R Textbook Companion for Numerical Methods in Finance and Economics: A MATLAB-Based Introduction by Paolo Brandimarte[1]

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

July 18, 2025

[1]Funded 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

Chapter 2

Financial Theory

R code Exa 2.7 Basic theory of interest rates

```
mypvvar <- function(cf,r) {
  # get number of periods
  n = length(cf)
  #1 get vector of discount factors
  df<-matrix(0,n)
  for (i in 0:n-1){
    df[i+1] = 1/(1+r)^(i)
  }
  #compute result
  pv = cf*df
}

cf<-c(0, 8, 8, 8, 8, 108)
pv = mypvvar (cf ,0.08)
sum(pv)

pv = mypvvar (cf ,0.09)
sum(pv)

pv = mypvvar (cf ,0.07)
sum(pv)
```

R code Exa 2.8 Basic theory of interest rates

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

R code Exa 2.9 Basic pricing of fixed income securities

```
mypvvar <- function(cf,r) {
  # get number of periods
  n = length(cf)
  #1 get vector of discount factors
  df<-matrix(0,n)
  for (i in 0:n-1){
    df[i+1] = 1/(1+r)^(i)
  }
  #compute result
  pv = cf*df
}

r1=0.08
r2=0.09
P1=100/(1+r1)^5
P1
```

```
P2=100/(1+r2)^5
P2
(P2-P1)/P1

P1=100/(1+r1)^20
P1
P2=100/(1+r2)^20
P2
(P2-P1)/P1

cf1<-c(0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 108)
cf2<-c(0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 104)
P1=mypvvar(cf1,0.08)
P1 = sum(P1)
P1
P2=mypvvar(cf1,0.09)
P2 = sum(P2)
P2
(P2-P1)/P1
P1=mypvvar(cf2,0.08)
P1 = sum(P1)
P1
P2=mypvvar(cf2,0.09)
P2 = sum(P2)
P2
(P2-P1)/P1
```

R code Exa 2.10 Interest rate sensitivity and bond portfolio immunization

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

```
  for (i in 0:n-1){
    df[i+1] = 1/(1+r)^(i)
  }
  #compute result
  pv = cf*df
}

cfdur <- function(cf,yld) {
  #   CFDUR  Cash flow duration and modified duration.
  #   [D,MD] = CFDUR(CF,YLD) calculates the duration D and modified duration
  #   (volatility) MD of a cash flow given the cash flow, CF, and the periodic
  #   yield, YLD.
  #
  #   For example, nine payments of $2.50 and a final payment of $102.50 with
  #   a yield of 2.5% returns a duration of 8.97 periods and a modified duration
  #   of 8.75 periods.
  #
  #   See also BONDCONV, BONDDUR, CFCONV.
  #   Copyright 1995-2006 The MathWorks, Inc.
  x<-dim(cf)
  rowcf = x[1]
  colcf = x[2]
  if (rowcf == 1){
    cf = t(cf)
    colcf = 1
  }

  if (colcf > 1 ){
    if (length(yld) == 1){
      yld = yld*matrix(1,colcf)
    }
  }

```

```
  pv = matrix(0,colcf)
  d = pv
  md = pv
  m = length(cf[,1])
  fac = t((1:m))
  for (loop in 1:colcf){
    # Compound the yield
    rates = (matrix(data = 1,ncol = m,nrow = 1)*(1+yld[loop]))^fac
    # find the net present value
    pv[loop] = sum(mypvvar(c(0,cf[,loop]),yld[loop]))
    # duration
    d[loop] = sum(cf[,loop]/(rates)*fac)/pv[loop]
    # modified duration
    md[loop] = d[loop]/(1+yld[loop])
  }
  cat("Duration of cash flow:",d,"\n")
  cat("Modified Duration (volatility) of cash flow:",md,"\n")
  return(list(d=d,dm=md))
}

cfconv <- function(cf,yld) {
  #   CFCONV Cash flow convexity.
  #   CX = CFCONV(CF,YLD) calculates the convexity C of a cash flow
  #   given the cash flow, CF, and the periodic yield, YLD.
  #
  #   For example, nine payments of $2.50 and a final payment of $102.50
  #   with a yield of 2.5% returns a convexity of 90.45 periods.
  #
  #   See also BONDCONV, BONDDUR, CFDUR.

  #       Copyright 1995-2006 The MathWorks, Inc.
```

```
  x<-dim(cf)
  rowcf = x[1]
  colcf = x[2]
  if (rowcf == 1){
    cf = t(cf)
    colcf = 1
  }

  if (colcf > 1 ){
    if (length(yld) == 1){
      yld = yld*matrix(1,colcf)
    }
  }

  pv = matrix(0,colcf)
  cx = pv
  m = length(cf[,1])
  fac = t((1:m))
  for (loop in 1:colcf){
    #Compound the yield
    rates = (matrix(data = 1,ncol = m,nrow = 1)*(1+yld[loop]))^fac
    #find the net present value
    pv[loop] = sum(mypvvar(c(0,cf[,loop]),yld[loop]))
    cx[loop] = sum(cf[,loop]/(rates)*fac*(fac+1))/((1+yld[loop])^2*pv[loop])
  }
  cat("Cash flow convexity:",cx,"\n")
  return(cx)
}

cf<-c(10,10,10,10)
p1=abs(pv.uneven(0.05,cf))
p2=abs(pv.uneven(0.055,cf))
(p2-p1)
cf<-matrix(cf,nrow = 1,ncol = 4)
yld<-matrix(data = c(0.05),nrow = 1,ncol = 1)
```

```
x<-cfdur(cf,yld)
d<-x$d
dm<-x$dm
cv=cfconv(cf,yld)
-dm*p1*0.005
-dm*p1*0.005+0.5*cv*p1*(0.005)^2
```

R code Exa 2.24 Black Scholes model in MATLAB

```
#install.packages("OptionPricing")
require(OptionPricing)
C0 = BS_EC(T = 5/12, K = 50, r = 0.1, sigma = 0.3,
   S0 = 50)
dS = 2
C1 = BS_EC(T = 5/12, K = 50, r = 0.1, sigma = 0.3,
   S0 = 50+dS)
C0[1]+ C0[2]*dS + 0.5*C0[3]*dS-2

#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

```
require(rootSolve)
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
sort(uniroot.all(fun, c(0, 8)),decreasing = TRUE)
p2<-c(1, -36.001, 546, -4536, 22449, -67284, 118124, -109584, 40320)
1/polyroot(p2)
```

R code Exa 3.4 Vector and matrix norms

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

R code Exa 3.6 Vector and matrix norms

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

R code Exa 3.7 Condition number for a matrix

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

R code Exa 3.8 Condition number for a matrix

```
N = 20
A<-diag(N)

for (i in 1:N){
  for (j in (i+1):N){
    A[i,j] = -1
  }
}

b=-rep(1,N)
```

```
b[N] = 1
solve(A,b)

b[N] = 1.00001
solve(A,b)

1/rcond(A,'I')
2^18
0.00001 * 2^18
```

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

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

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

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

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

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

```
Jacobi <- function(A,b,x0,eps,MaxIter) {
  dA = diag(A)
  C = A - diag(dA)
  Dinv = diag(1/dA)
  B = - Dinv %*% C
  b1 = Dinv %*% b
  oldx = x0

  for (i in 1:MaxIter){
    x = B %*% oldx + b1
    if (norm(x-oldx) < eps*norm(matrix(oldx))){
      break
    }
    oldx = x
  }
  cat("Case of Matrix:",A,"\n")
  cat("Terminated after iterations:",i,"\n")
  cat("Jacobi:",x,"\n")
  cat("Exact:",solve(A,b),"\n")
}

A1 <-matrix(c(3, 1, 1, 0, 1, 5, - 1, 2, 1, 0, 3, 1,
   0, 1, 1, 4),nrow = 4,ncol = 4,byrow = T)
b<-matrix(c(1, 4, -2, 1),nrow = 4,ncol = 1,byrow = T
   )
Jacobi(A1,b,rep(0,4),1e-08,10000)

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
   )
Jacobi(A2,b,rep(0,4),1e-08,10000)
```

```
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)
Jacobi(A3,b,rep(0,4),1e-08,10000)
```

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

```
SORGaussSeidel <- function(A, b,x0, omega, eps,
   MaxIter) {
  oldx = x0
  x = x0
  N = length(x0)
  omega1 = 1 - omega
  for (k in 1:MaxIter){
    for (i in 1:(N-1)){
      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]
      x[i] = omega * z + omega1 * oldx[i]
    }
    if(norm(matrix(x-oldx))<eps*norm(matrix(oldx))){
      break
    }
    oldx = x
  }
  result$x = x
  result$k = k
  return(result)
}

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)
b<-matrix(c(1, 4, -2, 1),nrow = 4,ncol = 1,byrow = T
   )
omega = seq(0,2,0.1)
N = length(omega)
```

```
NumIterations = rep(0,N)
for (i in 1:N){
  result = list()
  result = SORGaussSeidel(A2,b,rep(0,4),omega[i],1e
     -08,1000)
  NumIterations[i] = result$k
}
plot(omega,NumIterations)
lines(omega,NumIterations)
```

R code Exa 3.15 FUNCTION APPROXIMATION AND INTERPOLATION

```
xdata<-c(1, 5, 10, 30, 50)
ydata<-log(xdata)
p = coef(lm(ydata ~ xdata + I(xdata^2)))
p

x = c(1, 5, 10, 30, 50)
y = log(x)
plot(x,y)
lm2 = lm(y~x+I(x^2))
lm4 = lm(y~x+I(x^2)+I(x^3)+I(x^4))
xplot=seq(from = 0,to = 50,by = .1)
lines(xplot,predict(lm4,newdata=data.frame(x=xplot))
   , col="blue")
# and so on
lines(xplot,predict(lm2,newdata=data.frame(x=xplot))
   , col="red")

#http://www.utstat.utoronto.ca/reid/sta414/Rsession-
   polys.pdf
```

R code Exa 3.16 Elementary polynomial interpolation

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

R code Exa 3.17 Elementary polynomial interpolation

```r
runge <- function(x) {
  1/(1+25*x^2)
}

EquiNodes = -5:5
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)))

x = -5:5
y = runge(x)
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))
xplot=seq(from = -5,to = 5,by = .01)
plot(x,y,asp = .33)
lines(x,y)
grid(10,10)
lines(xplot,predict(lm10,newdata=data.frame(x=xplot)), col="blue")

ChebNodes = 5*cos(pi*(11 - (1:11) + 0.5)/11)
```

```
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)))

x = 5*cos(pi*(11 - (1:11) + 0.5)/11)
y = runge(x)
plot(x,y,asp = 1)
lines(x,y)
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))
xplot=seq(from = -5,to = 5,by = .01)
lines(xplot,predict(lm10,newdata=data.frame(x=xplot)
    ), col="blue")
```

R code Exa 3.18 Interpolation by cubic splines

```
require(pracma)

x=1: 10
y<-c(8, 2.5, -2, 0, 5, 2, 4, 7 ,4.5, 2)
plot(x,y)
x2<-seq(from = 1, to = 10, by = 0.05)
y2=interp1(x,y,x2, 'spline')
lines(x2,y2)

x=1: 10
y<-c(8, 2.5, -2, 0, 5, 2, 4, 7 ,4.5, 2)
plot(x,y)
pp=cubicspline (x, y)
x2<-seq(from = 1, to = 10, by = 0.05)
y2 = ppval(pp,x2)
lines(x2,y2)

runge <- function(x) {
```

```
  1/(1+25*x^2)
}

# use 11 equispaced nodes
EquiNodes11 = -5:5
ppeq11 = cubicspline(EquiNodes11,runge(EquiNodes11))
xplot=seq(from = -5, to = 5, by = 0.01)

# use 20 equispaced nodes
EquiNodes20 = linspace(-5,5,20)
ppeq20 = cubicspline(EquiNodes20,runge(EquiNodes20))
xplot=seq(from = -5, to = 5, by = 0.01)

# use 21 equispaced nodes
EquiNodes21 = linspace(-5,5,21)
ppeq21 = cubicspline(EquiNodes21,runge(EquiNodes21))
xplot=seq(from = -5, to = 5, by = 0.01)

par(mfrow=c(3,1))
plot(EquiNodes11,runge(EquiNodes11),asp = 1)
lines(EquiNodes11,runge(EquiNodes11))
lines(xplot,ppval(ppeq11,xplot), col="blue")
plot(EquiNodes20,runge(EquiNodes20),asp = 1)
lines(EquiNodes20,runge(EquiNodes20))
lines(xplot,ppval(ppeq20,xplot), col="blue")
plot(EquiNodes21,runge(EquiNodes21),asp = 1)
lines(EquiNodes21,runge(EquiNodes21))
lines(xplot,ppval(ppeq21,xplot), col="blue")
```

R code Exa 3.22 Bisection method

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

```
ans$root
ans$f.root

#z<-curve(f)

f <- function(x) {
  x^2
}
ans2 = uniroot(f,c(-1,1),tol=2^-52)
ans2$root
ans2$f.root
```

R code Exa 3.23 Newtons method

```
require(OptionPricing)
c=BS_EC(K=54, r = 0.07, sigma = 0.3, T = 5/12, S0 =
   50)
c[1]
y <- function(y) (BS_EC(K=54, r = 0.07, sigma = y, T
    = 5/12, S0 = 50)[1]-2.846575)
h<-Vectorize(y)
uniroot(y,c(-1,1))

f <- Vectorize(function(y) (BS_EC(K=54, r = 0.07,
   sigma = y, T = 5/12, S0 = 50)[1]-2.846575))
curve(f)
```

R code Exa 3.24 Optimization based solution of non linear equations

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

```
lines(vx,f(vx))
fsolve(f,1)
fsolve(f,2)

#fsolve answers defer from matlab (book)
```

Chapter 4

Numerical Integration Deterministic and Monte Carlo Methods

R code Exa 4.1 Numerical integration in MATLAB

```
#require(pracma)
f <- function(x) {
  exp(-x)*sin(10*x)
}

integrate(f ,0,2*pi)

integrate(f ,0,2*pi,abs.tol = 10e-6)

integrate(f ,0,2*pi,abs.tol = 10e-8)

quadl(f ,0,2*pi)
```

R code Exa 4.2 MONTE CARLO INTEGRATION

```r
set.seed(12345)
mean(exp(runif(10)))
mean(exp(runif(10)))
mean(exp(runif(10)))
mean(exp(runif(1000000)))
mean(exp(runif(1000000)))
mean(exp(runif(1000000)))
```

R code Exa 4.3 MONTE CARLO INTEGRATION

```r
BlsMC1 <- function(S0, K , r , T , sigma, NRepl) {
  nuT = (r - 0.5*sigma**2)*T
  siT = sigma * sqrt(T)
  DiscPayoff = exp(-r*T) *pmax(0,(S0*exp(nuT+siT*
     rnorm(NRepl))-K))
  Price = mean(DiscPayoff)
  return(Price)
}
S0=50
K=60
r=0.05
T=1
sigma=0.2
set.seed(547)
BlsMC1(S0,K,r ,T, sigma, 1000)
BlsMC1(S0,K,r ,T, sigma, 1000)
BlsMC1(S0,K,r ,T, sigma, 1000)
BlsMC1(S0,K,r ,T, sigma, 1000000)
BlsMC1(S0,K,r ,T, sigma, 1000000)
BlsMC1(S0,K,r ,T, sigma, 1000000)
```

R code Exa 4.4 Generating pseudorandom numbers

```
LCG <- function(a,c,m,seed,N) {
  ZSeq <-matrix(0,N,1)
  USeq <-matrix(0,N,1)
  for (i in 1:N){
    seed = (a*seed+c) %% m
    ZSeq[i] = seed
    USeq[i] = seed/m
  }
  result<-list(ZSeq,USeq)
  return(result)
}

a = 5
c = 3
m = 16
seed = 7
N = 20
LCG(a,c,m,seed,N)
```

R code Exa 4.5 Generating pseudorandom numbers

```
LCG <- function(a,c,m,seed,N) {
  ZSeq <-matrix(0,N,1)
  USeq <-matrix(0,N,1)
  for (i in 1:N){
    seed = (a*seed+c) %% m
    ZSeq[i] = seed
    USeq[i] = seed/m
  }
  result<-list(ZSeq,USeq)
  return(result)
}

m = 2048;
a = 65;
```

```
c = 1;
seed = 0;
U = LCG(a,c,m,seed, 2048);
plot(unlist(U[1])[1:m-1],unlist(U[2])[2:m])
plot(unlist(U[1])[1:511],unlist(U[2])[2:512])

a=1365;
c=1 ;
U = LCG(a,c,m,seed, 2048)
plot(unlist(U[1])[1:m-1],unlist(U[2])[2:m])
```

R code Exa 4.6 Inverse transform method

```
set.seed(64657)
rexp(1, 1/1)

EmpiricalDrnd <- function(values, probs, howmany) {
  cumprobs = cumsum(probs)
  N = length(probs)
  samples = matrix(0,howmany, 1)
  for (k in 1:howmany){
    loc=sum(runif(1)*cumprobs[N] > cumprobs) + 1;
    samples[k]=values[loc]
  }
  return(samples)
}

values=1:5
probs<-c(0.1, 0.2, 0.4, 0.2, 0.1)
samples=EmpiricalDrnd(values ,probs ,10000)
hist(x = samples)
```

R code Exa 4.8 Generating normal variates by the polar approach

```r
LCG <- function(a,c,m,seed,N) {
  ZSeq <-matrix(0,N,1)
  USeq <-matrix(0,N,1)
  for (i in 1:N){
    seed = (a*seed+c) %% m
    ZSeq[i] = seed
    USeq[i] = seed/m
  }
  result<-list(ZSeq,USeq)
  return(result)
}

m = 2048;
a = 1229;
c = 1;
N = m-2;
seed = 0;
U = LCG(a,c,m,seed,N);
index<-seq(1,N-1,2)
U1 = unlist(U[2])[index]
index<-seq(2,N,2)
U2 = unlist(U[2])[index]
X=sqrt(-2*log(U1))* cos(2*pi*U2);
Y=sqrt(-2*log(U1))* sin(2*pi*U2);
plot(X,Y)
X=sqrt(-2*log(U2))* cos(2*pi*U1);
Y=sqrt(-2*log(U2))* sin(2*pi*U1);
plot(X,Y)
```

R code Exa 4.9 Generating normal variates by the polar approach

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

```
set.seed(2392)

MultiNormrnd <- function(mu,sigma,howmany) {
  n = length(mu)
  Z = matrix(0,howmany,n)
  U = chol(sigma)
  for (i in 1:howmany){
    Z[i,] = t(mu) + t(matrix(rnorm(n))) %*% U
  }
  return(Z)
}

Z = MultiNormrnd(mu,Sigma,10000)
mean(Z[1])
mean(Z[2])
mean(Z[3])
cov(Z)
```

R code Exa 4.10 SETTING THE NUMBER OF REPLICATIONS

```
#require(fBasics)
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
     z = qnorm((1 - conf.level)/2, lower.tail =
        FALSE)
     xbar = mean(data)
     sdx = sqrt(variance/length(data))
     c(xbar - z * sdx, xbar + z * sdx)
     }

BlsMC2 <- function(S0,K,r,T,sigma,NRepl) {
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  DiscPayoff = exp(-r*T)*pmax(0, S0*exp(nuT+siT*
     rnorm(NRepl))-K);
```

```
  parameter_estimation<-.normFit(DiscPayoff)
  ci<-norm.interval(DiscPayoff)
  final<-list(parameter_estimation,ci)
  return(final)
}

set.seed(483762)
S0=50;
K=55;
r=0.05;
T=5/12;
sigma=0.2;
answer<-BlsMC2(S0,K,r,T,sigma,50000)
answer2<-BlsMC2(S0,K,r,T,sigma,1000000)
#(answer[[2]][2] - answer[[2]][1])/1.168224
#(answer2[[2]][2] - answer2[[2]][1])/1.173184
```

R code Exa 4.11 Antithetic sampling

```
#require(fBasics)
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
set.seed(337282)
X=exp(runif(100))
parameter_estimation<-.normFit(X)
ci<-norm.interval(X)
#from parameter estimation, value of mean =
   1.7031094
(ci[2]-ci[1])/1.7031094

```

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

R code Exa 4.12 BlsMCAV

```
suppressMessages(require(fBasics))
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}

BlsMCAV <- function(S0,K,r,T,sigma,NPairs) {
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  Veps = rnorm(NPairs);
  Payoff1 = pmax( 0 , S0*exp(nuT+siT*Veps) - K);
  Payoff2 = pmax( 0 , S0*exp(nuT+siT*(-Veps)) - K);
  DiscPayoff = exp(-r*T) * 0.5 * (Payoff1+Payoff2);
  parameter_estimation<-.normFit(DiscPayoff)
  ci<-norm.interval(DiscPayoff)
  answer<-list(parameter_estimation,ci)
  return(answer)
}

BlsMC2 <- function(S0,K,r,T,sigma,NRepl) {
```

```
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  DiscPayoff = exp(-r*T)*pmax(0, S0*exp(nuT+siT*
     rnorm(NRepl))-K);
  parameter_estimation<-.normFit(DiscPayoff)
  ci<-norm.interval(DiscPayoff)
  final<-list(parameter_estimation,ci)
  return(final)
}

MCButterfly <- function(S0,r,T,sigma,NRepl,K1,K2,K3)
    {
  nuT = (r-0.5*sigma^2)*T;
  siT = sigma*sqrt(T);
  Veps = rnorm(NRepl);
  Stocks = S0*exp(nuT + siT*Veps);
  In1 = which((Stocks > K1) & (Stocks < K2));
  In2 = which((Stocks >= K2) & (Stocks < K3));
  Payoff = exp(-r*T)*matrix(c((Stocks[In1]-K1),  (K3
     -Stocks[In2]), matrix(0,(NRepl - length(In1) -
     length(In2)),1)))
  parameter_estimation<-.normFit(Payoff)
  ci<-norm.interval(Payoff)
  final<-list(parameter_estimation,ci)
  return(final)
}

MCAVButterfly <- function(S0,r,T,sigma,NPairs,K1,K2,
   K3) {
  nuT = (r-0.5*sigma^2)*T;
  siT = sigma*sqrt(T);
  Veps = rnorm(NPairs);
  Stocks1 = S0*exp(nuT + siT*Veps);
  Stocks2 = S0*exp(nuT - siT*Veps);
  Payoff1 = matrix(0,NPairs,1);
  Payoff2 = matrix(0,NPairs,1);
  In = which((Stocks1 > K1) & (Stocks1 < K2));
  Payoff1[In] = (Stocks1[In] - K1);
```

```
  In = which((Stocks1 >= K2) & (Stocks1 < K3));
  Payoff1[In] = (K3 - Stocks1[In]);
  In = which((Stocks2 > K1) & (Stocks2 < K2));
  Payoff2[In] = (Stocks2[In] - K1);
  In = which((Stocks2 >= K2) & (Stocks2 < K3));
  Payoff2[In] = (K3 - Stocks2[In]);
  Payoff = 0.5 * exp(-r*T) * (Payoff1 + Payoff2);
  parameter_estimation<-.normFit(Payoff)
  ci<-norm.interval(Payoff)
  final<-list(parameter_estimation,ci)
  return(final)
}

set.seed(3374)
Y<-BlsMC2(50,50 ,0.05,1, 0.4,200000)
#from parameter estimation, value of mean = 9.11143
(Y[[2]][2]-Y[[2]][1])/9.11143

Z<-BlsMCAV(50,50,0.05,1,0.4,100000)
#from parameter estimation, value of mean = 9.020972
(Z[[2]][2]-Z[[2]][1])/9.020972

set.seed(39489378)
S0 = 60;
K1 = 55;
K2 = 60;
K3 = 65;
T = 5/12;
r = 0.1;
sigma = 0.4;
a<-MCButterfly(S0,r,T,sigma,100000,K1,K2,K3);
#from parameter estimation, value of mean = 0.6104167
(a[[2]][2]-a[[2]][1])/0.6104167

set.seed(72725)
b<-MCAVButterfly(S0,r ,T, sigma,50000,K1 ,K2,K3)
#from parameter estimation, value of mean =
```

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

R code Exa 4.14 Importance sampling

```
estpi <- function(m) {
  z=sqrt(1-runif(m)^2) ;
  out = 4*sum(z)/m;
  return(out)
}
set.seed(483272)
estpi(1000)
estpi(1000)
estpi(1000)

estpiIS <- function(m,L) {
  s= seq(0,1-1/L,1/L) + 1/(2*L)
  hvals = matrix(sqrt(1 - s^2))
  # get cumulative probabilities
  cs=apply(hvals,2,cumsum);
  est = matrix(0,m)
  for (j in 1:m){
    # locate sub-interval
    loc=sum(runif(1)*cs[L] > cs) +1;
    # sample uniformly within sub-interval
    x=(loc-1)/L + runif(1)/L;
    p=hvals[loc]/cs[L];
    est[j] = sqrt(1 - x^2)/(p*L);
  }
  plot(est)
  z = 4*sum(est)/m;
  return(z)
}

estpiIS(1000,10)
```

```
estpiIS(1000,10)
estpiIS(1000,10)
estpiIS(1000,100)
estpiIS(1000,100)
estpiIS(1000,100)
```

R code Exa 4.15 Generating Halton low discrepancy sequences

```
#require(fBasics)
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}

BlsMCIS <- function(S0,K,r,T,sigma,NRepl) {
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  ISnuT = log(K/S0) - 0.5*sigma^2*T;
  Veps = rnorm(NRepl);
  VY = ISnuT + siT*Veps;
  ISRatios = exp( (2*(nuT - ISnuT)*VY - nuT^2 +
     ISnuT^2)/2/siT^2);
  DiscPayoff = exp(-r*T)*pmax(0, (S0*exp(VY)-K));
  parameter_estimation<-.normFit(DiscPayoff*ISRatios
     )
  ci<-norm.interval(DiscPayoff*ISRatios)
  final<-list(parameter_estimation,ci)
  return(final)
}

Halton <- function(n,b) {
  n0 = n;
```

```
  h = 0;
  f = 1/b;
  while (n0 > 0){
    n1 = floor(n0/b);
    r = n0 - n1*b;
    h = h+f*r;
    f = f/b;
    n0=n1;
  }
  return(h)
}

seq = matrix(0,10)
for (i in 1:10){
  seq[i] = Halton(i,2);
}

GetHalton <- function(HowMany, Base) {
  Seq = matrix(0,HowMany,1)
  NumBits = 1+round(log(HowMany)/log(Base));
  VetBase = Base^(-(1:NumBits));
  WorkVet = matrix(0,1,NumBits);
  for (i in 1:HowMany){
    j = 1;
    ok = 0;
    while (ok == 0){
      WorkVet[j] = WorkVet[j]+1;
      if (WorkVet[j] < Base){
        ok = 1;
      }
      else{
        WorkVet[j] = 0;
        j = j+1;
      }
    }
    Seq[i] = sum(WorkVet * VetBase)
  }
  return(Seq)
```

```
}



plot(runif(100) ,runif(100))
plot(GetHalton(100,2) ,GetHalton (100,7))
plot(GetHalton(100,2), GetHalton(100,4))
```

R code Exa 4.16 Generating Halton low discrepancy sequences

```
require(pracma)
f <- function(x,y) {
  exp(-x*y) *(sin(6*pi*x)+cos(8*pi*y))
}

dblquad(f = f,xa = 0,xb = 1,ya = 0,yb = 1)

n <- seq(0,1,0.01)
multiarray = list();
multiarray <- meshgrid(n,n)
Z<-f(multiarray$X,multiarray$Y)


persp(multiarray$X[1,],multiarray$Y[,1],Z,theta=30,
   phi=30, expand=0.6,col='lightblue', shade=0.75,
   ltheta=120,ticktype='detailed')

set.seed(4837)
mean(f(runif(10000),runif(10000)))
mean(f(runif(10000),runif(10000)))
mean(f(runif(10000),runif(10000)))

GetHalton <- function(HowMany, Base) {
  Seq = matrix(0,HowMany,1)
  NumBits = 1+round(log(HowMany)/log(Base));
```

```
  VetBase = Base^(-(1:NumBits));
  WorkVet = matrix(0,1,NumBits);
  for (i in 1:HowMany){
    j = 1;
    ok = 0;
    while (ok == 0){
      WorkVet[j] = WorkVet[j]+1;
      if (WorkVet[j] < Base){
        ok = 1;
      }
      else{
        WorkVet[j] = 0;
        j = j+1;
      }
    }
    Seq[i] = sum(WorkVet * VetBase)
  }
  return(Seq)
}

seq2 = GetHalton(10000,2)
seq4 = GetHalton(10000,4)
seq5 = GetHalton(10000,5)
seq7 = GetHalton(10000,7)
mean(f(seq2,seq5))
mean(f(seq2,seq4))
mean(f(seq2,seq7))
mean(f(seq5,seq7))

set.seed(327439)
mean(f(runif(100),runif(100)))
mean(f(runif(500),runif(500)))
mean(f(runif(1000),runif(1000)))
mean(f(runif(1500),runif(1500)))
mean(f(runif(2000),runif(2000)))

mean(f(seq2[1:100],seq7[1:100]))
mean(f(seq2[1:500],seq7[1:500]))
```

```
mean(f(seq2[1:1000],seq7[1:1000]))
mean(f(seq2[1:1500],seq7[1:1500]))
mean(f(seq2[1:2000],seq7[1:2000]))
```

R code Exa 4.17 Generating Halton low discrepancy sequences

```
require(pracma)
require(fBasics)
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}

GetHalton <- function(HowMany, Base) {
  Seq = matrix(0,HowMany,1)
  NumBits = 1+round(log(HowMany)/log(Base));
  VetBase = Base^(-(1:NumBits));
  WorkVet = matrix(0,1,NumBits);
  for (i in 1:HowMany){
    j = 1;
    ok = 0;
    while (ok == 0){
      WorkVet[j] = WorkVet[j]+1;
      if (WorkVet[j] < Base){
        ok = 1;
      }
      else{
        WorkVet[j] = 0;
        j = j+1;
      }
    }
    Seq[i] = sum(WorkVet * VetBase)
```

```
  }
  return(Seq)
}

BlsHaltonBM <- function(S0,K,r,T,sigma,NPoints,Base1
    ,Base2) {
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  H1 = GetHalton(ceiling(NPoints/2),Base1);
  H2 = GetHalton(ceiling(NPoints/2),Base2);
  VLog = sqrt(-2*log(H1))
  Norm1 = VLog * cos(2*pi*H2)
  Norm2 = VLog * sin(2*pi*H2)
  Norm = rbind(Norm1,Norm2)
  DiscPayoff = exp(-r*T) * pmax( 0 , S0*exp(nuT+siT*
     Norm) - K);
  Price = mean(DiscPayoff);
  return(Price)
}

BlsHaltonBM(50,52,0.1,5/12,0.4,5000,2,7)
BlsHaltonBM(50,52,0.1,5/12,0.4,5000,11,7)
BlsHaltonBM(50,52,0.1,5/12,0.4,5000,2,4)

BlsMC2 <- function(S0,K,r,T,sigma,NRepl) {
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  DiscPayoff = exp(-r*T)*pmax(0, S0*exp(nuT+siT*
     rnorm(NRepl))-K);
  parameter_estimation<-.normFit(DiscPayoff)
  ci<-norm.interval(DiscPayoff)
  final<-list(parameter_estimation,ci)
  return(final)
}

set.seed(3726)
BlsMC2(50,52,0.1,5/12,0.4,5000)
BlsMC2(50,52,0.1,5/12,0.4,5000)
```

```
BlsMC2(50,52,0.1,5/12,0.4,5000)

BlsHaltonINV <- function(S0,X,r,T,sigma,NPoints,Base) {
  nuT = (r - 0.5*sigma^2)*T;
  siT = sigma * sqrt(T);
  # Use inverse transform to generate standard normals
  H = GetHalton(NPoints,Base);
  Veps = qnorm(H);
  DiscPayoff = exp(-r*T)*pmax(0,S0*exp(nuT+siT*Veps)-X);
  Price = mean(DiscPayoff);
  return(Price)
}

BlsHaltonINV(50,52,0.1,5/12,0.4,1000,2)
BlsHaltonINV(50,52,0.1,5/12,0.4,2000,2)
BlsHaltonINV(50,52,0.1,5/12,0.4,5000,2)
BlsHaltonINV(50,52,0.1,5/12,0.4,1000,2)
BlsHaltonINV(50,52,0.1,5/12,0.4,10000,2)
BlsHaltonINV(50,52,0.1,5/12,0.4,50000,2)

GetHalton(17,17)

BlsHaltonINV(50,52,0.1,5/12,0.4,1000,499)
BlsHaltonINV(50,52,0.1,5/12,0.4,2000,499)
BlsHaltonINV(50,52,0.1,5/12,0.4,5000,499)
BlsHaltonINV(50,52 ,0.1,5/12,0.4,10000,499)
BlsHaltonINV(50,52,0.1,5/12,0.4,50000,499)

plot(GetHalton(1000,109), GetHalton(1000,113))
```

R code Exa 4.18 Generating Sobol low discrepancy sequences

```r
require(bitops)
GetDirNumbers <- function(p,m0,n) {
  degree = length(p)-1;
  p = p[2:degree];
  m = cbind(m0 , matrix(0,1,n-degree))
  for (i in (degree+1):n){
    m[i] = bitXor(m[i-degree], 2^degree * m[i-degree])
    for (j in 1:(degree-1)){
      m[i] = bitXor(m[i], 2^j * p[j] * m[i-j]);
    }
  }
  v=m/(2^(1:length(m)))
  final<-list()
  final$v<-v
  final$m<-m
  return(final)
}

p <-matrix(c(1,0, 1, 1),nrow = 1,ncol = 4)
m0 <-matrix(c(1,3, 7),nrow = 1,ncol = 3)
ans<-GetDirNumbers(p,m0, 6)
ans$v
ans$m
```

R code Exa 4.19 Generating Sobol low discrepancy sequences

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

R code Exa 4.20 Generating Sobol low discrepancy sequences

```
require(bitops)
options(warn=-1)

GetDirNumbers <- function(p,m0,n) {
  degree = length(p)-1;
  p = p[2:degree];
  m = cbind(m0 , matrix(0,1,n-degree))
  for (i in (degree+1):n){
    m[i] = bitXor(m[i-degree], 2^degree * m[i-degree])
    for (j in 1:(degree-1)){
      m[i] = bitXor(m[i], 2^j * p[j] * m[i-j]);
    }
  }
  v=m/(2^(1:length(m)))
  final<-list()
  final$v<-v
  final$m<-m
  return(final)
}

p <-matrix(c(1,0, 1, 1),nrow = 1,ncol = 4)
m0 <-matrix(c(1,3, 7),nrow = 1,ncol = 3)
ans<-GetDirNumbers(p,m0, 6)
ans$v
ans$m

GetSobol <- function(GenNumbers, x0, HowMany) {
  Nbits = 20;
  factor = 2^Nbits;
  BitNumbers = GenNumbers * factor;
  SobSeq = matrix(0,HowMany + 1, 1);
```

```
  SobSeq[1] = as.integer(x0*factor);
  for (i in 1:HowMany){
    c = pmin(which( intToBits(i-1) [1:16] == 0 ));
    SobSeq[i+1] = bitXor(SobSeq[i], BitNumbers[c]);
  }
  SobSeq = SobSeq / factor;
  return(SobSeq)
}
GetSobol(ans$v,0,10)


p <-matrix(c(1,0, 1, 1,1,1),nrow = 1,ncol = 6)
m0 <-matrix(c(1,3, 5, 9, 11),nrow = 1,ncol = 5)
ans<-GetDirNumbers(p,m0, 10)

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

```
require(PopED)

transport <- function(xmin, dx, xmax, dt, tmax, c,
    f0) {
  N = ceiling((xmax - xmin) / dx);
  xmax = xmin + N*dx;
  M = ceiling(tmax/dt);
  k1 = 1 - dt*c/dx;
  k2 = dt*c/dx;
  solution = matrix(0,N+1,M+1);
  vetx = seq(xmin,xmax,dx)
  for (i in 1:N+1) {
    solution[i,1] = feval(f0,vetx[i]);
  }
  fixedvalue = solution[1,1];
  # this is needed because of finite domain
  plot(solution[,1])
  for (j in 1:M){
    solution[,j+1] = k1*solution[,j] + k2* c(
```

```
        fixedvalue,solution[1:N,j]);
    lines(solution[,j])
  }
  return(solution)
}

f0transp <- function(x) {
  if(x < (-1)){
    y = 0
  } else if (x <= 0){
    y=x+1;
  } else{
    y = 1;
  }
}

xmin = -2;
xmax = 3;
dx = 0.05;
tmax = 2;
dt = 0.01;
c = 1;
sol = transport(xmin, dx, xmax, dt, tmax, c ,
   f0transp)

TransportPlot <- function(xmin, dx, xmax, times, sol
   ) {
  par(mfrow=c(2,2))
  plot(seq(xmin,xmax,dx), sol[,times[1]])
  lines(seq(xmin,xmax,dx), sol[,times[1]])
  plot(seq(xmin,xmax,dx), sol[,times[2]])
  lines(seq(xmin,xmax,dx), sol[,times[2]])
  plot(seq(xmin,xmax,dx), sol[,times[3]])
  lines(seq(xmin,xmax,dx), sol[,times[3]])
  plot(seq(xmin,xmax,dx), sol[,times[4]])
  lines(seq(xmin,xmax,dx), sol[,times[4]])
}

```

```
TransportPlot(xmin, dx, xmax, c(1, 51, 101, 201),
    sol)

dx = 0.01
sol = transport(xmin, dx, xmax, dt, tmax, c ,
    f0transp)
TransportPlot(xmin, dx, xmax, c(1, 51, 101, 201),
    sol)

#Blow-outs
dx = 0.005
sol = transport(xmin, dx, xmax, dt, tmax, c ,
    f0transp)
TransportPlot(xmin, dx, xmax, c(1, 51, 101, 201),
    sol)
```

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

```
HeatExpl <- function(deltax, deltat, tmax) {
  N = round(1/deltax)
  M = round(tmax/deltat)
  sol = matrix(0,N+1,M+1)
  rho = deltat / (deltax)^2
  rho2 = 1-2*rho
  vetx = seq(0,1,deltax)
  for (i in 2:ceiling((N+1)/2)){
    sol[i,1] = 2*vetx[i]
    sol[N+2-i,1] = sol[i,1]
  }
  for (j in 1:M){
    for (i in 2:N){
      sol[i,j+1] = rho*sol[i-1,j] + rho2*sol[i,j] +
         rho*sol[i+1,j]
    }
  }
```

```
  return(sol)
}

dx = 0.1;
dt = 0.001;
tmax = dt*100;
sol=HeatExpl(dx, dt , tmax)

par(mfrow=c(2,2))
plot(seq(0,1,dx), sol[,1])
lines(seq(0,1,dx), sol[,1])
plot(seq(0,1,dx), sol[,11])
lines(seq(0,1,dx), sol[,11])
plot(seq(0,1,dx), sol[,51])
lines(seq(0,1,dx), sol[,51])
plot(seq(0,1,dx), sol[,101])
lines(seq(0,1,dx), sol[,101])
```

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

```
HeatImpl <- function(deltax, deltat, tmax) {
  N = round(1/deltax)
  M = round(tmax/deltat)
  sol = matrix(0,N+1,M+1)
  rho = deltat / (deltax)^2
  B = diag(c((1+2*rho) * array(1,c(N-1,1)))) - diag(
     c(0,rho*array(1,c(N-1,1))))[2:100,1:99] - diag(
     c(0,rho*array(1,c(N-1,1))))[1:99,2:100]
  vetx = seq(0,1,deltax)
  for (i in 2:ceiling((N+1)/2)){
    sol[i,1] = 2*vetx[i]
    sol[N+2-i,1] = sol[i,1]
  }
  for (j in 1:M){
    sol[2:N,j+1] = solve(B,sol[2:N,j])
```

```
  }
  return(sol)
}

deltax=dx=0.01
deltat=dt=0.001
tmax=dt*100
sol=HeatImpl(dx,dt,tmax)

par(mfrow=c(2,2))
plot(seq(0,1,dx),sol[,1])
lines(seq(0,1,dx),sol[,1])
plot(seq(0,1,dx),sol[,11])
lines(seq(0,1,dx),sol[,11])
plot(seq(0,1,dx),sol[,51])
lines(seq(0,1,dx),sol[,51])
plot(seq(0,1,dx),sol[,101])
lines(seq(0,1,dx),sol[,101])
```

Chapter 6

Convex Optimization

R code Exa 6.1 Finite vs infinite dimensional problems

```
require(signal)
require("ucminf")

g <- function(x) {
  polyval(c(1, -10.5, 39, -59.5, 30), x)
}
xvet=seq(1,4,0.05)
plot(xvet,g(xvet))
lines(xvet,g(xvet))

x<-ucminf(c(0),g)$par
x
fval<-ucminf(c(0),g)$value
fval

x<-ucminf(c(5),g)$par
x
fval<-ucminf(c(5),g)$value
fval

f <- function(x) {
```

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

R code Exa 6.3 Linear vs non linear problems

```
require(lpSolve)
f.obj <- c(2, 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)
f.dir <- c("=", ">=", ">=", ">=", ">=")
f.rhs <- c(3, 3, 0, 0, 0)
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

```
require(pracma)
f <- function(sigma,x,y) {
  (x-1.5)^2+(y-0.5)^2+sigma/(1-x)+sigma/(1-y)
}
X<-meshgrid(x = seq(0.1,.99,.01))$X
Y<-meshgrid(x = seq(0.1,.99,.01))$Y
contour(f(0.1,X,Y))
contour(f(0.01,X,Y))
contour(f(0.001,X,Y))
contour(f(0.0001,X,Y))
```

R code Exa 6.8 Kuhn Tucker conditions

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

R code Exa 6.12 Geometric and algebraic features of linear programming

```
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)
b <- matrix(data=c(1, 3, 1), nrow=3, ncol=1, byrow=
    FALSE)
asvd = svd(A)
adiag = diag(1/asvd$d)
solution = asvd$v %*% adiag %*% t(asvd$u) %*% b
Check = A %*% solution

# This solution does not match with the book,
# but in any case it's NOT FEASIBLE either as values
      present are less than ZERO

# Book solution check
X<-matrix(data = c(0,3,-2,0,5),nrow = 5,ncol = 1,
    byrow = TRUE)
A %*% X
```

Chapter 7

Option Pricing by Binomial and Trinomial Lattices

R code Exa 7.1 Calibrating a binomial lattice

```
LatticeEurCall <- function(S0,K,r,T,sigma,N) {
  deltaT = T/N
  u=exp(sigma * sqrt(deltaT))
  d=1/u
  p=(exp(r*deltaT) - d)/(u-d)
  lattice = matrix(0,N+1,N+1)
  for(i in 0:N){
    lattice[i+1,N+1]=max(0,S0*(u^i)*(d^(N-i))-K)
  }
  for(j in (N-1):0){
    for (i in 0:j){
      lattice[i+1,j+1] = exp(-r*deltaT) * (p *
         lattice[i+2,j+2] + (1-p) * lattice[i+1,j
         +2])
    }
  }
  return(lattice[1,1])
}
call=LatticeEurCall(50,50,0.1,5/12,0.4,5)
```

```
call
call=LatticeEurCall(50,50,0.1,5/12,0.4,500)
call
```

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

```
require(OptionPricing)
LatticeEurCall <- function(S0,K,r,T,sigma,N) {
  deltaT = T/N
  u=exp(sigma * sqrt(deltaT))
  d=1/u
  p=(exp(r*deltaT) - d)/(u-d)
  lattice = matrix(0,N+1,N+1)
  for(i in 0:N){
    lattice[i+1,N+1]=max(0,S0*(u^i)*(d^(N-i))-K)
  }
  for(j in (N-1):0){
    for (i in 0:j){
      lattice[i+1,j+1] = exp(-r*deltaT) * (p *
         lattice[i+2,j+2] + (1-p) * lattice[i+1,j
         +2])
    }
  }
  return(lattice[1,1])
}
S0 = 50
K = 50
r = 0.1
sigma = 0.4
T = 5/12
N=50
BlsC = BS_EC(K = K, r = r, sigma = sigma, T = T, S0
    = S0)['price']
LatticeC = matrix(0,1,N)
for (i in 1:N){
```

```
  LatticeC[i] = LatticeEurCall(S0, K , r , T, sigma,
      i)
}
plot(1:N, matrix(1,1,N)*BlsC)
lines(1:N, matrix(1,1,N)*BlsC)
lines(1:N, LatticeC)
```

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

```
require(pracma)
L11 <- function(premium,S0,K,r,sigma,T,N) {
  deltaT = T/N
  u=exp(sigma * sqrt (deltaT))
  d=1/u
  p=(exp(r*deltaT) - d)/(u-d)
  lattice = matrix(0,N+1,N+1)
  for (i in 0:N){
    if (S0*(u^i)*(d^(N-i)) >= K){
      lattice[i+1,N+1]=S0*(u^i)*(d^(N-i)) - K -
          premium
    }
  }
  for (j in (N-1):0){
    for (i in 0:j){
      lattice[i+1, j+1] = p*lattice[i+2, j+2] + (1-p
          )*lattice[i+1, j+2]
    }
  }
  return(lattice[1,1])
}

f <- function(P) {
  L11(premium = P,S0 = 12,K = 14,r = 0.1, sigma =
     0.2, T = 10/12, N = 10)
}
```

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

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

```
require(OptionPricing)
require(tictoc)
LatticeEurCall <- function(S0,K,r,T,sigma,N) {
  deltaT = T/N
  u=exp(sigma * sqrt(deltaT))
  d=1/u
  p=(exp(r*deltaT) - d)/(u-d)
  lattice = matrix(0,N+1,N+1)
  for(i in 0:N){
    lattice[i+1,N+1]=max(0,S0*(u^i)*(d^(N-i))-K)
  }
  for(j in (N-1):0){
    for (i in 0:j){
      lattice[i+1,j+1] = exp(-r*deltaT) * (p *
         lattice[i+2,j+2] + (1-p) * lattice[i+1,j
         +2])
    }
  }
  return(lattice[1,1])
}
SmartEurLattice <- function(S0,K,r,T,sigma,N) {
  # Precompute invariant quantities
  deltaT = T/N
  u=exp(sigma * sqrt(deltaT))
  d=1/u
  p=(exp(r*deltaT) - d)/(u-d)
  discount = exp(-r*deltaT)
  p_u = discount*p
  p_d = discount*(1-p)
  # set up S values
  SVals = matrix(0,2*N+1,1)
```

```
  SVals[1] = S0*d^N
  for (i in 2:(2*N+1)){
    SVals[i] = u*SVals[i-1]
  }
  # set up terminal CALL values
  CVals = matrix(0,2*N+1,1)
  for (i in seq(1,2*N+1,2)){
    CVals[i] = max(SVals[i]-K,0)
  }
  # work backwards
  for (tau in 1:N){
    for (i in seq((tau+1),(2*N+1-tau),2)){
      CVals[i] = p_u*CVals[i+1] + p_d*CVals[i-1]
    }
  }
  return(CVals[N+1])
}
tic()
BS_EC(S0 = 50, K = 50, r = 0.1, sigma = 0.4, T = 5/
   12)['price']
toc()

tic()
LatticeEurCall(S0 = 50, K = 50, r = 0.1, sigma =
   0.4, T = 5/12,N = 2000)
toc()

tic()
SmartEurLattice(S0 = 50, K = 50, r = 0.1, sigma =
   0.4, T = 5/12,N = 2000)
toc()
```

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

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

```
  # Precompute invariant quantities
  deltaT = T/N
  u=exp(sigma * sqrt(deltaT))
  d=1/u
  p=(exp(r*deltaT) - d)/(u-d)
  discount = exp(-r*deltaT)
  p_u = discount*p
  p_d = discount*(1-p)
  # set up S values
  SVals = matrix(0,2*N+1,1)
  SVals[N+1] = S0
  for (i in 1:N){
    SVals[N+1+i] = u*SVals[N+i]
    SVals[N+1-i] = d*SVals[N+2-i]
  }
  # set up terminal values
  PVals = matrix(0,2*N+1,1)
  for (i in seq(1,2*N+1,2)){
    PVals[i] = max(K-SVals[i],0)
  }
  # work backwards
  for (tau in 1:N){
    for (i in seq((tau+1),(2*N+1-tau),2)){
      hold = p_u*PVals[i+1] + p_d*PVals[i-1]
      PVals[i] = max(hold, K-SVals[i])
    }
  }
  return(PVals[N+1])
}

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

```
AmSpreadLattice <- function(S10,S20,K,r,T,sigma1,sigma2,rho,q1,q2,N) {
  # Precompute invariant quantities
  deltaT = T/N
  nu1 = r - q1 - 0.5*sigma1^2
  nu2 = r - q2 - 0.5*sigma2^2
  u1 = exp(sigma1*sqrt(deltaT))
  d1 = 1/u1
  u2 = exp(sigma2*sqrt(deltaT))
  d2 = 1/u2
  discount = exp(-r*deltaT)
  p_uu = discount*0.25*(1 + sqrt(deltaT)*(nu1/sigma1 + nu2/sigma2) + rho)
  p_ud = discount*0.25*(1 + sqrt(deltaT)*(nu1/sigma1 - nu2/sigma2) - rho)
  p_du = discount*0.25*(1 + sqrt(deltaT)*(-nu1/sigma1 + nu2/sigma2) - rho)
  p_dd = discount*0.25*(1 + sqrt(deltaT)*(-nu1/sigma1 - nu2/sigma2) + rho)
  # set up S values
  S1vals = matrix(0,2*N+1,1)
  S2vals = matrix(0,2*N+1,1)
  S1vals[1] = S10*d1^N
  S2vals[1] = S20*d2^N
  for (i in 2:(2*N+1)){
    S1vals[i] = u1*S1vals[i-1]
    S2vals[i] = u2*S2vals[i-1]
  }
  # set up terminal values
  Cvals = matrix(0,2*N+1,2*N+1)
  for (i in seq(1,2*N+1,2)){
    for (j in seq(1,2*N+1,2)){
      Cvals[i,j] = max(S1vals[i]-S2vals[j]-K,0)
    }
  }
  # roll back
  for (tau in 1:N){
    for (i in seq((tau+1),(2*N+1-tau),2)){
```

```
      for (j in seq((tau+1),(2*N+1-tau),2)){
        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]
        Cvals[i,j] = max(hold, S1vals[i] - S2vals[j]
             - K)
      }
    }
  }
  return(Cvals[N+1,N+1])
}

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

```
require(OptionPricing)
EuCallTrinomial <- function(S0,K,r,T,sigma,N,deltaX)
    {
  # Precompute invariant quantities
  deltaT = T/N
  nu = r - 0.5*sigma^2
  discount = exp(-r*deltaT)
  p_u = discount*0.5*((sigma^2*deltaT+nu^2*deltaT^2)
     /deltaX^2 + nu*deltaT/deltaX)
  p_m = discount*(1 - (sigma^2*deltaT+nu^2*deltaT^2)
     /deltaX^2)
  p_d = discount*0.5*((sigma^2*deltaT+nu^2*deltaT^2)
     /deltaX^2 - nu*deltaT/deltaX)
  # set up S values (at maturity)
  Svals = matrix(0,2*N+1,1)
  Svals[1] = S0*exp(-N*deltaX)
  exp_dX = exp(deltaX)
```

```
  for (j in 2:(2*N+1)){
    Svals[j] = exp_dX*Svals[j-1]
  }
  # set up lattice and terminal values
  Cvals = matrix(0,2*N+1,2)
  t = mod(N,2)+1
  for (j in 1:(2*N+1)){
    Cvals[j,t] = max(Svals[j]-K,0)
  }
  for (t in (N-1):0){
    know = mod(t,2)+1
    knext = mod(t+1,2)+1
    for (j in (N-t+1):(N+t+1)){
      Cvals[j,know] = p_d*Cvals[j-1,knext]+p_m*Cvals[j,knext]+p_u*Cvals[j+1,knext]
    }
  }
  return(Cvals[N+1,1])
}
BS_EC(S0 = 100, K = 100, r = 0.06, sigma = 0.3, T = 1)['price']
EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1, sigma = 0.3, N = 3, deltaX = 0.2)
EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1, sigma = 0.3, N = 100, deltaX = 0.2)
EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1, sigma = 0.3, N = 100, deltaX = 0.5)
EuCallTrinomial(S0 = 100, K = 100, r = 0.06, T = 1, sigma = 0.3, N = 100, deltaX = 0.3*sqrt (1/100))
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

```
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
}
set.seed(37456)
paths=AssetPaths(50,0.1,0.3,1,365,3)
plot(1:length(paths[3,]) ,paths[3,],type = 'l')
lines(1:length(paths[1,]) ,paths[1,],type = 'l')
lines(1:length(paths[2,]) ,paths[2,],type = 'l')
```

R code Exa 8.2 Vectorized code to generate asset price paths

```
require(varbvs)
require(tictoc)
AssetPathsV <- function(S0,mu,sigma,T,NSteps,NRepl)
  {
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  Increments = nudt + sidt*randn(NRepl, NSteps)
  LogPaths = apply(cbind(matrix(log(S0)*matrix(1,
     NRepl,1)),Increments),2,cumsum)
  SPaths = exp(LogPaths)
  SPaths[,1] = S0
  return(SPaths)
}
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
}
Paths = AssetPathsV(50,0.1,0.3,1,100,1000)

N = dim(Paths)[2]
for (i in 1:N){
```

```
  plot(Paths[,i],type = 'l')
}

tic()
AssetPaths(50,0.1,0.3,1,100,1000)
toc()

tic()
AssetPathsV(50,0.1,0.3,1,100,1000)
toc()
```

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

```
require(pracma)
require(tictoc)
StopLoss <- function(S0,K,mu,sigma,r,T,Paths) {
  NRepl = dim(Paths)[1]
  NSteps = dim(Paths)[2]
  NSteps = NSteps - 1
  # true number of steps
  Cost = matrix(0,NRepl,1)
  dt = T/NSteps
  DiscountFactors = exp(-r*(seq(0,NSteps,1)*dt))
  for (k in 1:NRepl){
    CashFlows = matrix(0,NSteps+1,1)
    if (Paths[k,1] >= K){
      Covered = 1
      CashFlows[1] = -Paths[k,1]
    } else {
      Covered = 0
    }
    for (t in 2:(NSteps+1)){
      if ((Covered == 1) & (Paths[k,t] < K)){
        # Sell
        Covered = 0
```

```
        CashFlows[t] = Paths[k,t]
      } else if ((Covered == 0) & (Paths[k,t] > K)){
        # Buy
        Covered = 1
        CashFlows[t] = -Paths[k,t]
      }
    }
    if (Paths[k,NSteps + 1] >= K){
      # Option is exercised
      CashFlows[NSteps + 1] = CashFlows[NSteps + 1]
         + K
      }
    Cost[k] = - dot(DiscountFactors,CashFlows)
  }
  return(mean(Cost))
}
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
}
S0 = 50
K = 50
mu = 0.1
sigma = 0.4
r = 0.05
T = 5/12
NRepl =100000
NSteps = 10
```

```
set.seed(39473)
Paths=AssetPaths(S0,mu, sigma,T,NSteps,NRepl)
tic()
StopLoss(S0,K,mu,sigma,r,T,Paths)
toc()
```

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

```
require(pracma)
require(tictoc)
StopLoss <- function(S0,K,mu,sigma,r,T,Paths) {
  NRepl = dim(Paths)[1]
  NSteps = dim(Paths)[2]
  NSteps = NSteps - 1
  # true number of steps
  Cost = matrix(0,NRepl,1)
  dt = T/NSteps
  DiscountFactors = exp(-r*(seq(0,NSteps,1)*dt))
  for (k in 1:NRepl){
    CashFlows = matrix(0,NSteps+1,1)
    if (Paths[k,1] >= K){
      Covered = 1
      CashFlows[1] = -Paths[k,1]
    } else {
      Covered = 0
    }
    for (t in 2:(NSteps+1)){
      if ((Covered == 1) & (Paths[k,t] < K)){
        # Sell
        Covered = 0
        CashFlows[t] = Paths[k,t]
      } else if ((Covered == 0) & (Paths[k,t] > K)){
        # Buy
        Covered = 1
        CashFlows[t] = -Paths[k,t]
```

```
      }
    }
    if (Paths[k,NSteps + 1] >= K){
      # Option is exercised
      CashFlows[NSteps + 1] = CashFlows[NSteps + 1] + K
      }
    Cost[k] = - dot(DiscountFactors,CashFlows)
  }
  return(mean(Cost))
}
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*rnorm(1))
    }
  }
  return(SPaths)
}
S0 = 50
K = 50
mu = 0.1
sigma = 0.4
r = 0.05
T = 5/12
NRepl =100000
NSteps = 10
set.seed(39473)
Paths=AssetPaths(S0,mu, sigma,T,NSteps,NRepl)

StopLossV <- function(S0,K,mu,sigma,r,T,Paths) {
  NRepl = dim(Paths)[1]
```

```
  NSteps = dim(Paths)[2]
  NSteps = NSteps - 1
  Cost = matrix(0,NRepl,1)
  CashFlows = matrix(0,NRepl,NSteps+1)
  dt = T/NSteps
  DiscountFactors = exp(-r*(seq(0,NSteps,1))*dt)
  OldPrice = cbind(matrix(0,NRepl,1), Paths[,1:
     NSteps])
  UpTimes = which(OldPrice < K & Paths >= K)
  DownTimes = which(OldPrice >= K & Paths < K)
  CashFlows[UpTimes] = -Paths[UpTimes]
  CashFlows[DownTimes] = Paths[DownTimes]
  ExPaths = which(Paths[,NSteps+1] >= K)
  CashFlows[ExPaths,NSteps+1] = CashFlows[ExPaths,
     NSteps+1] + K
  Cost = -CashFlows %*%(matrix(DiscountFactors))
  return(mean(Cost))
}

tic()
StopLossV(S0,K,mu,sigma,r,T,Paths)
toc()
```

R code Exa 8.5 Evaluating the performance of delta hedging

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

```
  CashFlows = matrix(0,1,NSteps+1)
  dt = T/NSteps
  DiscountFactors = exp(-r*(seq(0,NSteps,1)*dt))
  for (i in 1:NRepl){
    Path = Paths[i,]
    Position = 0
    Deltas = matrix()
    for (k in 1:NSteps){
      Deltas[k] = f(T = T-(k-1)*dt, Path = Path[k])
    }
    for (j in 1:NSteps){
      CashFlows[j] = (Position - Deltas[j])*Path[j]
      Position = Deltas[j]
    }
    if (Path[NSteps+1] > K){
      CashFlows[NSteps+1] = K - (1-Position)*Path[
         NSteps+1]
    } else {
      CashFlows[NSteps+1] = Position*Path[NSteps+1]
    }
    Cost[i] = -CashFlows %*% DiscountFactors
  }
  return(mean(Cost))
}

AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
```

```
}

StopLossV <- function(S0,K,mu,sigma,r,T,Paths) {
  NRepl = dim(Paths)[1]
  NSteps = dim(Paths)[2]
  NSteps = NSteps - 1
  Cost = matrix(0,NRepl,1)
  CashFlows = matrix(0,NRepl,NSteps+1)
  dt = T/NSteps
  DiscountFactors = exp(-r*(seq(0,NSteps,1))*dt)
  OldPrice = cbind(matrix(0,NRepl,1), Paths[,1:
     NSteps])
  UpTimes = which(OldPrice < K & Paths >= K)
  DownTimes = which(OldPrice >= K & Paths < K)
  CashFlows[UpTimes] = -Paths[UpTimes]
  CashFlows[DownTimes] = Paths[DownTimes]
  ExPaths = which(Paths[,NSteps+1] >= K)
  CashFlows[ExPaths,NSteps+1] = CashFlows[ExPaths,
     NSteps+1] + K
  Cost = -CashFlows %*%(matrix(DiscountFactors))
  return(mean(Cost))
}

S0 = 50
K = 52
mu = 0.1
sigma = 0.4
r = 0.05
T = 5/12
NRepl =10000
NSteps = 10
C = BS_EC(K = K, r = r, sigma = sigma, T = T, S0 =
   S0)['price']
set.seed(3872)
Paths=AssetPaths(S0,mu,sigma,T,NSteps,NRepl)
True_Price = BS_EC(S0 = S0,K = K,r = r,sigma = sigma
   ,T = T)['price']
SL = StopLossV(S0,K,mu,sigma,r,T,Paths)
```

```
DC = DeltaHedging(S0,K,mu,sigma,r,T,Paths)

#NSteps = 100
#set.seed(38232)
#Paths=AssetPaths(S0,mu,sigma,T,NSteps,NRepl)
#True_Price = BS_EC(S0 = S0,K = K,r = r,sigma =
    sigma,T = T)['price']
#SL = StopLossV(S0,K,mu,sigma,r,T,Paths)
#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

```
WienerBridge <- function(T, NSteps) {
  NBisections = log2(NSteps)
  if (round(NBisections) != NBisections){
    cat('ERROR in WienerBridge: NSteps must be a
        power of 2','\n')
    return
  }
  WSamples = matrix(0,NSteps+1,1)
  WSamples[1] = 0
  WSamples[NSteps+1] = sqrt(T)*rnorm(1)
  TJump = T
  IJump = NSteps
  for (k in 1:NBisections){
    left = 1
    i = IJump/2 + 1
    right = IJump + 1
    for (j in 1:(2^(k-1))){
      a = 0.5*(WSamples[left] + WSamples[right])
      b = 0.5*sqrt(TJump)
      WSamples[i] = a + b*rnorm(1)
      right = right + IJump
      left = left + IJump
```

```
      i = i + IJump
    }
    IJump = IJump/2
    TJump = TJump/2
  }
  return(WSamples)
}


# CheckBridge.m
set.seed(3826)
NRepl = 100000
T = 1
NSteps = 4
WSamples = matrix(0,NRepl, 1+NSteps)
for (i in 1:NRepl){
  WSamples[i,] =WienerBridge(T, NSteps)
}
a <- function(X) {
  mean(X)
}
b <- function(X) {
  sqrt(var(X))
}
m = matrix()
sdev = matrix()
for (z in 2:(1+NSteps)){
  m[z-1] = a(WSamples[,z])
  sdev[z-1] = b(WSamples[,z])
}
m
sdev
sqrt((1:NSteps)*T/NSteps)
```

R code Exa 8.7 Code to price an exchange option analytically

```
require(lmom)
Exchange <- function(V0,U0,sigmaV,sigmaU,rho,T,r) {
  sigmahat = sqrt(sigmaU^2 + sigmaV^2 - 2*rho*sigmaU
    *sigmaV)
  d1 = (log(V0/U0) + 0.5*T*sigmahat^2)/(sigmahat*
    sqrt(T))
  d2 = d1 - sigmahat*sqrt(T)
  p = V0*cdfnor(d1) - U0*cdfnor(d2)
  return(p)
}

V0 = 50
U0 = 60
sigmaV = 0.3
sigmaU = 0.4
rho = 0.7
T = 5/12
r = 0.05
Exchange(V0 ,U0, sigmaV, sigmaU, rho, T ,r)
```

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

```
require(fBasics)
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
f <- function(r,T,VT,UT) {
  exp(-r*T)*max(VT-UT, 0)
}
ExchangeMC <- function(V0,U0,sigmaV,sigmaU,rho,T,r,
```

```
    NRepl) {
  eps1 = rnorm(NRepl)
  eps2 = rho*eps1 + sqrt(1-rho^2)*rnorm(NRepl)
  VT = V0*exp((r - 0.5*sigmaV^2)*T + sigmaV*sqrt(T)*
     eps1)
  UT = U0*exp((r - 0.5*sigmaU^2)*T + sigmaU*sqrt(T)*
     eps2)
  DiscPayoff = matrix()
  for(i in 1:length(VT)){
    DiscPayoff[i] = f(r,T,VT[i],UT[i])
  }
  parameter_estimation<-.normFit(DiscPayoff)
  ci<-norm.interval(DiscPayoff)
  return(c(parameter_estimation,ci))
}
V0 = 50
U0 = 60
sigmaV = 0.3
sigmaU = 0.4
rho = 0.7
T = 5/12
r = 0.05
NRepl = 200000
ExchangeMC(V0,U0,sigmaV,sigmaU,rho,T,r,NRepl)
```

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

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

```
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
}
norm.interval = function(data, variance = var(data),
     conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
DOPutMC<-function(S0,K,r,T,sigma,Sb,NSteps,NRepl){
  # Generate asset paths
  Call = BS_EC(S0,K,r,T,sigma)
  Put = BS_EP(S0,K,r,T,sigma)
  Payoff = matrix(0,NRepl,1)
  NCrossed = 0
  for (i in 1:NRepl){
    Path=AssetPaths(S0,r,sigma,T,NSteps,1)
    crossed = any(Path <= Sb)
    if (crossed == 0){
      Payoff[i] = max(0, K - Path[NSteps+1])
    } else{
      Payoff[i] = 0
      NCrossed = NCrossed + 1
    }
  }
  parameter_estimation<-.normFit(exp(-r*T) * Payoff)
  ci<-norm.interval(exp(-r*T) * Payoff)
  return(c(parameter_estimation,ci))
}
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

```
require(OptionPricing)
require(fBasics)
require(fOptions)
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
}

norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}

is.integer0 <- function(x)
{
  is.integer(x) && length(x) == 0L
}

```

```
DOPutMCCond <- function(S0,K,r,T,sigma,Sb,NSteps,
    NRepl) {
  dt = T/NSteps;
  Call = GBSOption(TypeFlag = "c", S = S0, X = K,
      Time = T, r = r, sigma = sigma, b =r)
  Put = GBSOption(TypeFlag = "p", S = S0, X = K,
      Time = T, r = r, sigma = sigma, b =r)
  # Generate asset paths and payoffs for the down
      and in option
  NCrossed = 0
  Payoff = matrix(0,NRepl,1)
  Times = matrix(0,NRepl,1)
  StockVals = matrix(0,NRepl,1)
  for (i in 1:NRepl){
    Path=AssetPaths(S0,r,sigma,T,NSteps,1)
    tcrossed = pmin(which(Path <= Sb))
    if (!(is.integer0(tcrossed))){
      NCrossed = NCrossed + 1
      Times[NCrossed,] = (length(tcrossed) - 1) * dt
      StockVals[NCrossed,] = Path[,length(tcrossed)]
    }
  }
  Paux<-matrix()
  f <- function(S0,K,r,T,sigma) {
    GBSOption(TypeFlag = "p", S = S0, X = K, Time =
        T, r = r, sigma = sigma, b =r)
  }
  if (NCrossed > 0){
    for (j in 1:NCrossed){
      Paux[j] = f(StockVals[j],K,r,(T-Times[j]),
          sigma)@price
      Payoff[j] = exp(-r*Times[j]) * Paux[j]
    }
  }
  parameter_estimation<-.normFit(Put@price - Payoff)
  ci<-norm.interval(Put@price - Payoff)
  return(c(parameter_estimation,ci,NCrossed))
}
```

```
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

```
require(OptionPricing)
require(fBasics)
require(fOptions)
require(varbvs)
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
          rnorm(1))
    }
  }
  return(SPaths)
}

norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}

is.integer0 <- function(x)
{
```

```
  is.integer(x) && length(x) == 0L
}

DOPutMCCondIS <- function(S0,K,r,T,sigma,Sb,NSteps,NRepl,bp){
  dt = T/NSteps
  nudt = (r-0.5*sigma^2)*dt
  b = bp*nudt
  sidt = sigma*sqrt(dt)
  Call = GBSOption(TypeFlag = "c", S = S0, X = K, Time = T, r = r, sigma = sigma, b =r)
  Put = GBSOption(TypeFlag = "p", S = S0, X = K, Time = T, r = r, sigma = sigma, b =r)
  # Generate asset paths and payoffs for the down and in option
  NCrossed = 0
  Payoff = matrix(0,NRepl,1)
  Times = matrix(0,NRepl,1)
  StockVals = matrix(0,NRepl,1)
  ISRatio = matrix(0,NRepl,1)
  for (i in 1:NRepl){
    # generate normals
    vetZ = nudt - b + sidt*randn(1,NSteps)
    LogPath = apply(cbind(log(S0), vetZ),1,cumsum)
    Path = exp(LogPath)
    jcrossed = pmin(which(Path <= Sb ))
    if (!(is.integer0(jcrossed))){
      jcrossed = min(jcrossed)
      NCrossed = NCrossed + 1
      TBreach = jcrossed - 1
      Times[NCrossed,] = TBreach * dt
    StockVals[NCrossed,] = Path[jcrossed]
      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))
    }
  }
  Paux<-matrix()
```

```
  f <- function(S0,K,r,T,sigma) {
    GBSOption(TypeFlag = "p", S = S0, X = K, Time =
       T, r = r, sigma = sigma, b =r)
  }
  if (NCrossed > 0){
    for (j in 1:NCrossed){
      Paux[j] = f(StockVals[j],K,r,(T-Times[j]),
         sigma)@price
      Payoff[j] = exp(-r*Times[j])* Paux[j] *
         ISRatio[j]
    }
  }
  parameter_estimation<-.normFit(Put@price - Payoff)
  ci<-norm.interval(Put@price - Payoff)
  return(c(parameter_estimation,ci,NCrossed))
}
DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,0)
DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,20)
DOPutMCCondIS(50,52,0.1,2/12,0.4,30,60,10000,50)
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

```
require(OptionPricing)
require(fBasics)
require(fOptions)
require(varbvs)
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
```

```
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
          rnorm(1))
    }
  }
  return(SPaths)
}
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
AsianMC<-function(S0,K,r,T,sigma,NSamples,NRepl){
  Payoff = matrix(0,NRepl,1)
  for (i in 1:NRepl){
    Path=AssetPaths(S0,r,sigma,T,NSamples,1)
    Payoff[i] = max(0, mean(Path[2:(NSamples+1)]) -
        K)
  }
  parameter_estimation<-.normFit(exp(-r*T) * Payoff)
  ci<-norm.interval(exp(-r*T) * Payoff)
  return(c(parameter_estimation,ci))
}
set.seed(28282)
X<-AsianMC(50,50,0.1,5/12,0.4,5,50000)
X
X[[3]]-X[[2]]
```

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

```
require(OptionPricing)
require(fBasics)
require(fOptions)
```

```r
require(varbvs)
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*
         rnorm(1))
    }
  }
  return(SPaths)
}
norm.interval = function(data, variance = var(data),
     conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
AsianMCCV<-function(S0,K,r,T,sigma,NSamples,NRepl,
    NPilot){
  # pilot replications to set control parameter
  TryPath=AssetPaths(S0,r,sigma,T,NSamples,NPilot)
  StockSum<-matrix()
  PP<-matrix()
  TryPayoff<-matrix()
  for (i in 1:length(TryPath[,1])){
    StockSum[i] = sum(TryPath[i,])
    PP[i] = mean(TryPath[i,2:(NSamples+1)])
    TryPayoff[i] = exp(-r*T) * max(0, PP[i] - K)
  }
  MatCov = cov(cbind(StockSum, TryPayoff))
  c = - MatCov[1,2] / var(StockSum)
  dt = T / NSamples
  ExpSum = S0 * (1 - exp((NSamples + 1)*r*dt)) / (1
```

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

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

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

```
    sqrt(b)))
  return(P)
}
norm.interval = function(data, variance = var(data), conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
AssetPaths <- function(S0,mu,sigma,T,NSteps,NRepl) {
  SPaths = matrix(0,NRepl, 1+NSteps)
  SPaths[,1] = S0
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  for (i in 1:NRepl){
    for (j in 1:NSteps){
      SPaths[i,j+1]=SPaths[i,j]*exp(nudt + sidt*rnorm(1))
    }
  }
  return(SPaths)
}
AsianMCGeoCV<-function(S0,K,r,T,sigma,NSamples,NRepl,NPilot){
  # precompute quantities
  DF = exp(-r*T)
  GeoExact = GeometricAsian(S0,K,r,T,sigma,0,NSamples)
  # pilot replications to set control parameter
  GeoPrices = matrix(0,NPilot,1)
  AriPrices = matrix(0,NPilot,1)
  for (i in 1:NPilot){
    Path=AssetPaths(S0,r,sigma,T,NSamples,1)
    GeoPrices[i]=DF*max(0,(prod(Path[,2:(NSamples+1)]))^(1/NSamples) - K)
    AriPrices[i]=DF*max(0,mean(Path[,2:(NSamples+1)
```

```
        ]) - K)
  }
  MatCov = cov(cbind(GeoPrices, AriPrices))
  c = - MatCov[1,2] / var(GeoPrices)
  # MC run
  ControlVars = matrix(0,NRepl,1)
  for (i in 1:NRepl){
   Path = AssetPaths(S0,r,sigma,T,NSamples,1)
   GeoPrice = DF*max(0, (prod(Path[2:(NSamples+1)]))
       ^(1/NSamples) - K)
   AriPrice = DF*max(0, mean(Path[2:(NSamples+1)]) -
        K)
   ControlVars[i] = AriPrice + c * (GeoPrice -
       GeoExact)
  }
  parameter_estimation<-.normFit(ControlVars)
  ci<-norm.interval(ControlVars)
  return(c(parameter_estimation,ci))
}
set.seed(2372)
S0 = 50
K = 55
r = 0.05
sigma = 0.4
T = 1
NSamples = 12
NRepl = 9000
NPilot = 1000
AsianMCGeoCV(S0,K,r,T,sigma,NSamples,NRepl,NPilot)
```

R code Exa 8.15 Pricing an Asian option by Halton sequences

```
require(gmp)
require(lmom)
require(lestat)
```

```
GetHalton <- function(HowMany, Base) {
  Seq = matrix(0,HowMany,1)
  NumBits = 1+round(log(HowMany)/log(Base));
  VetBase = Base^(-(1:NumBits));
  WorkVet = matrix(0,1,NumBits);
  for (i in 1:HowMany){
    j = 1;
    ok = 0;
    while (ok == 0){
      WorkVet[j] = WorkVet[j]+1;
      if (WorkVet[j] < Base){
        ok = 1;
      }
      else{
        WorkVet[j] = 0;
        j = j+1;
      }
    }
    Seq[i] = sum(WorkVet * VetBase)
  }
  return(Seq)
}

myprimes<-function(N){
  found = 0
  trynumber = 2
  p <- matrix()
  while (found < N){
    if (isprime(trynumber)){
      p <-c(p , trynumber)
      found = found + 1
    }
    trynumber = trynumber + 1
  }
  return(p)
}
```

```
HaltonPaths<-function(S0,mu,sigma,T,NSteps,NRepl){
  dt = T/NSteps
  nudt = (mu-0.5*sigma^2)*dt
  sidt = sigma*sqrt(dt)
  # Use inverse transform to generate standard
     normals
  NormMat = matrix(0,NRepl, NSteps)
  Bases = myprimes(NSteps)
  RandMat<-matrix(0,NRepl,NSteps)
  H <- matrix()
  for (i in 2:(NSteps+1)){
    H = GetHalton(NRepl,Bases[i])
    for (j in 1:length(H)){
      RandMat[j,i-1] = invcdf(normal(),H[j])
    }
  }
  Increments = nudt + sidt*RandMat
  LogPaths = apply(cbind(log(S0)*matrix(1,NRepl,1),
     Increments),1,cumsum)
  LogPaths = t(LogPaths)
  SPaths = exp(LogPaths)
  SPaths[,1] = S0
  return(SPaths)
}

AsianHalton<-function(S0,K,r,T,sigma,NSamples,NRepl)
   {
  Payoff = matrix(0,NRepl,1)
  Path=HaltonPaths(S0,r,sigma,T,NSamples,NRepl)
  Payoff<-matrix(0,NSamples,1)
  for(k in 1:length(Path[,1])){
    Payoff[k] = max(0, mean(Path[k,2:(NSamples+1)])
       - K)
  }
  P = mean( exp(-r*T) * matrix(Payoff))
  return(P)
}

```

```
set.seed(3226)
AsianHalton(50,50,0.1,5/12,0.4,5,1000)
AsianHalton(50,50,0.1,5/12,0.4,5,3000)
AsianHalton(50,50,0.1,5/12,0.4,5,10000)
AsianHalton(50,50,0.1,5/12,0.4,5,50000)

AsianHalton(50,50,0.1,2,0.4,24,1000)
AsianHalton(50,50,0.1,2,0.4,24,5000)
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

```
require(gmp)
require(lmom)
require(lestat)
require(matrixStats)

myprimes<-function(N){
  found = 0
  trynumber = 2
  p <- matrix()
  while (found < N){
    if (isprime(trynumber)){
      p <-c(p , trynumber)
      found = found + 1
    }
    trynumber = trynumber + 1
  }
  return(p)
}

GetHalton <- function(HowMany, Base) {
  Seq = matrix(0,HowMany,1)
  NumBits = 1+round(log(HowMany)/log(Base));
```

```
  VetBase = Base^(-(1:NumBits));
  WorkVet = matrix(0,1,NumBits);
  for (i in 1:HowMany){
    j = 1;
    ok = 0;
    while (ok == 0){
      WorkVet[j] = WorkVet[j]+1;
      if (WorkVet[j] < Base){
        ok = 1;
      }
      else{
        WorkVet[j] = 0;
        j = j+1;
      }
    }
    Seq[i] = sum(WorkVet * VetBase)
  }
  return(Seq)
}

WienerHaltonBridge<-function(T, NSteps, NRepl, Limit
   ){
  NBisections = log2(NSteps)
  if (round(NBisections) != NBisections){
    cat('ERROR in WienerHB: NSteps must be a power
       of 2','\n')
    return
  }
  # Generate standard normal samples
  NormMat = matrix(0,NRepl, NSteps)
  Bases = myprimes(NSteps)
  for (i in 2:(NSteps+1)){
    H = GetHalton(NRepl,Bases[i])
    for (j in 1:length(H)){
      NormMat[j,i-1] = invcdf(normal(),H[j])
    }
  }
  # Initialize extreme points of paths
```

```
  WSamples = matrix(0,NRepl,NSteps+1)
  WSamples[,1] = 0
  WSamples[,NSteps+1] = sqrt(T)*NormMat[,1]
  # Fill paths
  HUse = 2
  TJump = T
  IJump = NSteps
  for (k in 1:NBisections){
    left = 1
    i = IJump/2 + 1
    right = IJump + 1
    for (l in 1:(2^(k-1))){
      a = 0.5*(WSamples[,left] + WSamples[,right])
      b = 0.5*sqrt(TJump)
      if (HUse <= Limit){
        WSamples[,i] = a + b*NormMat[,HUse]
      } else {
        WSamples[,i] = a + b*rnorm(NRepl)
        }
        right = right + IJump
        left = left + IJump
        i = i + IJump
    }
    IJump = IJump/2
    TJump = TJump/2
    HUse = HUse + 1
    }
  return(WSamples)
}

GBMHaltonBridge<-function(S0,mu,sigma,T,NSteps,NRepl,Limit){
  if (round(log2(NSteps)) != log2(NSteps)){
    cat('ERROR in GBMBridge: NSteps must be a power of 2','\n')
    return
  }
  dt = T/NSteps
```

```
  nudt = (mu-0.5*sigma^2)*dt
  W = WienerHaltonBridge(T,NSteps,NRepl,Limit)
  Increments = nudt + sigma*t(diff(t(W)))
  LogPaths = apply(cbind(log(S0)*matrix(1,NRepl,1),
     Increments),1,cumsum)
  LogPaths = t(LogPaths)
  Paths = exp(LogPaths)
  Paths[,1] = S0
  return(Paths)
}

set.seed(271782)
NRepl = 10000
T = 5
NSteps = 16
Limit = NSteps
S0 = 50
mu = 0.1
sigma = 0.4
Paths = GBMHaltonBridge(S0, mu, sigma, T, NSteps,
   NRepl, Limit)
r = mu
NSamples = NSteps
K = 55
Payoff<-matrix()
for(p in 1:length(Paths[,1])){
  Payoff[p] = max(0, mean(Paths[p,2:(NSamples+1)]) -
      K)
}
P = mean( exp(-r*T) * matrix(Payoff))
```

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

```
require(varbvs)
```

```r
require(fBasics)
norm.interval = function(data, variance = var(data), conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}
BlsDeltaMCNaive<-function(S0,K,r,T,sigma,dS,NRepl){
  nuT = (r - 0.5*sigma^2)*T
  siT = sigma * sqrt(T)
  Payoff1<-matrix()
  Payoff2<-matrix()
  for (i in 1:NRepl){
    Payoff1[i] = max(0, S0*exp(nuT+siT*randn(1,1))-K)
    Payoff2[i] = max(0, (S0+dS)*exp(nuT+siT*randn(1,1))-K)
  }
  SampleDiff = exp(-r*T)*(Payoff2 - Payoff1)/dS
  parameter_estimation<-.normFit(SampleDiff)
  ci<-norm.interval(SampleDiff)
  return(c(parameter_estimation,ci))
}
set.seed(762567)
S0=50
K=52
r=0.05
T=5/12
sigma = 0.4
NRepl=50000
dS = 0.5
BlsDeltaMCNaive(S0,K,r,T,sigma,dS,NRepl)
```

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

```
require(varbvs)
require(fBasics)
norm.interval = function(data, variance = var(data),
    conf.level = 0.95) {
  z = qnorm((1 - conf.level)/2, lower.tail = FALSE)
  xbar = mean(data)
  sdx = sqrt(variance/length(data))
  c(xbar - z * sdx, xbar + z * sdx)
}

BlsDeltaMCPath<-function(S0,K,r,T,sigma,NRepl){
  nuT = (r - 0.5*sigma^2)*T
  siT = sigma * sqrt(T)
  VLogn<-matrix()
  for (i in 1:NRepl){
    VLogn[i] = exp(nuT+siT*randn(1,1))
  }
  SampleDelta = exp(-r*T) * VLogn * (S0*VLogn > K)
  parameter_estimation<-.normFit(SampleDelta)
  ci<-norm.interval(SampleDelta)
  return(c(parameter_estimation,ci))
}

set.seed(3725678)
S0=50
K=52
r=0.05
T=5/12
sigma = 0.4
NRepl=50000
BlsDeltaMCPath(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

```
require(pracma)

EuPutExpl<-function(S0,K,r,T,sigma,Smax,dS,dt){
  # set up grid and adjust increments if necessary
  M = round(Smax/dS)
  dS = Smax/M
  N = round(T/dt)
  dt = T/N
  matval = matrix(0,M+1,N+1)
  vetS = seq(0,Smax,length=M+1)
  veti = 0:M
  vetj = 0:N
  # set up boundary conditions
  for (k in 1:(M+1)){
    matval[k,N+1] = max(K-vetS[k],0)
  }
  matval[1,] = K*exp(-r*dt*(N-vetj))
  matval[M+1,] = 0
```

```
  # set up coefficients
  a = 0.5*dt*(sigma^2*veti - r)*veti
  b = 1- dt*(sigma^2*veti^2 + r)
  c = 0.5*dt*(sigma^2*veti + r)*veti
  # solve backward in time
  for (j in N:1){
    for (i in 2:M){
      matval[i,j] = a[i]*matval[i-1,j+1] + b[i]*
         matval[i,j+1]+ c[i]*matval[i+1,j+1]
    }
  }
  # return price, possibly by linear interpolation
     outside the grid
  price = interp1(vetS, matval[,1], S0)
  return(price)
}
EuPutExpl(50,50,0.1,5/12,0.4,100,2,5/1200)
EuPutExpl(50,50,0.1,5/12,0.3,100,2,5/1200)
EuPutExpl(50,50,0.1,5/12,0.3,100,1.5,5/1200)
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

```
require(pracma)

EuPutImpl<-function(S0,K,r,T,sigma,Smax,dS,dt){
  # set up grid and adjust increments if necessary
  M = round(Smax/dS)
  dS = Smax/M
  N = round(T/dt)
  dt = T/N
  matval = matrix(0,M+1,N+1)
  vetS = seq(0,Smax,length=M+1)
  veti = 0:M
  vetj = 0:N
```

```
  # set up boundary conditions
  for (k in 1:(M+1)){
    matval[k,N+1] = max(K-vetS[k],0)
  }
  matval[1,] = K*exp(-r*dt*(N-vetj))
  matval[M+1,] = 0
  # set up the tridiagonal coefficients matrix
  a = 0.5*(r*dt*veti-sigma^2*dt*(veti^2))
  b = 1+sigma^2*dt*(veti^2)+r*dt
  c = -0.5*(r*dt*veti+sigma^2*dt*(veti^2))
  zero<-matrix(0,1,M-1)
  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)]
  L = lu(coeff)$L
  U = lu(coeff)$U
  # solve the sequence of linear systems
  aux = matrix(0,M-1,1)
  for (j in N:1){
    aux[1] = - a[2] * matval[1,j]
    # other term from BC is zero
    matval[2:M,j] = solve(U,(solve(L,(matval[2:M,j+1] + aux))))
  }
  # return price, possibly by linear interpolation outside the grid
  price = interp1(vetS, matval[,1], S0)
  return(price)
}
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

```
require(pracma)
```

```r

DOPutCK<-function(S0,K,r,T,sigma,Sb,Smax,dS,dt){
  # set up grid and adjust increments if necessary
  M = round((Smax-Sb)/dS)
  dS = (Smax-Sb)/M
  N = round(T/dt)
  dt = T/N
  matval = matrix(0,M+1,N+1)
  vetS = seq(Sb,Smax,length = M+1)
  veti = vetS / dS
  vetj = 0:N
  # set up boundary conditions
  for (k in 1:(M+1)){
    matval[k,N+1] = max(K-vetS[k],0)
  }
  matval[1,] = 0
  matval[M+1,] = 0
  # set up the coefficients matrix
  alpha = 0.25*dt*( sigma^2*(veti^2) - r*veti)
  beta = -dt*0.5*( sigma^2*(veti^2) + r )
  gamma = 0.25*dt*( sigma^2*(veti^2) + r*veti )
  zero<-matrix(0,1,M-1)
  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)]
  L = lu(M1)$L
  U = lu(M1)$U
  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)]
  # solve the sequence of linear systems
  for (j in N:1){
    matval[2:M,j] = solve(U,solve(L,(M2%*%matval[2:M,j+1])))
  }
  # return price, possibly by linear interpolation outside the grid
  price = interp1(vetS, matval[,1], S0)
```

```
  return(price)
}
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

```
require(varbvs)
require(pracma)
require(zeallot)
OptFolioMC<-function(W0,S0,mu,sigma,r,T,NScen,utilf){
  muT = (mu - 0.5*sigma^2)*T
  sigmaT = sigma*sqrt(T)
  R = exp(r*T)
  NormSamples = muT + sigmaT*randn(NScen,1)
  ExcessRets = exp(NormSamples) - R
  MExpectedUtility<-function(x){
    -mean(utilf(W0*((x*ExcessRets) + R)))
  }
  share = fminbnd(MExpectedUtility, 0, 1)
  return(share)
}
set.seed(294)
share = OptFolioMC(1000,50,0.1,0.4,0.05,1,10000,log)
share$xmin

```

```
set.seed(2947)
share = OptFolioMC(1000,50,0.1,0.4,0.05,1,5000000,
   log)
share$xmin

OptFolioGauss<-function(W0,S0,mu,sigma,r,T,NScen,
   utilf){
  muT = (mu - 0.5*sigma^2)*T
  sigmaT = sigma*sqrt(T)
  R = exp(r*T)
  print(GaussHermite(muT,sigmaT^2,NScen))
  c(x,w) %<-% GaussHermite(muT,sigmaT^2,NScen)
  ExcessRets = exp(x) - R
  MExpectedUtility<-function(x){
    - Re(dot(Re(w),  utilf(W0*((Re(x)*ExcessRets) +
       R))))
  }
  share = fminbnd(MExpectedUtility, 0, 1)
}


GaussHermite<-function(mu,sigma2,N){
  HPoly1 = c(1/pi^0.25)
  HPoly2 = c(sqrt(2) / pi^0.25, 0)
  for (j in 1:(N-1)){
    HPoly3 = c(sqrt(2/(j+1)) * HPoly2 , 0) - c(0, 0,
        sqrt(j/(j+1))*HPoly1)
    HPoly1 = HPoly2
    HPoly2 = HPoly3
  }
  x1 = polyroot(HPoly3)
  w1 = matrix(0,N,1)
  for (i in 1:N){
    w1[i] = 1/(N)/(polyval(HPoly1, x1[i]))^2
  }
  x = sort(x1*sqrt(2*sigma2)+mu)
  index= order(x1*sqrt(2*sigma2)+mu)
```

```
  w = w1[index]/sqrt(pi)
  return(list(x,w))
}

share = OptFolioGauss(1000,50 ,0.1, 0.4, 0.05,1,2,
   log)
share

#different answers as R polyroot function
   calculating different roots than matlab.
```