial Differential Equations

R code Exa 5.1 Instability in a finite difference scheme

```
1 require(PopED)
2
3 transport <- function(xmin, dx, xmax, dt, tmax, c,
4   f0) {
5   N = ceiling((xmax - xmin) / dx);
6   xmax = xmin + N*dx;
7   M = ceiling(tmax/dt);
8   k1 = 1 - dt*c/dx;
9   k2 = dt*c/dx;
10  solution = matrix(0,N+1,M+1);
11  vetx = seq(xmin,xmax,dx)
12  for (i in 1:N+1) {
13    solution[i,1] = feval(f0,vetx[i]);
14  }
15  fixedvalue = solution[1,1];
16  # this is needed because of finite domain
17  plot(solution[,1])
18  for (j in 1:M){
19    solution[,j+1] = k1*solution[,j] + k2* c(
```

```

        fixedvalue,solution[1:N,j]));
19     lines(solution[,j])
20   }
21   return(solution)
22 }
23
24 f0transp <- function(x) {
25   if(x < (-1)){
26     y = 0
27   } else if (x <= 0){
28     y=x+1;
29   } else{
30     y = 1;
31   }
32 }
33
34 xmin = -2;
35 xmax = 3;
36 dx = 0.05;
37 tmax = 2;
38 dt = 0.01;
39 c = 1;
40 sol = transport(xmin, dx, xmax, dt, tmax, c ,
    f0transp)
41
42 TransportPlot <- function(xmin, dx, xmax, times, sol
    ) {
43   par(mfrow=c(2,2))
44   plot(seq(xmin,xmax,dx), sol[,times[1]])
45   lines(seq(xmin,xmax,dx), sol[,times[1]])
46   plot(seq(xmin,xmax,dx), sol[,times[2]])
47   lines(seq(xmin,xmax,dx), sol[,times[2]])
48   plot(seq(xmin,xmax,dx), sol[,times[3]])
49   lines(seq(xmin,xmax,dx), sol[,times[3]])
50   plot(seq(xmin,xmax,dx), sol[,times[4]])
51   lines(seq(xmin,xmax,dx), sol[,times[4]])
52 }
53

```



```

54 TransportPlot(xmin, dx, xmax, c(1, 51, 101, 201),
    sol)
55
56 dx = 0.01
57 sol = transport(xmin, dx, xmax, dt, tmax, c ,
    f0transp)
58 TransportPlot(xmin, dx, xmax, c(1, 51, 101, 201),
    sol)
59
60 #Blow-outs
61 dx = 0.005
62 sol = transport(xmin, dx, xmax, dt, tmax, c ,
    f0transp)
63 TransportPlot(xmin, dx, xmax, c(1, 51, 101, 201),
    sol)

```

R code Exa 5.3 Solving the heat equation by an explicit method

```

1 HeatExpl <- function(deltax, deltat, tmax) {
2   N = round(1/deltax)
3   M = round(tmax/deltat)
4   sol = matrix(0,N+1,M+1)
5   rho = deltat / (deltax)^2
6   rho2 = 1-2*rho
7   vetx = seq(0,1,deltax)
8   for (i in 2:ceiling((N+1)/2)){
9     sol[i,1] = 2*vetx[i]
10    sol[N+2-i,1] = sol[i,1]
11  }
12  for (j in 1:M){
13    for (i in 2:N){
14      sol[i,j+1] = rho*sol[i-1,j] + rho2*sol[i,j] +
        rho*sol[i+1,j]
15    }
16  }

```

```

17   return(sol)
18 }
19
20 dx = 0.1;
21 dt = 0.001;
22 tmax = dt*100;
23 sol=HeatExpl(dx, dt , tmax)
24
25 par(mfrow=c(2,2))
26 plot(seq(0,1,dx), sol[,1])
27 lines(seq(0,1,dx), sol[,1])
28 plot(seq(0,1,dx), sol[,11])
29 lines(seq(0,1,dx), sol[,11])
30 plot(seq(0,1,dx), sol[,51])
31 lines(seq(0,1,dx), sol[,51])
32 plot(seq(0,1,dx), sol[,101])
33 lines(seq(0,1,dx), sol[,101])

```

R code Exa 5.4 Solving the heat equation by a fully implicit method

```

1 HeatImpl <- function(deltax, deltat, tmax) {
2   N = round(1/deltax)
3   M = round(tmax/deltat)
4   sol = matrix(0,N+1,M+1)
5   rho = deltat / (deltax)^2
6   B = diag(c((1+2*rho) * array(1,c(N-1,1)))) - diag(
7     c(0,rho*array(1,c(N-1,1))))[2:100,1:99] - diag(
8     c(0,rho*array(1,c(N-1,1))))[1:99,2:100]
9   vetx = seq(0,1,deltax)
10  for (i in 2:ceiling((N+1)/2)){
11    sol[i,1] = 2*vetx[i]
12    sol[N+2-i,1] = sol[i,1]
13  }
14  for (j in 1:M){
15    sol[2:N,j+1] = solve(B,sol[2:N,j])
16  }
17 }

```

```
14     }
15     return(sol)
16 }
17
18 deltax=dx=0.01
19 deltat=dt=0.001
20 tmax=dt*100
21 sol=HeatImpl(dx,dt,tmax)
22
23 par(mfrow=c(2,2))
24 plot(seq(0,1,dx),sol[,1])
25 lines(seq(0,1,dx),sol[,1])
26 plot(seq(0,1,dx),sol[,11])
27 lines(seq(0,1,dx),sol[,11])
28 plot(seq(0,1,dx),sol[,51])
29 lines(seq(0,1,dx),sol[,51])
30 plot(seq(0,1,dx),sol[,101])
31 lines(seq(0,1,dx),sol[,101])
```

Chapter 6

Convex Optimization

R code Exa 6.1 Finite vs infinite dimensional problems

```
1 require(signal)
2 require("ucminf")
3
4 g <- function(x) {
5   polyval(c(1, -10.5, 39, -59.5, 30), x)
6 }
7 xvet=seq(1,4,0.05)
8 plot(xvet,g(xvet))
9 lines(xvet,g(xvet))
10
11 x<-ucminf(c(0),g)$par
12 x
13 fval<-ucminf(c(0),g)$value
14 fval
15
16 x<-ucminf(c(5),g)$par
17 x
18 fval<-ucminf(c(5),g)$value
19 fval
20
21 f <- function(x) {
```

```

22 polyval(c(1, -8, 22, -24, 1), x)
23 }
24 xvet=seq(0,4,0.05)
25 plot(xvet,f(xvet))
26 lines(xvet,f(xvet))

```

R code Exa 6.3 Linear vs non linear problems

```

1 require(lpSolve)
2 f.obj <- c(2, 3, 3)
3 f.con <- matrix (c(1, 2, 0, 1, 0, 1, 1, 0, 0, 0, 1,
    0, 0, 0, 1), nrow=5, byrow=TRUE)
4 f.dir <- c("=", ">=", ">=", ">=", ">=")
5 f.rhs <- c(3, 3, 0, 0, 0)
6 lpSolve::lp(direction = "min",objective.in = f.obj,
    const.mat = f.con,const.dir = f.dir,const.rhs = f
    .rhs)

```

R code Exa 6.5 Penalty function approach

```

1 require(pracma)
2 f <- function(sigma,x,y) {
3   (x-1.5)^2+(y-0.5)^2+sigma/(1-x)+sigma/(1-y)
4 }
5 X<-meshgrid(x = seq(0.1,.99,.01))$X
6 Y<-meshgrid(x = seq(0.1,.99,.01))$Y
7 contour(f(0.1,X,Y))
8 contour(f(0.01,X,Y))
9 contour(f(0.001,X,Y))
10 contour(f(0.0001,X,Y))

```

R code Exa 6.8 Kuhn Tucker conditions

```
1 require(quadprog)
2 H = 2*eye(2)
3 f<-c(0,0)
4 Aeq<-matrix(c(1,1),nrow = 1)
5 beq<-4
6 lb<-matrix(c(0,3),ncol =1,nrow = 2)
7 quadprog(C = H,d = f,Aeq = Aeq,beq = beq,lb = lb)$
  xmin
```

R code Exa 6.12 Geometric and algebraic features of linear programming

```
1 A <- matrix(data=c(-1, 1, 1, -1, 0, 0, 1, 0, 4, 0,
  0, 0, 2, 2, 1), nrow=3, ncol=5, byrow=TRUE)
2 b <- matrix(data=c(1, 3, 1), nrow=3, ncol=1, byrow=
  FALSE)
3 asvd = svd(A)
4 adia = diag(1/asvd$d)
5 solution = asvd$v %*% adia %*% t(asvd$u) %*% b
6 Check = A %*% solution
7
8 # This solution does not match with the book,
9 # but in any case it's NOT FEASIBLE either as values
  present are less than ZERO
10
11 # Book solution check
12 X<-matrix(data = c(0,3,-2,0,5),nrow = 5,ncol = 1,
  byrow = TRUE)
13 A %*% X
```

Chapter 7

Option Pricing by Binomial and Trinomial Lattices

R code Exa 7.1 Calibrating a binomial lattice

```
1 LatticeEurCall <- function(S0,K,r,T,sigma,N) {
2   deltaT = T/N
3   u=exp(sigma * sqrt(deltaT))
4   d=1/u
5   p=(exp(r*deltaT) - d)/(u-d)
6   lattice = matrix(0,N+1,N+1)
7   for(i in 0:N){
8     lattice[i+1,N+1]=max(0,S0*(u^i)*(d^(N-i))-K)
9   }
10  for(j in (N-1):0){
11    for (i in 0:j){
12      lattice[i+1,j+1] = exp(-r*deltaT) * (p *
13        lattice[i+2,j+2] + (1-p) * lattice[i+1,j
14        +2])
15    }
16  }
17  return(lattice[1,1])
18 }
19 call=LatticeEurCall(50,50,0.1,5/12,0.4,5)
```

```

18 call
19 call=LatticeEurCall(50,50,0.1,5/12,0.4,500)
20 call

```

R code Exa 7.2 Accuracy of the binomial lattice for decreasing deltaT

```

1 require(OptionPricing)
2 LatticeEurCall <- function(S0,K,r,T,sigma,N) {
3   deltaT = T/N
4   u=exp(sigma * sqrt(deltaT))
5   d=1/u
6   p=(exp(r*deltaT) - d)/(u-d)
7   lattice = matrix(0,N+1,N+1)
8   for(i in 0:N){
9     lattice[i+1,N+1]=max(0,S0*(u^i)*(d^(N-i))-K)
10  }
11  for(j in (N-1):0){
12    for (i in 0:j){
13      lattice[i+1,j+1] = exp(-r*deltaT) * (p *
14        lattice[i+2,j+2] + (1-p) * lattice[i+1,j
15        +2])
16    }
17  }
18  return(lattice[1,1])
19 }
20 S0 = 50
21 K = 50
22 r = 0.1
23 sigma = 0.4
24 T = 5/12
25 N=50
26 BlsC = BS_EC(K = K, r = r, sigma = sigma, T = T, S0
27   = S0)['price']
28 LatticeC = matrix(0,1,N)
29 for (i in 1:N){

```



```

27   LatticeC[i] = LatticeEurCall(S0, K , r , T, sigma,
    i)
28 }
29 plot(1:N, matrix(1,1,N)*BlsC)
30 lines(1:N, matrix(1,1,N)*BlsC)
31 lines(1:N, LatticeC)

```

R code Exa 7.3 Price a pay later option by a binomial lattice

```

1  require(pracma)
2  L11 <- function(premium,S0,K,r,sigma,T,N) {
3    deltaT = T/N
4    u=exp(sigma * sqrt (deltaT))
5    d=1/u
6    p=(exp(r*deltaT) - d)/(u-d)
7    lattice = matrix(0,N+1,N+1)
8    for (i in 0:N){
9      if (S0*(u^i)*(d^(N-i)) >= K){
10        lattice[i+1,N+1]=S0*(u^i)*(d^(N-i)) - K -
          premium
11      }
12    }
13    for (j in (N-1):0){
14      for (i in 0:j){
15        lattice[i+1, j+1] = p*lattice[i+2, j+2] + (1-p
          )*lattice[i+1, j+2]
16      }
17    }
18    return(lattice[1,1])
19  }
20
21  f <- function(P) {
22    L11(premium = P,S0 = 12,K = 14,r = 0.1, sigma =
      0.2, T = 10/12, N = 10)
23  }

```

```
24 fzero(f = f,x = 2)$x
```

R code Exa 7.4 Pricing an European call by a binomial lattice

```
1 require(OptionPricing)
2 require(tictoc)
3 LatticeEurCall <- function(S0,K,r,T,sigma,N) {
4   deltaT = T/N
5   u=exp(sigma * sqrt(deltaT))
6   d=1/u
7   p=(exp(r*deltaT) - d)/(u-d)
8   lattice = matrix(0,N+1,N+1)
9   for(i in 0:N){
10     lattice[i+1,N+1]=max(0,S0*(u^i)*(d^(N-i))-K)
11   }
12   for(j in (N-1):0){
13     for (i in 0:j){
14       lattice[i+1,j+1] = exp(-r*deltaT) * (p *
15         lattice[i+2,j+2] + (1-p) * lattice[i+1,j
16         +2])
17     }
18   }
19   return(lattice[1,1])
20 }
21 SmartEurLattice <- function(S0,K,r,T,sigma,N) {
22   # Precompute invariant quantities
23   deltaT = T/N
24   u=exp(sigma * sqrt(deltaT))
25   d=1/u
26   p=(exp(r*deltaT) - d)/(u-d)
27   discount = exp(-r*deltaT)
28   p_u = discount*p
29   p_d = discount*(1-p)
30   # set up S values
31   SVals = matrix(0,2*N+1,1)
```

```

30   SVals[1] = S0*d^N
31   for (i in 2:(2*N+1)){
32     SVals[i] = u*SVals[i-1]
33   }
34   # set up terminal CALL values
35   CVals = matrix(0,2*N+1,1)
36   for (i in seq(1,2*N+1,2)){
37     CVals[i] = max(SVals[i]-K,0)
38   }
39   # work backwards
40   for (tau in 1:N){
41     for (i in seq((tau+1),(2*N+1-tau),2)){
42       CVals[i] = p_u*CVals[i+1] + p_d*CVals[i-1]
43     }
44   }
45   return(CVals[N+1])
46 }
47 tic()
48 BS_EC(S0 = 50, K = 50, r = 0.1, sigma = 0.4, T = 5/
      12)['price']
49 toc()
50
51 tic()
52 LatticeEurCall(S0 = 50, K = 50, r = 0.1, sigma =
      0.4, T = 5/12, N = 2000)
53 toc()
54
55 tic()
56 SmartEurLattice(S0 = 50, K = 50, r = 0.1, sigma =
      0.4, T = 5/12, N = 2000)
57 toc()

```

R code Exa 7.5 Pricing an American put by a binomial lattice

```

1 AmPutLattice <- function(S0,K,r,T,sigma,N) {

```

```

2  # Precompute invariant quantities
3  deltaT = T/N
4  u=exp(sigma * sqrt(deltaT))
5  d=1/u
6  p=(exp(r*deltaT) - d)/(u-d)
7  discount = exp(-r*deltaT)
8  p_u = discount*p
9  p_d = discount*(1-p)
10 # set up S values
11 SVals = matrix(0,2*N+1,1)
12 SVals[N+1] = S0
13 for (i in 1:N){
14     SVals[N+1+i] = u*SVals[N+1]
15     SVals[N+1-i] = d*SVals[N+2-i]
16 }
17 # set up terminal values
18 PVals = matrix(0,2*N+1,1)
19 for (i in seq(1,2*N+1,2)){
20     PVals[i] = max(K-SVals[i],0)
21 }
22 # work backwards
23 for (tau in 1:N){
24     for (i in seq((tau+1),(2*N+1-tau),2)){
25         hold = p_u*PVals[i+1] + p_d*PVals[i-1]
26         PVals[i] = max(hold, K-SVals[i])
27     }
28 }
29 return(PVals[N+1])
30 }
31
32 AmPutLattice(S0 = 50,K = 50,r = 0.05,T = 5/12, sigma
    = 0.4, N = 1000)

```

R code Exa 7.6 Pricing an American spread option by a bidimensional binomial lattice

```

1 AmSpreadLattice <- function(S10,S20,K,r,T,sigma1,
    sigma2,rho,q1,q2,N) {
2   # Precompute invariant quantities
3   deltaT = T/N
4   nu1 = r - q1 - 0.5*sigma1^2
5   nu2 = r - q2 - 0.5*sigma2^2
6   u1 = exp(sigma1*sqrt(deltaT))
7   d1 = 1/u1
8   u2 = exp(sigma2*sqrt(deltaT))
9   d2 = 1/u2
10  discount = exp(-r*deltaT)
11  p_uu = discount*0.25*(1 + sqrt(deltaT)*(nu1/sigma1
    + nu2/sigma2) + rho)
12  p_ud = discount*0.25*(1 + sqrt(deltaT)*(nu1/sigma1
    - nu2/sigma2) - rho)
13  p_du = discount*0.25*(1 + sqrt(deltaT)*(-nu1/
    sigma1 + nu2/sigma2) - rho)
14  p_dd = discount*0.25*(1 + sqrt(deltaT)*(-nu1/
    sigma1 - nu2/sigma2) + rho)
15  # set up S values
16  S1vals = matrix(0,2*N+1,1)
17  S2vals = matrix(0,2*N+1,1)
18  S1vals[1] = S10*d1^N
19  S2vals[1] = S20*d2^N
20  for (i in 2:(2*N+1)){
21    S1vals[i] = u1*S1vals[i-1]
22    S2vals[i] = u2*S2vals[i-1]
23  }
24  # set up terminal values
25  Cvals = matrix(0,2*N+1,2*N+1)
26  for (i in seq(1,2*N+1,2)){
27    for (j in seq(1,2*N+1,2)){
28      Cvals[i,j] = max(S1vals[i]-S2vals[j]-K,0)
29    }
30  }
31  # roll back
32  for (tau in 1:N){
33    for (i in seq((tau+1),(2*N+1-tau),2)){

```

```

34     for (j in seq((tau+1),(2*N+1-tau),2)){
35         hold = p_uu * Cvals[i+1,j+1] + p_ud * Cvals[
            i+1,j-1] + p_du * Cvals[i-1,j+1] + p_dd *
            Cvals[i-1,j-1]
36         Cvals[i,j] = max(hold, S1vals[i] - S2vals[j]
            - K)
37     }
38 }
39 }
40 return(Cvals[N+1,N+1])
41 }
42
43 AmSpreadLattice(S10 = 100, S20 = 100, K = 1, r =
    0.06, T = 1, sigma1 = 0.2, sigma2 = 0.3, rho =
    0.5, q1 = 0.03, q2 = 0.04, N = 3)

```

R code Exa 7.7 Pricing an European call by a trinomial lattice

```

1 require(OptionPricing)
2 EuCallTrinomial <- function(S0,K,r,T,sigma,N,deltaX)
3 {
4     # Precompute invariant quantities
5     deltaT = T/N
6     nu = r - 0.5*sigma^2
7     discount = exp(-r*deltaT)
8     p_u = discount*0.5*((sigma^2*deltaT+nu^2*deltaT^2)
9         /deltaX^2 + nu*deltaT/deltaX)
10    p_m = discount*(1 - (sigma^2*deltaT+nu^2*deltaT^2)
11        /deltaX^2)
12    p_d = discount*0.5*((sigma^2*deltaT+nu^2*deltaT^2)
13        /deltaX^2 - nu*deltaT/deltaX)
14    # set up S values (at maturity)
15    Svals = matrix(0,2*N+1,1)
16    Svals[1] = S0*exp(-N*deltaX)
17    exp_dX = exp(deltaX)

```

```

14     for (j in 2:(2*N+1)){
15         Svals[j] = exp_dX*Svals[j-1]
16     }
17     # set up lattice and terminal values
18     Cvals = matrix(0,2*N+1,2)
19     t = mod(N,2)+1
20     for (j in 1:(2*N+1)){
21         Cvals[j,t] = max(Svals[j]-K,0)
22     }
23     for (t in (N-1):0){
24         know = mod(t,2)+1
25         knext = mod(t+1,2)+1
26         for (j in (N-t+1):(N+t+1)){
27             Cvals[j,know] = p_d*Cvals[j-1,knext]+p_m*Cvals
                [j,knext]+p_u*Cvals[j+1,knext]
28         }
29     }
30     return(Cvals[N+1,1])
31 }
32 BS_EC(S0 = 100, K = 100, r = 0.06, sigma = 0.3, T =
    1)['price']
33 EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1,
    sigma = 0.3, N = 3, deltaX = 0.2)
34 EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1,
    sigma = 0.3, N = 100, deltaX = 0.2)
35 EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1,
    sigma = 0.3, N = 100, deltaX = 0.5)
36 EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1,
    sigma = 0.3, N = 100, deltaX = 0.3*sqrt(1/100))
37 EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1,
    sigma = 0.3, N = 1000, deltaX = 0.3*sqrt(1/1000)
    )

```

Chapter 8

Option Pricing by Monte Carlo Methods

R code Exa 8.1 Generate asset price paths by Monte Carlo simulation

```
1 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
2   SPaths = matrix(0,NRepl, 1+NSteps)
3   SPaths[,1] = S0
4   dt = T/NSteps
5   nudt = (mu-0.5*sigma^2)*dt
6   sidt = sigma*sqrt(dt)
7   for (i in 1:NRepl){
8     for (j in 1:NSteps){
9       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
10         rnorm(1))
11     }
12   }
13   return(SPaths)
14 }
15 set.seed(37456)
16 paths=AssetPaths(50,0.1,0.3,1,365,3)
17 plot(1:length(paths[3,]),paths[3,],type = 'l')
18 lines(1:length(paths[1,]),paths[1,],type = 'l')
19 lines(1:length(paths[2,]),paths[2,],type = 'l')
```

R code Exa 8.2 Vectorized code to generate asset price paths

```
1 require(varbvs)
2 require(tictoc)
3 AssetPathsV <- function(S0,mu,sigma,T,NSteps,NRepl)
4 {
5   dt = T/NSteps
6   nudt = (mu-0.5*sigma^2)*dt
7   sidt = sigma*sqrt(dt)
8   Increments = nudt + sidt*randn(NRepl, NSteps)
9   LogPaths = apply(cbind(matrix(log(S0)*matrix(1,
10     NRepl,1)),Increments),2,cumsum)
11   SPaths = exp(LogPaths)
12   SPaths[,1] = S0
13   return(SPaths)
14 }
15 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
16   SPaths = matrix(0,NRepl, 1+NSteps)
17   SPaths[,1] = S0
18   dt = T/NSteps
19   nudt = (mu-0.5*sigma^2)*dt
20   sidt = sigma*sqrt(dt)
21   for (i in 1:NRepl){
22     for (j in 1:NSteps){
23       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
24         rnorm(1))
25     }
26   }
27   return(SPaths)
28 }
29 Paths = AssetPathsV(50,0.1,0.3,1,100,1000)
30
31 N = dim(Paths)[2]
32 for (i in 1:N){
```

```

30   plot(Paths[,i],type = 'l')
31 }
32
33 tic()
34 AssetPaths(50,0.1,0.3,1,100,1000)
35 toc()
36
37 tic()
38 AssetPathsV(50,0.1,0.3,1,100,1000)
39 toc()

```

R code Exa 8.3 Evaluating the cost of a stop loss hedging strategy

```

1  require(pracma)
2  require(tictoc)
3  StopLoss <- function(S0,K,mu,sigma,r,T,Paths) {
4    NRepl = dim(Paths)[1]
5    NSteps = dim(Paths)[2]
6    NSteps = NSteps - 1
7    # true number of steps
8    Cost = matrix(0,NRepl,1)
9    dt = T/NSteps
10   DiscountFactors = exp(-r*(seq(0,NSteps,1)*dt))
11   for (k in 1:NRepl){
12     CashFlows = matrix(0,NSteps+1,1)
13     if (Paths[k,1] >= K){
14       Covered = 1
15       CashFlows[1] = -Paths[k,1]
16     } else {
17       Covered = 0
18     }
19     for (t in 2:(NSteps+1)){
20       if ((Covered == 1) & (Paths[k,t] < K)){
21         # Sell
22         Covered = 0

```

```

23     CashFlows[t] = Paths[k,t]
24   } else if ((Covered == 0) & (Paths[k,t] > K)){
25     # Buy
26     Covered = 1
27     CashFlows[t] = -Paths[k,t]
28   }
29 }
30 if (Paths[k,NSteps + 1] >= K){
31   # Option is exercised
32   CashFlows[NSteps + 1] = CashFlows[NSteps + 1]
      + K
33 }
34 Cost[k] = - dot(DiscountFactors,CashFlows)
35 }
36 return(mean(Cost))
37 }
38 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
39   SPaths = matrix(0,NRepl, 1+NSteps)
40   SPaths[,1] = S0
41   dt = T/NSteps
42   nudt = (mu-0.5*sigma^2)*dt
43   sidt = sigma*sqrt(dt)
44   for (i in 1:NRepl){
45     for (j in 1:NSteps){
46       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
      rnorm(1))
47     }
48   }
49   return(SPaths)
50 }
51 S0 = 50
52 K = 50
53 mu = 0.1
54 sigma = 0.4
55 r = 0.05
56 T = 5/12
57 NRepl =100000
58 NSteps = 10

```

```

59 set.seed(39473)
60 Paths=AssetPaths(S0,mu, sigma,T,NSteps,NRepl)
61 tic()
62 StopLoss(S0,K,mu,sigma,r,T,Paths)
63 toc()

```

R code Exa 8.4 Vectorized code for the stop loss hedging strategy

```

1 require(pracma)
2 require(tictoc)
3 StopLoss <- function(S0,K,mu,sigma,r,T,Paths) {
4   NRepl = dim(Paths)[1]
5   NSteps = dim(Paths)[2]
6   NSteps = NSteps - 1
7   # true number of steps
8   Cost = matrix(0,NRepl,1)
9   dt = T/NSteps
10  DiscountFactors = exp(-r*(seq(0,NSteps,1)*dt))
11  for (k in 1:NRepl){
12    CashFlows = matrix(0,NSteps+1,1)
13    if (Paths[k,1] >= K){
14      Covered = 1
15      CashFlows[1] = -Paths[k,1]
16    } else {
17      Covered = 0
18    }
19    for (t in 2:(NSteps+1)){
20      if ((Covered == 1) & (Paths[k,t] < K)){
21        # Sell
22        Covered = 0
23        CashFlows[t] = Paths[k,t]
24      } else if ((Covered == 0) & (Paths[k,t] > K)){
25        # Buy
26        Covered = 1
27        CashFlows[t] = -Paths[k,t]

```

```

28     }
29 }
30 if (Paths[k,NSteps + 1] >= K){
31     # Option is exercised
32     CashFlows[NSteps + 1] = CashFlows[NSteps + 1]
        + K
33 }
34 Cost[k] = - dot(DiscountFactors,CashFlows)
35 }
36 return(mean(Cost))
37 }
38 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
39     SPaths = matrix(0,NRepl, 1+NSteps)
40     SPaths[,1] = S0
41     dt = T/NSteps
42     nudt = (mu-0.5*sigma^2)*dt
43     sidt = sigma*sqrt(dt)
44     for (i in 1:NRepl){
45         for (j in 1:NSteps){
46             SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
                rnorm(1))
47         }
48     }
49     return(SPaths)
50 }
51 S0 = 50
52 K = 50
53 mu = 0.1
54 sigma = 0.4
55 r = 0.05
56 T = 5/12
57 NRepl =100000
58 NSteps = 10
59 set.seed(39473)
60 Paths=AssetPaths(S0,mu, sigma,T,NSteps,NRepl)
61
62 StopLossV <- function(S0,K,mu,sigma,r,T,Paths) {
63     NRepl = dim(Paths)[1]

```

```

64   NSteps = dim(Paths)[2]
65   NSteps = NSteps - 1
66   Cost = matrix(0,NRepl,1)
67   CashFlows = matrix(0,NRepl,NSteps+1)
68   dt = T/NSteps
69   DiscountFactors = exp(-r*(seq(0,NSteps,1))*dt)
70   OldPrice = cbind(matrix(0,NRepl,1), Paths[,1:
      NSteps])
71   UpTimes = which(OldPrice < K & Paths >= K)
72   DownTimes = which(OldPrice >= K & Paths < K)
73   CashFlows[UpTimes] = -Paths[UpTimes]
74   CashFlows[DownTimes] = Paths[DownTimes]
75   ExPaths = which(Paths[,NSteps+1] >= K)
76   CashFlows[ExPaths,NSteps+1] = CashFlows[ExPaths,
      NSteps+1] + K
77   Cost = -CashFlows %*%(matrix(DiscountFactors))
78   return(mean(Cost))
79 }
80
81 tic()
82 StopLossV(S0,K,mu,sigma,r,T,Paths)
83 toc()

```

R code Exa 8.5 Evaluating the performance of delta hedging

```

1 f<-function(T, Path){
2   BS_EC(S0 = Path,K = K,r = r,sigma = sigma,T = T)[',
      delta']
3 }
4 require(OptionPricing)
5 DeltaHedging <- function(S0,K,mu,sigma,r,T,Paths) {
6   NRepl = dim(Paths)[1]
7   NSteps = dim(Paths)[2]
8   NSteps = NSteps - 1
9   Cost = matrix(0,NRepl,1)

```

```

10  CashFlows = matrix(0,1,NSteps+1)
11  dt = T/NSteps
12  DiscountFactors = exp(-r*(seq(0,NSteps,1)*dt))
13  for (i in 1:NRepl){
14    Path = Paths[i,]
15    Position = 0
16    Deltas = matrix()
17    for (k in 1:NSteps){
18      Deltas[k] = f(T = T-(k-1)*dt, Path = Path[k])
19    }
20    for (j in 1:NSteps){
21      CashFlows[j] = (Position - Deltas[j])*Path[j]
22      Position = Deltas[j]
23    }
24    if (Path[NSteps+1] > K){
25      CashFlows[NSteps+1] = K - (1-Position)*Path[
        NSteps+1]
26    } else {
27      CashFlows[NSteps+1] = Position*Path[NSteps+1]
28    }
29    Cost[i] = -CashFlows %*% DiscountFactors
30  }
31  return(mean(Cost))
32 }
33
34 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
35   SPaths = matrix(0,NRepl, 1+NSteps)
36   SPaths[,1] = S0
37   dt = T/NSteps
38   nudt = (mu-0.5*sigma^2)*dt
39   sidt = sigma*sqrt(dt)
40   for (i in 1:NRepl){
41     for (j in 1:NSteps){
42       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
        rnorm(1))
43     }
44   }
45   return(SPaths)

```

```

46 }
47
48 StopLossV <- function(S0,K,mu,sigma,r,T,Paths) {
49   NRepl = dim(Paths)[1]
50   NSteps = dim(Paths)[2]
51   NSteps = NSteps - 1
52   Cost = matrix(0,NRepl,1)
53   CashFlows = matrix(0,NRepl,NSteps+1)
54   dt = T/NSteps
55   DiscountFactors = exp(-r*(seq(0,NSteps,1))*dt)
56   OldPrice = cbind(matrix(0,NRepl,1), Paths[,1:
      NSteps])
57   UpTimes = which(OldPrice < K & Paths >= K)
58   DownTimes = which(OldPrice >= K & Paths < K)
59   CashFlows[UpTimes] = -Paths[UpTimes]
60   CashFlows[DownTimes] = Paths[DownTimes]
61   ExPaths = which(Paths[,NSteps+1] >= K)
62   CashFlows[ExPaths,NSteps+1] = CashFlows[ExPaths,
      NSteps+1] + K
63   Cost = -CashFlows %*%(matrix(DiscountFactors))
64   return(mean(Cost))
65 }
66
67 S0 = 50
68 K = 52
69 mu = 0.1
70 sigma = 0.4
71 r = 0.05
72 T = 5/12
73 NRepl = 10000
74 NSteps = 10
75 C = BS_EC(K = K, r = r, sigma = sigma, T = T, S0 =
      S0)['price']
76 set.seed(3872)
77 Paths=AssetPaths(S0,mu,sigma,T,NSteps,NRepl)
78 True_Price = BS_EC(S0 = S0,K = K,r = r,sigma = sigma
      ,T = T)['price']
79 SL = StopLossV(S0,K,mu,sigma,r,T,Paths)

```



```

80 DC = DeltaHedging(S0,K,mu,sigma,r,T,Paths)
81
82 #NSteps = 100
83 #set.seed(38232)
84 #Paths=AssetPaths(S0,mu,sigma,T,NSteps,NRepl)
85 #True_Price = BS_EC(S0 = S0,K = K,r = r,sigma =
      sigma,T = T)['price']
86 #SL = StopLossV(S0,K,mu,sigma,r,T,Paths)
87 #DC = DeltaHedging(S0,K,mu,sigma,r,T,Paths)

```

R code Exa 8.6 Implementing and checking path generation for the standard Wiener process by a Brownian bridge

```

1 WienerBridge <- function(T, NSteps) {
2   NBisections = log2(NSteps)
3   if (round(NBisections) != NBisections){
4     cat('ERROR in WienerBridge: NSteps must be a
        power of 2', '\n')
5     return
6   }
7   WSamples = matrix(0,NSteps+1,1)
8   WSamples[1] = 0
9   WSamples[NSteps+1] = sqrt(T)*rnorm(1)
10  TJump = T
11  IJump = NSteps
12  for (k in 1:NBisections){
13    left = 1
14    i = IJump/2 + 1
15    right = IJump + 1
16    for (j in 1:(2^(k-1))){
17      a = 0.5*(WSamples[left] + WSamples[right])
18      b = 0.5*sqrt(TJump)
19      WSamples[i] = a + b*rnorm(1)
20      right = right + IJump
21      left = left + IJump

```

```

22         i = i + IJump
23     }
24     IJump = IJump/2
25     TJump = TJump/2
26 }
27 return(WSamples)
28 }
29
30
31 # CheckBridge.m
32 set.seed(3826)
33 NRepl = 100000
34 T = 1
35 NSteps = 4
36 WSamples = matrix(0,NRepl, 1+NSteps)
37 for (i in 1:NRepl){
38     WSamples[i,] =WienerBridge(T, NSteps)
39 }
40 a <- function(X) {
41     mean(X)
42 }
43 b <- function(X) {
44     sqrt(var(X))
45 }
46 m = matrix()
47 sdev = matrix()
48 for (z in 2:(1+NSteps)){
49     m[z-1] = a(WSamples[,z])
50     sdev[z-1] = b(WSamples[,z])
51 }
52 m
53 sdev
54 sqrt((1:NSteps)*T/NSteps)

```

R code Exa 8.7 Code to price an exchange option analytically

```

1 require(lmom)
2 Exchange <- function(V0,U0,sigmaV,sigmaU,rho,T,r) {
3   sigmahat = sqrt(sigmaU^2 + sigmaV^2 - 2*rho*sigmaU
4     *sigmaV)
5   d1 = (log(V0/U0) + 0.5*T*sigmahat^2)/(sigmahat*
6     sqrt(T))
7   d2 = d1 - sigmahat*sqrt(T)
8   p = V0*cdfnor(d1) - U0*cdfnor(d2)
9   return(p)
10 }
11
12 V0 = 50
13 U0 = 60
14 sigmaV = 0.3
15 sigmaU = 0.4
16 rho = 0.7
17 T = 5/12
18 r = 0.05
19 Exchange(V0 ,U0, sigmaV, sigmaU, rho, T ,r)

```

R code Exa 8.8 Code to price an exchange option by Monte Carlo simulation

```

1 require(fBasics)
2 norm.interval = function(data, variance = var(data),
3   conf.level = 0.95) {
4   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
5   xbar = mean(data)
6   sdx = sqrt(variance/length(data))
7   c(xbar - z * sdx, xbar + z * sdx)
8 }
9
10 f <- function(r,T,VT,UT) {
11   exp(-r*T)*max(VT-UT, 0)
12 }
13
14 ExchangeMC <- function(V0,U0,sigmaV,sigmaU,rho,T,r,

```

```

      NRepl) {
12  eps1 = rnorm(NRepl)
13  eps2 = rho*eps1 + sqrt(1-rho^2)*rnorm(NRepl)
14  VT = V0*exp((r - 0.5*sigmaV^2)*T + sigmaV*sqrt(T)*
      eps1)
15  UT = U0*exp((r - 0.5*sigmaU^2)*T + sigmaU*sqrt(T)*
      eps2)
16  DiscPayoff = matrix()
17  for(i in 1:length(VT)){
18    DiscPayoff[i] = f(r,T,VT[i],UT[i])
19  }
20  parameter_estimation<-normFit(DiscPayoff)
21  ci<-norm.interval(DiscPayoff)
22  return(c(parameter_estimation,ci))
23 }
24 V0 = 50
25 U0 = 60
26 sigmaV = 0.3
27 sigmaU = 0.4
28 rho = 0.7
29 T = 5/12
30 r = 0.05
31 NRepl = 200000
32 ExchangeMC(V0,U0,sigmaV,sigmaU,rho,T,r,NRepl)

```

R code Exa 8.9 Crude Monte Carlo simulation for a discrete barrier option

```

1 require(OptionPricing)
2 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
3   SPaths = matrix(0,NRepl, 1+NSteps)
4   SPaths[,1] = S0
5   dt = T/NSteps
6   nudt = (mu-0.5*sigma^2)*dt
7   sidt = sigma*sqrt(dt)
8   for (i in 1:NRepl){

```

```

9      for (j in 1:NSteps){
10          SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
              rnorm(1))
11      }
12  }
13  return(SPaths)
14 }
15 norm.interval = function(data, variance = var(data),
      conf.level = 0.95) {
16  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
17  xbar = mean(data)
18  sdx = sqrt(variance/length(data))
19  c(xbar - z * sdx, xbar + z * sdx)
20 }
21 DOPutMC<-function(S0,K,r,T,sigma,Sb,NSteps,NRepl){
22   # Generate asset paths
23   Call = BS_EC(S0,K,r,T,sigma)
24   Put = BS_EP(S0,K,r,T,sigma)
25   Payoff = matrix(0,NRepl,1)
26   NCrossed = 0
27   for (i in 1:NRepl){
28     Path=AssetPaths(S0,r,sigma,T,NSteps,1)
29     crossed = any(Path <= Sb)
30     if (crossed == 0){
31       Payoff[i] = max(0, K - Path[NSteps+1])
32     } else{
33       Payoff[i] = 0
34       NCrossed = NCrossed + 1
35     }
36   }
37   parameter_estimation<-.normFit(exp(-r*T) * Payoff)
38   ci<-norm.interval(exp(-r*T) * Payoff)
39   return(c(parameter_estimation,ci))
40 }
41 DOPutMC(50,50,0.1,2/12,0.4,40,60,50000)

```

R code Exa 8.10 Conditional Monte Carlo simulation for a discrete barrier option

```
1 require(OptionPricing)
2 require(fBasics)
3 require(fOptions)
4 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
5   SPaths = matrix(0,NRepl, 1+NSteps)
6   SPaths[,1] = S0
7   dt = T/NSteps
8   nudt = (mu-0.5*sigma^2)*dt
9   sidt = sigma*sqrt(dt)
10  for (i in 1:NRepl){
11    for (j in 1:NSteps){
12      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
13        rnorm(1))
14    }
15  }
16  return(SPaths)
17 }
18 norm.interval = function(data, variance = var(data),
19   conf.level = 0.95) {
20   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
21   xbar = mean(data)
22   sdx = sqrt(variance/length(data))
23   c(xbar - z * sdx, xbar + z * sdx)
24 }
25 is.integer0 <- function(x)
26 {
27   is.integer(x) && length(x) == 0L
28 }
29
```

```

30 DOPutMCCond <- function(S0,K,r,T,sigma,Sb,NSteps,
    NRepl) {
31   dt = T/NSteps;
32   Call = GBSOption(TypeFlag = "c", S = S0, X = K,
    Time = T, r = r, sigma = sigma, b =r)
33   Put = GBSOption(TypeFlag = "p", S = S0, X = K,
    Time = T, r = r, sigma = sigma, b =r)
34   # Generate asset paths and payoffs for the down
    and in option
35   NCrossed = 0
36   Payoff = matrix(0,NRepl,1)
37   Times = matrix(0,NRepl,1)
38   StockVals = matrix(0,NRepl,1)
39   for (i in 1:NRepl){
40     Path=AssetPaths(S0,r,sigma,T,NSteps,1)
41     tcrossed = pmin(which(Path <= Sb))
42     if (!(is.integer0(tcrossed))){
43       NCrossed = NCrossed + 1
44       Times[NCrossed,] = (length(tcrossed) - 1) * dt
45       StockVals[NCrossed,] = Path[,length(tcrossed)]
46     }
47   }
48   Paux<-matrix()
49   f <- function(S0,K,r,T,sigma) {
50     GBSOption(TypeFlag = "p", S = S0, X = K, Time =
    T, r = r, sigma = sigma, b =r)
51   }
52   if (NCrossed > 0){
53     for (j in 1:NCrossed){
54       Paux[j] = f(StockVals[j],K,r,(T-Times[j]),
    sigma)@price
55       Payoff[j] = exp(-r*Times[j]) * Paux[j]
56     }
57   }
58   parameter_estimation<-.normFit(Put@price - Payoff)
59   ci<-norm.interval(Put@price - Payoff)
60   return(c(parameter_estimation,ci,NCrossed))
61 }

```

```

62
63 DOPutMCCond(50,52,0.1,2/12,0.4,30,60,200000)

```

R code Exa 8.11 Using conditional Monte Carlo and importance sampling for a discrete barrier option

```

1  require(OptionPricing)
2  require(fBasics)
3  require(fOptions)
4  require(varbvs)
5  AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
6    SPaths = matrix(0,NRepl, 1+NSteps)
7    SPaths[,1] = S0
8    dt = T/NSteps
9    nudt = (mu-0.5*sigma^2)*dt
10   sidt = sigma*sqrt(dt)
11   for (i in 1:NRepl){
12     for (j in 1:NSteps){
13       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
14         rnorm(1))
15     }
16   }
17   return(SPaths)
18 }
19 norm.interval = function(data, variance = var(data),
20   conf.level = 0.95) {
21   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
22   xbar = mean(data)
23   sdx = sqrt(variance/length(data))
24   c(xbar - z * sdx, xbar + z * sdx)
25 }
26 is.integer0 <- function(x)
27 {

```



```

28   is.integer(x) && length(x) == 0L
29 }
30
31 DOPutMCCondIS <- function(S0,K,r,T,sigma,Sb,NSteps,
   NRepl,bp){
32   dt = T/NSteps
33   nudt = (r-0.5*sigma^2)*dt
34   b = bp*nudt
35   sidt = sigma*sqrt(dt)
36   Call = GBSOption(TypeFlag = "c", S = S0, X = K,
   Time = T, r = r, sigma = sigma, b =r)
37   Put = GBSOption(TypeFlag = "p", S = S0, X = K,
   Time = T, r = r, sigma = sigma, b =r)
38   # Generate asset paths and payoffs for the down
   and in option
39   NCrossed = 0
40   Payoff = matrix(0,NRepl,1)
41   Times = matrix(0,NRepl,1)
42   StockVals = matrix(0,NRepl,1)
43   ISRatio = matrix(0,NRepl,1)
44   for (i in 1:NRepl){
45     # generate normals
46     vetZ = nudt - b + sidt*randn(1,NSteps)
47     LogPath = apply(cbind(log(S0), vetZ),1,cumsum)
48     Path = exp(LogPath)
49     jcrossed = pmin(which(Path <= Sb ))
50     if (!(is.integer0(jcrossed))){
51       jcrossed = min(jcrossed)
52       NCrossed = NCrossed + 1
53       TBreach = jcrossed - 1
54       Times[NCrossed,] = TBreach * dt
55       StockVals[NCrossed,] = Path[jcrossed]
56       ISRatio[NCrossed,] = exp( TBreach*b^2/2/sigma
   ^2/dt + b/sigma^2/dt*sum(vetZ[1:TBreach]) -
   TBreach*b/sigma^2*(r - sigma^2/2))
57     }
58   }
59   Paux<-matrix()

```

```

60  f <- function(S0,K,r,T,sigma) {
61    GBSOption(TypeFlag = "p", S = S0, X = K, Time =
        T, r = r, sigma = sigma, b =r)
62  }
63  if (NCrossed > 0){
64    for (j in 1:NCrossed){
65      Paux[j] = f(StockVals[j],K,r,(T-Times[j]),
        sigma)@price
66      Payoff[j] = exp(-r*Times[j])* Paux[j] *
        ISRatio[j]
67    }
68  }
69  parameter_estimation<-.normFit(Put@price - Payoff)
70  ci<-norm.interval(Put@price - Payoff)
71  return(c(parameter_estimation,ci,NCrossed))
72 }
73 DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,0)
74 DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,20)
75 DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,50)
76 DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,200)

```

R code Exa 8.12 Monte Carlo simulation for an Asian option

```

1  require(OptionPricing)
2  require(fBasics)
3  require(fOptions)
4  require(varbvs)
5  AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
6    SPaths = matrix(0,NRepl, 1+NSteps)
7    SPaths[,1] = S0
8    dt = T/NSteps
9    nudt = (mu-0.5*sigma^2)*dt
10   sidt = sigma*sqrt(dt)
11   for (i in 1:NRepl){
12     for (j in 1:NSteps){

```

```

13       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
           rnorm(1))
14   }
15 }
16 return(SPaths)
17 }
18 norm.interval = function(data, variance = var(data),
           conf.level = 0.95) {
19   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
20   xbar = mean(data)
21   sdx = sqrt(variance/length(data))
22   c(xbar - z * sdx, xbar + z * sdx)
23 }
24 AsianMC<-function(S0,K,r,T,sigma,NSamples,NRepl){
25   Payoff = matrix(0,NRepl,1)
26   for (i in 1:NRepl){
27     Path=AssetPaths(S0,r,sigma,T,NSamples,1)
28     Payoff[i] = max(0, mean(Path[2:(NSamples+1)]) -
           K)
29   }
30   parameter_estimation<-.normFit(exp(-r*T) * Payoff)
31   ci<-norm.interval(exp(-r*T) * Payoff)
32   return(c(parameter_estimation,ci))
33 }
34 set.seed(28282)
35 X<-AsianMC(50,50,0.1,5/12,0.4,5,50000)
36 X
37 X[[3]]-X[[2]]

```

R code Exa 8.13 Monte Carlo simulation with control variates for an Asian option

```

1 require(OptionPricing)
2 require(fBasics)
3 require(fOptions)

```

```

4 require(varbvs)
5 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
6   SPaths = matrix(0,NRepl, 1+NSteps)
7   SPaths[,1] = S0
8   dt = T/NSteps
9   nudt = (mu-0.5*sigma^2)*dt
10  sidt = sigma*sqrt(dt)
11  for (i in 1:NRepl){
12    for (j in 1:NSteps){
13      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
14        rnorm(1))
15    }
16  }
17  return(SPaths)
18 }
19 norm.interval = function(data, variance = var(data),
20   conf.level = 0.95) {
21   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
22   xbar = mean(data)
23   sdx = sqrt(variance/length(data))
24   c(xbar - z * sdx, xbar + z * sdx)
25 }
26 AsianMCCV<-function(S0,K,r,T,sigma,NSamples,NRepl,
27   NPilot){
28   # pilot replications to set control parameter
29   TryPath=AssetPaths(S0,r,sigma,T,NSamples,NPilot)
30   StockSum<-matrix()
31   PP<-matrix()
32   TryPayoff<-matrix()
33   for (i in 1:length(TryPath[,1])){
34     StockSum[i] = sum(TryPath[i,])
35     PP[i] = mean(TryPath[i,2:(NSamples+1)])
36     TryPayoff[i] = exp(-r*T) * max(0, PP[i] - K)
37   }
38   MatCov = cov(cbind(StockSum, TryPayoff))
39   c = - MatCov[1,2] / var(StockSum)
40   dt = T / NSamples
41   ExpSum = S0 * (1 - exp((NSamples + 1)*r*dt)) / (1

```

```

      - exp(r*dt))
39 # MC run
40 ControlVars = matrix(0,NRepl,1)
41 for (i in 1:NRepl){
42   StockPath = AssetPaths(S0,r,sigma,T,NSamples,1)
43   Payoff = exp(-r*T) * max(0, mean(StockPath[2:(
      NSamples+1)])) - K)
44   ControlVars[i] = Payoff + c * (sum(StockPath) -
      ExpSum)
45 }
46 parameter_estimation<-.normFit(ControlVars)
47 ci<-norm.interval(ControlVars)
48 return(c(parameter_estimation,ci))
49 }
50 X<-AsianMCCV(50,50 ,0.1,5/12,0.4,5,45000,5000)
51 X
52 X[[3]]-X[[2]]

```

R code Exa 8.14 Using the geometric average Asian option as a control variate

```

1 require(OptionPricing)
2 require(fBasics)
3 require(fOptions)
4 require(varbvs)
5 require(lmom)
6 GeometricAsian<-function(S0,K,r,T,sigma,delta,
  NSamples){
7   dT = T/NSamples
8   nu = r - sigma^2/2-delta
9   a = log(S0)+nu*dT+0.5*nu*(T-dT)
10  b = sigma^2*dT + sigma^2*(T-dT)*(2*NSamples-1)/6/
    NSamples
11  x = (a-log(K)+b)/sqrt(b)
12  P = exp(-r*T)*(exp(a+b/2)*cdfnor(x) - K*cdfnor(x-

```

```

        sqrt(b)))
13   return(P)
14 }
15 norm.interval = function(data, variance = var(data),
        conf.level = 0.95) {
16   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
17   xbar = mean(data)
18   sdx = sqrt(variance/length(data))
19   c(xbar - z * sdx, xbar + z * sdx)
20 }
21 AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
22   SPaths = matrix(0,NRepl, 1+NSteps)
23   SPaths[,1] = S0
24   dt = T/NSteps
25   nudt = (mu-0.5*sigma^2)*dt
26   sidt = sigma*sqrt(dt)
27   for (i in 1:NRepl){
28     for (j in 1:NSteps){
29       SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
        rnorm(1))
30     }
31   }
32   return(SPaths)
33 }
34 AsianMCGeoCV<-function(S0,K,r,T,sigma,NSamples,NRepl
        ,NPilot){
35   # precompute quantities
36   DF = exp(-r*T)
37   GeoExact = GeometricAsian(S0,K,r,T,sigma,0,
        NSamples)
38   # pilot replications to set control parameter
39   GeoPrices = matrix(0,NPilot,1)
40   AriPrices = matrix(0,NPilot,1)
41   for (i in 1:NPilot){
42     Path=AssetPaths(S0,r,sigma,T,NSamples,1)
43     GeoPrices[i]=DF*max(0,(prod(Path[,2:(NSamples+1)
        ]))^(1/NSamples) - K)
44     AriPrices[i]=DF*max(0,mean(Path[,2:(NSamples+1)

```

```

    ]) - K)
45 }
46 MatCov = cov(cbind(GeoPrices, AriPrices))
47 c = - MatCov[1,2] / var(GeoPrices)
48 # MC run
49 ControlVars = matrix(0,NRepl,1)
50 for (i in 1:NRepl){
51   Path = AssetPaths(S0,r,sigma,T,NSamples,1)
52   GeoPrice = DF*max(0, (prod(Path[2:(NSamples+1)]))
53     ^ (1/NSamples) - K)
54   AriPrice = DF*max(0, mean(Path[2:(NSamples+1)]) -
55     K)
56   ControlVars[i] = AriPrice + c * (GeoPrice -
57     GeoExact)
58 }
59 parameter_estimation<-.normFit(ControlVars)
60 ci<-norm.interval(ControlVars)
61 return(c(parameter_estimation,ci))
62 }
63 set.seed(2372)
64 S0 = 50
65 K = 55
66 r = 0.05
67 sigma = 0.4
68 T = 1
69 NSamples = 12
70 NRepl = 9000
71 NPilot = 1000
72 AsianMCGeoCV(S0,K,r,T,sigma,NSamples,NRepl,NPilot)

```

R code Exa 8.15 Pricing an Asian option by Halton sequences

```

1 require(gmp)
2 require(lmom)
3 require(leststat)

```

```

4
5 GetHalton <- function(HowMany, Base) {
6   Seq = matrix(0,HowMany,1)
7   NumBits = 1+round(log(HowMany)/log(Base));
8   VetBase = Base^(-(1:NumBits));
9   WorkVet = matrix(0,1,NumBits);
10  for (i in 1:HowMany){
11    j = 1;
12    ok = 0;
13    while (ok == 0){
14      WorkVet[j] = WorkVet[j]+1;
15      if (WorkVet[j] < Base){
16        ok = 1;
17      }
18      else{
19        WorkVet[j] = 0;
20        j = j+1;
21      }
22    }
23    Seq[i] = sum(WorkVet * VetBase)
24  }
25  return(Seq)
26 }
27
28 myprimes<-function(N){
29   found = 0
30   trynumber = 2
31   p <- matrix()
32   while (found < N){
33     if (isprime(trynumber)){
34       p <-c(p , trynumber)
35       found = found + 1
36     }
37     trynumber = trynumber + 1
38   }
39   return(p)
40 }
41

```



```

42 HaltonPaths<-function(S0,mu,sigma,T,NSteps,NRepl){
43   dt = T/NSteps
44   nudt = (mu-0.5*sigma^2)*dt
45   sidt = sigma*sqrt(dt)
46   # Use inverse transform to generate standard
      normals
47   NormMat = matrix(0,NRepl, NSteps)
48   Bases = myprimes(NSteps)
49   RandMat<-matrix(0,NRepl,NSteps)
50   H <- matrix()
51   for (i in 2:(NSteps+1)){
52     H = GetHalton(NRepl,Bases[i])
53     for (j in 1:length(H)){
54       RandMat[j,i-1] = invcdf(normal(),H[j])
55     }
56   }
57   Increments = nudt + sidt*RandMat
58   LogPaths = apply(cbind(log(S0)*matrix(1,NRepl,1),
      Increments),1,cumsum)
59   LogPaths = t(LogPaths)
60   SPaths = exp(LogPaths)
61   SPaths[,1] = S0
62   return(SPaths)
63 }
64
65 AsianHalton<-function(S0,K,r,T,sigma,NSamples,NRepl)
      {
66   Payoff = matrix(0,NRepl,1)
67   Path=HaltonPaths(S0,r,sigma,T,NSamples,NRepl)
68   Payoff<-matrix(0,NSamples,1)
69   for(k in 1:length(Path[,1])){
70     Payoff[k] = max(0, mean(Path[k,2:(NSamples+1)])
      - K)
71   }
72   P = mean( exp(-r*T) * matrix(Payoff))
73   return(P)
74 }
75

```

```

76 set.seed(3226)
77 AsianHalton(50,50,0.1,5/12,0.4,5,1000)
78 AsianHalton(50,50,0.1,5/12,0.4,5,3000)
79 AsianHalton(50,50,0.1,5/12,0.4,5,10000)
80 AsianHalton(50,50,0.1,5/12,0.4,5,50000)
81
82 AsianHalton(50,50,0.1,2,0.4,24,1000)
83 AsianHalton(50,50,0.1,2,0.4,24,5000)
84 AsianHalton(50,50,0.1,2,0.4,24,50000)

```

R code Exa 8.16 Simulating geometric Brownian motion by Halton sequences and the Brownian bridge

```

1  require(gmp)
2  require(lmom)
3  require(lestat)
4  require(matrixStats)
5
6  myprimes<-function(N){
7    found = 0
8    trynumber = 2
9    p <- matrix()
10   while (found < N){
11     if (isprime(trynumber)){
12       p <-c(p , trynumber)
13       found = found + 1
14     }
15     trynumber = trynumber + 1
16   }
17   return(p)
18 }
19
20 GetHalton <- function(HowMany, Base) {
21   Seq = matrix(0,HowMany,1)
22   NumBits = 1+round(log(HowMany)/log(Base));

```

```

23 VetBase = Base^(-(1:NumBits));
24 WorkVet = matrix(0,1,NumBits);
25 for (i in 1:HowMany){
26     j = 1;
27     ok = 0;
28     while (ok == 0){
29         WorkVet[j] = WorkVet[j]+1;
30         if (WorkVet[j] < Base){
31             ok = 1;
32         }
33         else{
34             WorkVet[j] = 0;
35             j = j+1;
36         }
37     }
38     Seq[i] = sum(WorkVet * VetBase)
39 }
40 return(Seq)
41 }
42
43 WienerHaltonBridge<-function(T, NSteps, NRepl, Limit
44 ){
45     NBisections = log2(NSteps)
46     if (round(NBisections) != NBisections){
47         cat('ERROR in WienerHB: NSteps must be a power
48             of 2 ', '\n')
49     }
50     return
51 }
52 # Generate standard normal samples
53 NormMat = matrix(0,NRepl, NSteps)
54 Bases = myprimes(NSteps)
55 for (i in 2:(NSteps+1)){
56     H = GetHalton(NRepl,Bases[i])
57     for (j in 1:length(H)){
58         NormMat[j,i-1] = invcdf(normal(),H[j])
59     }
60 }
61 # Initialize extreme points of paths

```

```

59   WSamples = matrix(0,NRepl,NSteps+1)
60   WSamples[,1] = 0
61   WSamples[,NSteps+1] = sqrt(T)*NormMat[,1]
62   # Fill paths
63   HUse = 2
64   TJump = T
65   IJump = NSteps
66   for (k in 1:NBisections){
67     left = 1
68     i = IJump/2 + 1
69     right = IJump + 1
70     for (l in 1:(2^(k-1))){
71       a = 0.5*(WSamples[,left] + WSamples[,right])
72       b = 0.5*sqrt(TJump)
73       if (HUse <= Limit){
74         WSamples[,i] = a + b*NormMat[,HUse]
75       } else {
76         WSamples[,i] = a + b*rnorm(NRepl)
77       }
78       right = right + IJump
79       left = left + IJump
80       i = i + IJump
81     }
82     IJump = IJump/2
83     TJump = TJump/2
84     HUse = HUse + 1
85   }
86   return(WSamples)
87 }
88
89 GBMHaltonBridge<-function(S0,mu,sigma,T,NSteps,NRepl
,Limit){
90   if (round(log2(NSteps)) != log2(NSteps)){
91     cat('ERROR in GBMBridge: NSteps must be a power
of 2', '\n')
92     return
93   }
94   dt = T/NSteps

```

```

95     nudt = (mu-0.5*sigma^2)*dt
96     W = WienerHaltonBridge(T,NSteps,NRepl,Limit)
97     Increments = nudt + sigma*t(diff(t(W)))
98     LogPaths = apply(cbind(log(S0)*matrix(1,NRepl,1),
        Increments),1,cumsum)
99     LogPaths = t(LogPaths)
100    Paths = exp(LogPaths)
101    Paths[,1] = S0
102    return(Paths)
103 }
104
105 set.seed(271782)
106 NRepl = 10000
107 T = 5
108 NSteps = 16
109 Limit = NSteps
110 S0 = 50
111 mu = 0.1
112 sigma = 0.4
113 Paths = GBMHaltonBridge(S0, mu, sigma, T, NSteps,
        NRepl, Limit)
114 r = mu
115 NSamples = NSteps
116 K = 55
117 Payoff<-matrix()
118 for(p in 1:length(Paths[,1])){
119     Payoff[p] = max(0, mean(Paths[p,2:(NSamples+1)]) -
        K)
120 }
121 P = mean( exp(-r*T) * matrix(Payoff))

```

R code Exa 8.17 Improving the estimate of the option Delta by Common Random Numbers

```
1 require(varbvs)
```

```

2 require(fBasics)
3 norm.interval = function(data, variance = var(data),
  conf.level = 0.95) {
4   z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
5   xbar = mean(data)
6   sdx = sqrt(variance/length(data))
7   c(xbar - z * sdx, xbar + z * sdx)
8 }
9 BlsDeltaMCNaive<-function(S0,K,r,T,sigma,dS,NRepl){
10   nuT = (r - 0.5*sigma^2)*T
11   siT = sigma * sqrt(T)
12   Payoff1<-matrix()
13   Payoff2<-matrix()
14   for (i in 1:NRepl){
15     Payoff1[i] = max(0, S0*exp(nuT+siT*randn(1,1))-K
16       )
17     Payoff2[i] = max(0, (S0+dS)*exp(nuT+siT*randn
18       (1,1))-K)
19   }
20   SampleDiff = exp(-r*T)*(Payoff2 - Payoff1)/dS
21   parameter_estimation<-.normFit(SampleDiff)
22   ci<-norm.interval(SampleDiff)
23   return(c(parameter_estimation,ci))
24 }
25 set.seed(762567)
26 S0=50
27 K=52
28 r=0.05
29 T=5/12
30 sigma = 0.4
31 NRepl=50000
32 dS = 0.5
33 BlsDeltaMCNaive(S0,K,r,T,sigma,dS,NRepl)

```

R code Exa 8.18 Estimating the option Delta by a pathwise estimator

```

1  require(varbvs)
2  require(fBasics)
3  norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
4    z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
5    xbar = mean(data)
6    sdx = sqrt(variance/length(data))
7    c(xbar - z * sdx, xbar + z * sdx)
8  }
9
10 BlsDeltaMCPPath<-function(S0,K,r,T,sigma,NRepl){
11   nuT = (r - 0.5*sigma^2)*T
12   siT = sigma * sqrt(T)
13   VLogn<-matrix()
14   for (i in 1:NRepl){
15     VLogn[i] = exp(nuT+siT*randn(1,1))
16   }
17   SampleDelta = exp(-r*T) * VLogn * (S0*VLogn > K)
18   parameter_estimation<-.normFit(SampleDelta)
19   ci<-norm.interval(SampleDelta)
20   return(c(parameter_estimation,ci))
21 }
22
23 set.seed(3725678)
24 S0=50
25 K=52
26 r=0.05
27 T=5/12
28 sigma = 0.4
29 NRepl=50000
30 BlsDeltaMCPPath(S0,K,r,T,sigma,NRepl)

```

Chapter 9

Option Pricing by Finite Difference Methods

R code Exa 9.3 price a European vanilla put by a straightforward explicit scheme

```
1 require(pracma)
2
3 EuPutExpl<-function(S0,K,r,T,sigma,Smax,dS,dt){
4   # set up grid and adjust increments if necessary
5   M = round(Smax/dS)
6   dS = Smax/M
7   N = round(T/dt)
8   dt = T/N
9   matval = matrix(0,M+1,N+1)
10  vetS = seq(0,Smax,length=M+1)
11  veti = 0:M
12  vetj = 0:N
13  # set up boundary conditions
14  for (k in 1:(M+1)){
15    matval[k,N+1] = max(K-vetS[k],0)
16  }
17  matval[1,] = K*exp(-r*dt*(N-vetj))
18  matval[M+1,] = 0
```



```

19 # set up coefficients
20 a = 0.5*dt*(sigma^2*veti - r)*veti
21 b = 1- dt*(sigma^2*veti^2 + r)
22 c = 0.5*dt*(sigma^2*veti + r)*veti
23 # solve backward in time
24 for (j in N:1){
25   for (i in 2:M){
26     matval[i,j] = a[i]*matval[i-1,j+1] + b[i]*
      matval[i,j+1]+ c[i]*matval[i+1,j+1]
27   }
28 }
29 # return price , possibly by linear interpolation
   outside the grid
30 price = interp1(vetS, matval[,1], S0)
31 return(price)
32 }
33 EuPutExpl(50,50,0.1,5/12,0.4,100,2,5/1200)
34 EuPutExpl(50,50,0.1,5/12,0.3,100,2,5/1200)
35 EuPutExpl(50,50,0.1,5/12,0.3,100,1.5,5/1200)
36 EuPutExpl(50,50,0.1,5/12,0.3,100,1,5/1200)

```

R code Exa 9.4 price a vanilla European option by a fully implicit method

```

1 require(pracma)
2
3 EuPutImpl<-function(S0,K,r,T,sigma,Smax,dS,dt){
4   # set up grid and adjust increments if necessary
5   M = round(Smax/dS)
6   dS = Smax/M
7   N = round(T/dt)
8   dt = T/N
9   matval = matrix(0,M+1,N+1)
10  vetS = seq(0,Smax,length=M+1)
11  veti = 0:M
12  vetj = 0:N

```

```

13 # set up boundary conditions
14 for (k in 1:(M+1)){
15     matval[k,N+1] = max(K-vetS[k],0)
16 }
17 matval[1,] = K*exp(-r*dt*(N-vetj))
18 matval[M+1,] = 0
19 # set up the tridiagonal coefficients matrix
20 a = 0.5*(r*dt*veti-sigma^2*dt*(veti^2))
21 b = 1+sigma^2*dt*(veti^2)+r*dt
22 c = -0.5*(r*dt*veti+sigma^2*dt*(veti^2))
23 zero<-matrix(0,1,M-1)
24 coeff = (diag(c(0,a[3:M],0)))[1:M-1,2:M] + diag(b
    [2:M]) + (rbind(diag(c[2:M-1]),zero))[2:M,1:(M
    -1)]
25 L = lu(coeff)$L
26 U = lu(coeff)$U
27 # solve the sequence of linear systems
28 aux = matrix(0,M-1,1)
29 for (j in N:1){
30     aux[1] = - a[2] * matval[1,j]
31     # other term from BC is zero
32     matval[2:M,j] = solve(U,(solve(L,(matval[2:M,j
    +1] + aux))))
33 }
34 # return price, possibly by linear interpolation
    outside the grid
35 price = interp1(vetS, matval[,1], S0)
36 return(price)
37 }
38 EuPutImpl(50,50,0.1,5/12,0.4,100,0.5,5/2400)

```

R code Exa 9.5 price a down and out put option by the Crank Nicolson method

```
1 require(pracma)
```

```

2
3 DOPutCK<-function(S0,K,r,T,sigma,Sb,Smax,dS,dt){
4   # set up grid and adjust increments if necessary
5   M = round((Smax-Sb)/dS)
6   dS = (Smax-Sb)/M
7   N = round(T/dt)
8   dt = T/N
9   matval = matrix(0,M+1,N+1)
10  vetS = seq(Sb,Smax,length = M+1)
11  veti = vetS / dS
12  vetj = 0:N
13  # set up boundary conditions
14  for (k in 1:(M+1)){
15    matval[k,N+1] = max(K-vetS[k],0)
16  }
17  matval[1,] = 0
18  matval[M+1,] = 0
19  # set up the coefficients matrix
20  alpha = 0.25*dt*( sigma^2*(veti^2) - r*veti)
21  beta = -dt*0.5*( sigma^2*(veti^2) + r )
22  gamma = 0.25*dt*( sigma^2*(veti^2) + r*veti )
23  zero<-matrix(0,1,M-1)
24  M1 = -(diag(c(0,alpha[3:M],0)))[1:M-1,2:M] + diag
        (1-beta[2:M]) - (rbind(diag(gamma[2:M-1]),zero)
        )[2:M,1:(M-1)]
25  L = lu(M1)$L
26  U = lu(M1)$U
27  M2 = (diag(c(0,alpha[3:M],0)))[1:M-1,2:M] + diag
        (1+beta[2:M]) + (rbind(diag(gamma[2:M-1]),zero)
        )[2:M,1:(M-1)]
28  # solve the sequence of linear systems
29  for (j in N:1){
30    matval[2:M,j] = solve(U,solve(L,(M2%*%matval[2:M
        ,j+1])))
31  }
32  # return price , possibly by linear interpolation
        outside the grid
33  price = interp1(vetS, matval[,1], S0)

```

```
34     return(price)
35 }
36 DOPutCK(50,50,0.1,5/12,0.4,40,100,0.5,1/120)
```

Chapter 10

Dynamic Programming

R code Exa 10.4 Simple asset allocation problem under uncertainty Monte Carlo sampling

```
1 require(varbvs)
2 require(pracma)
3 require(zeallot)
4 OptFolioMC<-function(W0,S0,mu,sigma,r,T,NScen,utilf)
  {
5   muT = (mu - 0.5*sigma^2)*T
6   sigmaT = sigma*sqrt(T)
7   R = exp(r*T)
8   NormSamples = muT + sigmaT*randn(NScen,1)
9   ExcessRets = exp(NormSamples) - R
10  MExpectedUtility<-function(x){
11    -mean(utilf(W0*((x*ExcessRets) + R)))
12  }
13  share = fminbnd(MExpectedUtility, 0, 1)
14  return(share)
15 }
16 set.seed(294)
17 share = OptFolioMC(1000,50,0.1,0.4,0.05,1,10000,log)
18 share$xmin
19
```

```

20
21 set.seed(2947)
22 share = OptFolioMC(1000,50,0.1,0.4,0.05,1,5000000,
    log)
23 share$xmin
24
25 OptFolioGauss<-function(W0,S0,mu,sigma,r,T,NScen,
    utilf){
26     muT = (mu - 0.5*sigma^2)*T
27     sigmaT = sigma*sqrt(T)
28     R = exp(r*T)
29     print(GaussHermite(muT,sigmaT^2,NScen))
30     c(x,w) %<-% GaussHermite(muT,sigmaT^2,NScen)
31     ExcessRets = exp(x) - R
32     MExpectedUtility<-function(x){
33         - Re(dot(Re(w), utilf(W0*((Re(x)*ExcessRets) +
            R))))
34     }
35     share = fminbnd(MExpectedUtility, 0, 1)
36 }
37
38
39 GaussHermite<-function(mu,sigma2,N){
40     HPoly1 = c(1/pi^0.25)
41     HPoly2 = c(sqrt(2) / pi^0.25, 0)
42     for (j in 1:(N-1)){
43         HPoly3 = c(sqrt(2/(j+1)) * HPoly2 , 0) - c(0, 0,
            sqrt(j/(j+1))*HPoly1)
44         HPoly1 = HPoly2
45         HPoly2 = HPoly3
46     }
47     x1 = polyroot(HPoly3)
48     w1 = matrix(0,N,1)
49     for (i in 1:N){
50         w1[i] = 1/(N)/(polyval(HPoly1, x1[i]))^2
51     }
52     x = sort(x1*sqrt(2*sigma2)+mu)
53     index= order(x1*sqrt(2*sigma2)+mu)

```

```

54     w = w1[index]/sqrt(pi)
55     return(list(x,w))
56 }
57
58 share = OptFolioGauss(1000,50 ,0.1, 0.4, 0.05,1,2,
    log)
59 share
60
61 #different answers as R polyroot function
    calculating different roots than matlab.

```
