0910_Introduction to Monte-Carlo Sim

```
##Simulating coin-tossing experiments
x=sample(c("H","T"),10,replace=TRUE)
table(x)
table(x)/10
#
x=sample(c("H","T"),100,replace=TRUE)
table(x)/100
#
x=sample(c("H","T"),10000,replace=TRUE)
table(x)/10000

#replace 是否重置

##Simulating a game of chance
win=sample(c(-1,1),size=50,replace=T)
#cumsum 累積加總 可看出財富變化，可看出領先的時候
(cum.win>0)
cum.win=cumsum(win)
cum.win
sum(win)
#
win=sample(c(-1,1),
          size=50,replace=T,
          prob=c(0.6,0.4))

#看總共贏幾塊
KoP=function(n=50){
   win=sample(c(-1,1),size=n,replace=T)
   sum(win)
}
#sample取樣,replicate重複

#算損益兩平的機率
F=replicate(1000,KoP())
table(F)
plot(table(F))
sum(F==0)/1000

# F總財富，L領先幾次，M最多贏幾次
KoP=function(n=50){
  win=sample(c(-1,1),size=n,replace=T)
  cum.win=cumsum(win)
  c(F=sum(win),
   L=sum(cum.win>0),
   M=max(cum.win))
}
KoP()
S=replicate(1000,KoP())
#dim矩陣
dim(S)
#
S[,1:10]
mean(S["L",])
mean(S["M",])
sum(S["M",]>10)/1000



##IC Module Analysis
preg=0.7 #0.7有電
pbad.reg=0.1 #之中0.1壞了
pgood.reg=1-pbad.reg
#
pirreg=1-preg #0.3沒電
pbad.irreg=0.4 #之中0.4壞了
pgood.irreg=1-pbad.irreg

ICmodule.sim=function(n=10){
  #創空的位置
   simulated.modules=rep(NA,n)
   #1 denotes regular
   #-1 denotes irregular
   labels=sample(c(1,-1),n,
                 prob=c(preg,pirreg),
                 replace=TRUE)
   #電好
   if(any(labels==1)){
      simulated.modules[which(labels==1)]=
        sample(c("goodreg","badreg"),
               sum(labels==1),
               prob=c(pgood.reg,pbad.reg),
               replace=TRUE)
   }
   #電不好
   if(any(labels==-1)){
      simulated.modules[which(labels==-1)]=
       sample(c("goodirreg","badirreg"),
              sum(labels==-1),
              prob=c(pgood.irreg,pbad.irreg),
              replace=TRUE)
   }
   simulated.modules
}

ICmodule.sim()
S=10000
sim.table=replicate(S,ICmodule.sim())
dim(sim.table)

#壞的數量
badnum=c()
#壞的 電好
badnumreg=c()
#壞的 電不好
badnumirreg=c()
#電好
numreg=c()
#電不好
numirreg=c()
for(i in 1:ncol(sim.table)){
    badnumreg[i]=sum(sim.table[,i]=="badreg")
    badnumirreg[i]=sum(sim.table[,i]=="badirreg")
    badnum[i]=sum(sim.table[,i]=="badreg")+
              sum(sim.table[,i]=="badirreg")
    numreg[i]=sum(sim.table[,i]=="badreg")+
              sum(sim.table[,i]=="goodreg")
    numirreg[i]=sum(sim.table[,i]=="badirreg")+
                sum(sim.table[,i]=="goodirreg")
}

#p(兩個壞掉的)
sum(badnum==2)/S

sum(numreg==10 & badnum==2)
#兩個壞掉的且電好/兩個壞掉
sum(numreg==10 & badnum==2)/sum(badnum==2)

k=1
sum(badnum==k)
sum(numirreg>=1 & badnum==k)
sum(numirreg>=1 & badnum==k)/sum(badnum==k)


#John's payoff is deterministic
John_pay    = 12000

#The probability of getting the job from Vanessa
prob_interval    = seq(0.1,0.9,0.1)
#算各種offer拿到的機會

#Simulation times
simulation_times = 1:5000
```

0917_Chapter 1

```
#Earning if Vanessa gives an offer
#results1：reject John, if Vanessa = 1, take it,
else School
results1  = matrix(NA, nrow=length(prob_interval),
                   ncol=length(simulation_times))
rownames(results1) = prob_interval
```

```
colnames(results1) = simulation_times
#Earning if Vanessa does NOT give an offer
#建一個各種offer的矩陣
#results2：reject John, skip Vanessa, try School
results2  = matrix(NA,  nrow=length(prob_interval),
                  ncol=length(simulation_times))
rownames(results2) = prob_interval
colnames(results2) = simulation_times

#外圈0.1-0.9, 內圈：給定一個機率的情況下，做第一次到5000次的模擬
for(p in prob_interval){
  for(times in simulation_times){

    #The payoff from Vanessa is deterministic
    Vanessa_pay = 14000
    #The chance to get the job from Vanessa
    #模擬是否得到V的offer 得出0or1
    Vanessa_offer = sample(c(1,0),1,
                        prob = c(p,1-p))

    Vanessa_pay = Vanessa_offer*Vanessa_pay


    #The payoff from school is uncertain
    #模擬學校pay 機率會對應
    School_pay =
sample(c(21600,16800,12000,6000,0),1,
                      prob =
c(0.05,0.25,0.4,0.25,0.05))

    if(Vanessa_offer==1){
      results1[which(p==prob_interval),times] =
Vanessa_pay
    }else{
      results1[which(p==prob_interval),times] =
School_pay
    }
    #直接去學校
    results2[which(p==prob_interval),times] =
School_pay


  }
  results1
  cat("P(Vanessa Offer=1)=",p, "\n")
}


#Earnings when Vanessa gives the offer
#pctile 機率的範圍
pctile.range = c(0.05,0.1,0.25,0.5,0.75,0.9,0.95)
sim.pctile1 = matrix(NA, ncol=length(pctile.range),
                    nrow= length(prob_interval))
colnames(sim.pctile1)=pctile.range
rownames(sim.pctile1)=prob_interval

#apply(results1,1,summary)

for(i in 1:nrow(results1)){
  sim.pctile1[i,] =
    quantile(results1[i,],probs =pctile.range)
  #quantile 算百分位數
}

sim.pctile1

#Earnings from accepting the school offer
sim.pctile2 = matrix(NA, ncol=length(pctile.range),
                 nrow= length(prob_interval))
colnames(sim.pctile2)=pctile.range
rownames(sim.pctile2)=prob_interval

#apply(results2,1,summary)

for(i in 1:nrow(results2)){
  sim.pctile2[i,] = quantile(results2[i,],probs =
pctile.range)
}

sim.pctile2


#大於John需要至少多少獲得Vanessa工作的機會
#比John好的機率
Pbetter1=c()
#i = 1~9
for(i in 1:nrow(results1)){
    Pbetter1[i]=
      sum(results1[i,]>=John_pay )/ncol(results1)
    #比John好的機率 sum(pay>=$12000)/5000次
}

x11(width=8,height=5)
plot(prob_interval,Pbetter1,type='l',xaxt='n',lwd=3,
     xlab="P(Vanessa offer=1)",ylab="P(Earning>=John
offer)")
axis(1,seq(0.1,0.9,0.1))
#大於John的信心 = 0.95(討厭風險), V>0.8才會
abline(h=0.95,col='red',lty=2,lwd=2)
abline(h=0.9,col='green',lty=3,lwd=2)
```

0924Hello Kitty decisio analysis

```
collect.I=function(npurchased){
   costI=0
   cost.r=5
   cost.d=25
   cardsbought=sample(1:101,
                  size=npurchased,
                  replace=TRUE)
   nhit=length(unique(cardsbought))
   nmissed=101-nhit
   costI=cost.r*npurchased+cost.d*nmissed
   return(costI)
}
collect.I(0)
#S:Number of simulation runs
S=2000
costs.I=replicate(S,collect.I(200))
summary(costs.I)
sum(costs.I<=2525)/S

#seed 起點 亂設
set.seed(5566)

#儲存saving 平均
savings.I=c()
for(s in 1:505){
    savings.I[s]=
    mean(2525-replicate(S,collect.I(s)))
    if(s%%50==0)print(s)
}

x11(width=8,height=5)
par(mar=c(4,4,2,1))
plot(1:505,savings.I,type='l',ylim=c(0,1600),
     lwd=3,ylab="E[Savings]",xlab="Cards to Buy",
     xaxt="n")
axis(1,seq(0,505,50))
points(which.max(savings.I),
       savings.I[which.max(savings.I)],
       col='black',cex=1.5)
which.max(savings.I)


#Exchange allowed
prob.exchange.yes=0.3
collect.II=function(npurchased){
  costII=0
  cost.r=5
  cost.d=25
  cardsbought=sample(1:101,
```

```
                  size=npurchased,
                  replace=TRUE)
  nhit=length(unique(cardsbought))
  nleft=length(cardsbought)-nhit
  #
  if(nleft==0){
     nexchange.yes=0
     nmissed=101-nhit
     }else{
     nexchagne.yes=rbinom(1,min(101-nhit,nleft),
                         prob.exchange.yes)
     nmissed=101-nhit-nexchagne.yes
  }
  costII=cost.r*npurchased+cost.d*nmissed
  return(costII)
}
collect.II(0)

savings.II=c()
for(s in 1:505){
  savings.II[s]=
    mean(2525-replicate(S,collect.II(s)))
  if(s%%50==0)print(s)
}

lines(1:505,savings.II,col='blue',lwd=3,lty=2)
points(which.max(savings.II),
       savings.II[which.max(savings.II)],
       col='blue',cex=1.5)
which.max(savings.II)


#Exchange & resell allowed
#Assuming resell price=dealer price
prob.exchange.yes=0.3
prob.resell.yes=0.1
collect.III=function(npurchased){
  costIII=0
  cost.r=5
  cost.d=25
  cardsbought=sample(1:101,
                  size=npurchased,
                  replace=TRUE)
  nhit=length(unique(cardsbought))
  nleft=length(cardsbought)-nhit
  #
  if(nleft==0){
    nexchagne.yes=0
    nmissed=101-nhit
    }else{
    nexchagne.yes=rbinom(1,min(101-nhit,nleft),
                         prob.exchange.yes)
    nmissed=101-nhit-nexchagne.yes
  }
  nleft=nleft-nexchagne.yes
  if(nleft>0){
     nresell.ok=rbinom(1,nleft,prob.resell.yes)
  }else{
     nresell.ok=0
  }
  costIII=cost.r*npurchased+cost.d*nmissed-
        cost.d*nresell.ok
  return(costIII)
}
collect.III(0)

savings.III=c()
for(s in 1:505){
  savings.III[s]=
    mean(2525-replicate(S,collect.III(s)))
  if(s%%50==0)print(s)
}

lines(1:505,savings.III,col='green',lwd=3,lty=3)
points(which.max(savings.III),
       savings.III[which.max(savings.III)],
       col='green',cex=1.5)
```

```
which.max(savings.III)


#Exchange & resell allowed
#Assuming resell price<dealer price & both random
prob.exchange.yes=0.3
prob.resell.yes=0.1
collect.IV=function(npurchased){
  costIV=0
  cost.r=5
  #cost.d=25
  cardsbought=sample(1:101,
                  size=npurchased,
                  replace=TRUE)
  nhit=length(unique(cardsbought))
  nleft=length(cardsbought)-nhit
  #
  if(nleft==0){
    nexchagne.yes=0
    nmissed=101-nhit
  }else{
    nexchagne.yes=rbinom(1,min(101-nhit,nleft),
                         prob.exchange.yes)
    nmissed=101-nhit-nexchagne.yes
  }
  nleft=nleft-nexchagne.yes
  if(nleft>0){
    nresell.ok=rbinom(1,nleft,prob.resell.yes)
  }else{
    nresell.ok=0
  }
  if(nmissed==0 & nresell.ok==0){
     costIV=cost.r*npurchased
  }
  if(nmissed==0 & nresell.ok>0){
    cost.d.sell=
      sample(seq(5,15),nresell.ok,replace=TRUE)
    costIV=cost.r*npurchased-
          sum(cost.d.sell)
  }
  if(nmissed>0 & nresell.ok==0){
     cost.d.buy=
       sample(seq(20,30),nmissed,replace=TRUE)
     costIV=cost.r*npurchased+
          sum(cost.d.buy)
  }
  if(nmissed>0 & nresell.ok>0){
    cost.d.buy=
      sample(seq(20,30),nmissed,replace=TRUE)
    cost.d.sell=
      sample(seq(5,15),nresell.ok,replace=TRUE)
    costIV=cost.r*npurchased+
          sum(cost.d.buy)-
          sum(cost.d.sell)
  }

  return(costIV)
}
collect.IV(0)
costs.IV=replicate(S,collect.IV(500))
summary(costs.IV)
sum(costs.IV<=2525)/S

savings.IV=c()
for(s in 1:505){
  savings.IV[s]=
    mean(2525-replicate(S,collect.IV(s)))
  if(s%%50==0)print(s)
}

lines(1:505,savings.IV,col='red',lwd=3,lty=4)
points(which.max(savings.IV),
       savings.IV[which.max(savings.IV)],
       col='red',cex=1.5)
which.max(savings.IV)

legend(90,700,c("M1-Base","M2-Exchange","M3-Exchange
```

```
& Resell",
                "M4-Random Price"),
      lty=c(1,2,3,4),lwd=c(3,3,3,3),

bty="n",cex=1.25,col=c('black','blue','green','red')
)

which.max(savings.I)
which.max(savings.II)
which.max(savings.III)
which.max(savings.IV)
```

1001_Chapter 3

```
#抽樣
x = rbinom(10000, 50, 1/50)
x
table(x)
sum(x<=3)/10000

pbinom(3,50,1/50)

##S: The number of simulation runs
S=10000

set.seed(5566)

#Parameters of time for underwriting
mu1=150
sig1=30
#Parameters of time for rating
mu2=75
sig2=25

Time1=rnorm(S,mu1,sig1)
Time2=rnorm(S,mu2,sig2)

TotTime=Time1+Time2
hist(TotTime)
var(TotTime)

sum(TotTime<=180)/S
#What is the theoretical probability

quantile(TotTime,0.95)
#What is the theoretical percentile value?

S=10000

#ch3 p6
#Random processing time of cases 1-3 for
underwriting
Time11=rnorm(S,mu1,sig1)
Time12=rnorm(S,mu1,sig1)
Time13=rnorm(S,mu1,sig1)
#Random processing time of cases 1-3 for rating
Time21=rnorm(S,mu2,sig2)
Time22=rnorm(S,mu2,sig2)
Time23=rnorm(S,mu2,sig2)

##Assuming rating must wait for underwriting!
#Beginning time for the 2nd case of rating
#假設要等第一個人做完兩件事or自己完成第一件事的時間
#第二個人處理第二個case的時間
BeginTime22=c()
for(s in 1:S){
   BeginTime22[s]=max(Time11[s]+Time21[s],
                  Time11[s]+Time12[s])
}

#Beginning time for the 3rd case of rating
BeginTime23=c()
for(s in 1:S){
  BeginTime23[s]=max(Time11[s]+Time12[s]+Time13[s],
                 BeginTime22[s]+Time22[s])
}

#Ending time for the 3rd case of rating
EndTime=BeginTime23+Time23

summary(EndTime)
sum(EndTime<=480)/S



#製作矩陣
##Assuming X & Y are NOT independent of each other
corrXY=0.37
varcovMatrix=matrix(c(sig1^2,sig1*sig2*corrXY,sig1*s
ig2*corrXY,sig2^2),
                nrow=2,ncol=2)
varcovMatrix

library(MASS)
mvrnorm(10,mu=c(mu1,mu2),Sigma=varcovMatrix)
xy=mvrnorm(10000,mu=c(mu1,mu2),Sigma=varcovMatrix)
cor(xy[,1],xy[,2])

S=10000

Time1.corr=mvrnorm(S,mu=c(mu1,mu2),Sigma=varcovMatri
x)
Time2.corr=mvrnorm(S,mu=c(mu1,mu2),Sigma=varcovMatri
x)
Time3.corr=mvrnorm(S,mu=c(mu1,mu2),Sigma=varcovMatri
x)
#Random processing time of cases 1-3 for
underwriting
Time11.corr=Time1.corr[,1]
Time12.corr=Time2.corr[,1]
Time13.corr=Time3.corr[,1]
#Random processing time of cases 1-3 for rating
Time21.corr=Time1.corr[,2]
Time22.corr=Time2.corr[,2]
Time23.corr=Time3.corr[,2]

#Beginning time for the 2nd case of rating
BeginTime22.corr=c()
for(s in 1:S){

BeginTime22.corr[s]=max(Time11.corr[s]+Time21.corr[s
],
                    Time11.corr[s]+Time12.corr[s])
}

#Beginning time for the 3rd case of rating
BeginTime23.corr=c()
for(s in 1:S){

BeginTime23.corr[s]=max(Time11.corr[s]+Time12.corr[s
]+Time13.corr[s],

BeginTime22.corr[s]+Time22.corr[s])
}

#Ending time for the 3rd case of rating
EndTime.corr=BeginTime23.corr+Time23.corr

summary(EndTime.corr)

FinishTime1.corr=Time11.corr+Time21.corr
sum(FinishTime1.corr<=180)/S

quantile(FinishTime1.corr,0.95)

sum(EndTime.corr<=480)/S


##Estimating correlations/covariances from data
Fund1=c(65, 79, 85, 78, 107, 108, 124, 156, 195,
181, 216)
Fund2=c(47, 61, 73, 60, 89, 86, 104, 120, 140, 134,
175)
```

```
Fund3=c(38, 37, 39, 40, 47, 46, 57, 71, 74, 72, 87)
Fund4=c(61, 64, 74, 72, 95, 89, 114, 147, 146, 127,
152)

AnnualGR=matrix(0,nrow=(length(Fund1)-1),ncol=4)

AnnualGR

for(i in 2:length(Fund1)){
   AnnualGR[i-1,1]=Fund1[i]/Fund1[i-1]
   AnnualGR[i-1,2]=Fund2[i]/Fund2[i-1]
   AnnualGR[i-1,3]=Fund3[i]/Fund3[i-1]
   AnnualGR[i-1,4]=Fund4[i]/Fund4[i-1]
}

AnnualGR
#cov共變異 做矩陣
Sigma.est=cov(AnnualGR)
Sigma.est

#矩陣相關係數corr
cov2cor(Sigma.est)

mu.est=c(mean(AnnualGR[,1]),mean(AnnualGR[,2]),
       mean(AnnualGR[,3]),mean(AnnualGR[,4]))
mu.est

#import MASS
library(MASS)
MVN.AGR=mvrnorm(50,mu.est,Sigma.est)

Fund1.MVN=MVN.AGR[,1]
Fund2.MVN=MVN.AGR[,2]

#Ignore dependencies
Fund1.N=rnorm(50,mean(AnnualGR[,1]),sd(AnnualGR[,1])
)
Fund2.N=rnorm(50,mean(AnnualGR[,2]),sd(AnnualGR[,2])
)


x11(width=18,height=5)
par(mfrow=c(1,3))
plot(AnnualGR[,1],AnnualGR[,2],type='p',pch=1,lwd=5)
#有相關
plot(Fund1.MVN,Fund2.MVN,type='p',pch=2,lwd=4,col='r
ed')
#自己抽自己的
plot(Fund1.N,Fund2.N,type='p',pch=3,lwd=4,col='green
')
```

1001_The Hedging Problem

```
#DM put options
putDM=matrix(c(c(0.66,0.65,0.64,0.63,0.62,0.61,0.60,
0.59,0.55),

c(0.085855,0.032191,0.020795,0.017001,0.013711,

0.010851,0.008388,0.006291,0.001401)),ncol=2)

colnames(putDM)=c("kDM","cDM")

#BP put options
putBP=matrix(c(c(1.3,1.25,1.20,1.15,1.1,1.05,1,0.95,
0.9),

c(0.137213,0.082645,0.0450460,0.028348,0.016146,

0.007860,0.003277,0.001134,0.000245)),ncol=2)

colnames(putBP)=c("kBP","cBP")

#81種 因為各有9個可能，9*9=81
put.grid=expand.grid(putDM[,1],putBP[,1])

put.grid=cbind(put.grid[,1],rep(NA,nrow(put.grid)),
          put.grid[,2],rep(NA,nrow(put.grid)))

colnames(put.grid)=c(colnames(putDM),colnames(putBP)
)


for(i in 1:nrow(put.grid)){

put.grid[i,2]=putDM[which(putDM[,1]==put.grid[i,1]),
2]

put.grid[i,4]=putBP[which(putBP[,1]==put.grid[i,3]),
2]
}


#算HedgeRev
HedgeRevDM=function(DMfcst=645,currentDM=0.6513,nDM=
500,
               kDM,cDM,deltaDM){
    DMfcst*currentDM*(1+deltaDM/100)+
    nDM*(max(kDM-currentDM*(1+deltaDM/100),0)-cDM)
}


HedgeRevBP=function(BPfcst=272,currentBP=1.234,nBP=5
00,
               kBP,cBP,deltaBP){
    BPfcst*currentBP*(1+deltaBP/100)+
    nBP*(max(kBP-currentBP*(1+deltaBP/100),0)-cBP)
}


sigmaDM=9
sigmaBP=11
corrDMxBP=0.675
covDMxBP=sigmaDM*sigmaBP*corrDMxBP
#mu
mu.est=c(0,0)
#矩陣
covMatrix=matrix(c(sigmaDM^2,covDMxBP,covDMxBP,sigma
BP^2),nrow=2)
covMatrix

#轉回corr
cov2cor(covMatrix)

S=1000

library(MASS)
set.seed(9527)

ExRate=mvrnorm(S,mu.est,covMatrix)
#約等於0.675
cor(ExRate[,1],ExRate[,2])

CVARq5=rep(0,nrow(put.grid))
#平均收益
muRev=rep(0,nrow(put.grid))
sigRev=rep(0,nrow(put.grid))
#希望大於706
bottomlineRev=706
probtol=rep(0,nrow(put.grid))

#1~81種選擇
#S=1000
for(i in 1:nrow(put.grid)){
   revUS.temp=rep(0,S)
   for(s in 1:S){
     DMtoUS.s=HedgeRevDM(nDM=500,
              kDM=put.grid[i,1],cDM=put.grid[i,2],
              deltaDM=ExRate[s,1])
     BPtoUS.s=HedgeRevBP(nBP=500,
              kBP=put.grid[i,3],cBP=put.grid[i,4],
              deltaBP=ExRate[s,2])
     revUS.temp[s]=DMtoUS.s+BPtoUS.s
```

```
    }
    revUS.temp.q5=quantile(revUS.temp,0.05)

CVARq5[i]=mean(revUS.temp[which(revUS.temp<revUS.tem
p.q5)])
    muRev[i]=mean(revUS.temp)
    sigRev[i]=sd(revUS.temp)
    probtol[i]=sum(revUS.temp<bottomlineRev)/S
    print(i)
}

which.max(CVARq5)
which.max(muRev)
which.min(sigRev)
#81個組合裡，各個小於706的機率(越小越好)
which.min(probtol)

plot(probtol,CVARq5)
which(probtol<0.1 & CVARq5>690)
muRev[which(probtol<0.1 & CVARq5>690)]
summary(muRev)

put.grid[which(probtol<0.1 & CVARq5>690),]



##Each option bought for 55.55556 (i.e., 500/9)
##9個都買一樣多
HedgeRevDM.base=function(DMfcst=645,currentDM=0.6513
,nDM=500,
                         kDM,cDM,deltaDM){
  DMfcst*currentDM*(1+deltaDM/100)
}
#
HedgeRevDM.opt=function(DMfcst=645,currentDM=0.6513,
nDM=500,
                        kDM,cDM,deltaDM){
  nDM*(max(kDM-currentDM*(1+deltaDM/100),0)-cDM)
}
##
HedgeRevBP.base=function(BPfcst=272,currentBP=1.234,
nBP=500,
                         kBP,cBP,deltaBP){
  BPfcst*currentBP*(1+deltaBP/100)
}
#
HedgeRevBP.opt=function(BPfcst=272,currentBP=1.234,n
BP=500,
                        kBP,cBP,deltaBP){
  nBP*(max(kBP-currentBP*(1+deltaBP/100),0)-cBP)
}
###
revUS.base=rep(0,S)
revUS.opt=matrix(0,nrow=S,ncol=nrow(putDM))
for(s in 1:S){
  DMtoUS.base.s=HedgeRevDM.base(nDM=500/nrow(putDM),

kDM=putDM[i,1],cDM=putDM[i,2],
                              deltaDM=ExRate[s,1])
  BPtoUS.base.s=HedgeRevBP.base(nBP=500/nrow(putBP),

kBP=putBP[i,1],cBP=putBP[i,2],
                              deltaBP=ExRate[s,2])
  revUS.base[s]=DMtoUS.base.s+BPtoUS.base.s
}
#
for(i in 1:nrow(putDM)){
  revUS.opt.temp=rep(0,S)
  for(s in 1:S){
    DMtoUS.opt.s=HedgeRevDM.opt(nDM=500/nrow(putDM),
                        kDM=putDM[i,1],cDM=putDM[i,2],
                        deltaDM=ExRate[s,1])
    BPtoUS.opt.s=HedgeRevBP.opt(nBP=500/nrow(putBP),
                        kBP=putBP[i,1],cBP=putBP[i,2],
                        deltaBP=ExRate[s,2])
    revUS.opt.temp[s]=DMtoUS.opt.s+BPtoUS.opt.s
  }
  revUS.opt[,i]=revUS.opt.temp
  print(i)
}

revUS.equal=apply(revUS.opt,1,sum)+revUS.base
length(revUS.equal)

temp.q5=quantile(revUS.equal,0.05)
mean(revUS.equal[which(revUS.equal<temp.q5)])
mean(revUS.equal)
#懶人選擇較不好 低於706的機會變高 變成21.4%
sum(revUS.equal<bottomlineRev)/S



##Expand put options
nDM.opt=c(100L,300L,500L)
nBP.opt=c(100L,300L,500L)

n.opt=expand.grid(nDM.opt,nBP.opt)
colnames(n.opt)=c("nDM","nBP")

#1-729(9*81)
put.grid.ii=c()
for(j in 1:nrow(n.opt)){
    put.grid.temp=cbind(put.grid,
                        rep(n.opt[j,1],nrow(put.grid)),
                        rep(n.opt[j,2],nrow(put.grid)))
    put.grid.ii=rbind(put.grid.ii,put.grid.temp)
}


colnames(put.grid.ii)=c(colnames(put.grid),colnames(
n.opt))
head(put.grid.ii)

CVARq5=rep(0,nrow(put.grid.ii))
muRev=rep(0,nrow(put.grid.ii))
sigRev=rep(0,nrow(put.grid.ii))
bottomlineRev=706
probtol=rep(0,nrow(put.grid.ii))

for(i in 1:nrow(put.grid.ii)){
  revUS.temp=rep(0,S)
  for(s in 1:S){
    DMtoUS.s=HedgeRevDM(nDM=put.grid.ii[i,5],
                        kDM=put.grid.ii[i,1],
                        cDM=put.grid.ii[i,2],
                        deltaDM=ExRate[s,1])
    #
    BPtoUS.s=HedgeRevBP(nBP=put.grid.ii[i,6],
                        kBP=put.grid.ii[i,3],
                        cBP=put.grid.ii[i,4],
                        deltaBP=ExRate[s,2])
    revUS.temp[s]=DMtoUS.s+BPtoUS.s
  }
  revUS.temp.q5=quantile(revUS.temp,0.05)

CVARq5[i]=mean(revUS.temp[which(revUS.temp<revUS.tem
p.q5)])
  muRev[i]=mean(revUS.temp)
  sigRev[i]=sd(revUS.temp)
  probtol[i]=sum(revUS.temp<bottomlineRev)/S
  print(i)
}


which.max(CVARq5)
which.max(muRev)
which.min(sigRev)
which.min(probtol)

options(scipen = 999)
put.grid.ii[417,]
put.grid.ii[81,]
```

```
plot(probtol,CVARq5)
which(probtol<0.05 & CVARq5>700)
muRev[which(probtol<0.05 & CVARq5>700)]
summary(muRev)

#最好的買法
put.grid.ii[which(probtol<0.05 & CVARq5>700),]
```

1008_More Prob Distributions

```
##Case: Operations at Conley Fisheries
#The quantity of fish caught each day
fish.Q = 3500

##S: The number of simulation runs
S=10000

#expected price at Rock port
mu.PR = 3.65
#the standard devaition of the price at Rock port
sigma.PR = 0.2

#Simulated price at Rock port
sim.PR=rnorm(S,mu.PR,sigma.PR)

#Simulated demand at Rock port
sim.D = sample(c(0,1000,2000,3000,4000,5000,6000),S,
             prob =
c(0.02,0.03,0.05,0.08,0.33,0.29,0.2),
             replace = TRUE)

#The operation cost of the boat
oper.cost = 10000


F=c()
#F is the stochastic revenue when selling all the
fish at Rockport
for(s in 1:S){
  F[s] = sim.PR[s]*min(fish.Q, sim.D[s]) - oper.cost
}


summary(F)
hist(F, breaks=200)

#打敗G港
sum(F>1375)/S
#虧錢
sum(F<0)/S

quantile(F,0.975)
quantile(F,0.025)

lowestq5=F[which(F<=quantile(F,0.05))]
CVaRq5=mean(lowestq5)
CVaRq5



##Complications of Operations at Conley Fisheries
S=1000
PR.Rock=rnorm(S,3.65,0.25)
PR.Glou=rnorm(S,3.5,0.5)
summary(PR.Rock)
summary(PR.Glou)

x11(width=12,height=5)
par(mfrow=c(1,2))
hist(PR.Rock,breaks=50)
hist(PR.Glou,breaks=50)


install.packages("EnvStats")
#Install the package above if needed
library(EnvStats)
D.Glou=round(rtri(S,2000,6000,5000),0)
D.Rock=sample(c(0,1000,2000,3000,4000,5000,6000),S,
            prob =
c(0.02,0.03,0.05,0.08,0.33,0.29,0.2),
            replace = TRUE)

x11(width=12,height=5)
par(mfrow=c(1,2))
hist(D.Rock)
hist(D.Glou)


##S: The number of simulation runs
S=10000

#The quantity of fish caught each day
fullload = 3500
frac=runif(S,0.7,1)

fish.Q=round(fullload*frac,0)


#expected price at Rock port
mu.PRR = 3.65
#the standard devaition of the price at Rock port
sigma.PRR = 0.2
#expected price at Glou
mu.PRG = 3.5
#the standard devaition of the price at Glou
sigma.PRG = 0.5

#Simulated prices
sim.PRR=rnorm(S,mu.PRR,sigma.PRR)
sim.PRG=rnorm(S,mu.PRG,sigma.PRG)

#Simulated demand
sim.DR =
sample(c(0,1000,2000,3000,4000,5000,6000),S,
             prob =
c(0.02,0.03,0.05,0.08,0.33,0.29,0.2),
             replace = TRUE)

sim.DG = round(rtri(S,2000,6000,5000),0)

#The operation cost of the boat
oper.cost = 10000


F=c()
G=c()
#F is the stochastic revenue when selling all the
fish at Rockport
for(s in 1:S){
  F[s] = sim.PRR[s]*min(fish.Q[s], sim.DR[s]) -
oper.cost
  G[s] = sim.PRG[s]*min(fish.Q[s], sim.DG[s]) -
oper.cost
}


summary(G)
summary(F)
x11(width=12,height=5)
par(mfrow=c(1,2))
hist(G, breaks=200)
hist(F, breaks=200)

sum(G>1375)/S
sum(G<0)/S

sum(F>1375)/S
sum(F<0)/S

quantile(G,0.975)
quantile(G,0.025)

quantile(F,0.975)
quantile(F,0.025)
```

```r
#最壞的5%
lowestq5G=G[which(G<=quantile(G,0.05))]
CVaRq5G=mean(lowestq5G)
CVaRq5G

lowestq5F=F[which(F<=quantile(F,0.05))]
CVaRq5F=mean(lowestq5F)
CVaRq5F



##Modeling exponentially distributed time
#一個小時處理幾通電話
S=10000
calls=c()
for(s in 1:S){
    k=0
    totaltime=0
    while(totaltime<=60){
      totaltime=totaltime+rexp(1,1/10)
      k=k+1
      cat("totaltime=",totaltime,"; k=",k,"\n")
    }
    calls[s]=k-1
    if(s%%1000==0)print(s)
}

plot(table(calls)/S)
#理論值
lines(min(calls):max(calls),

dpois(min(calls):max(calls),6),col='red',lty=2,lwd=3
)


##Verify Memoryless
Tsamples=rexp(S,1/10)
sum(Tsamples>5)/S
sum(Tsamples>15)/sum(Tsamples>10)


##Verify gamma distribution
S=10000
time.five=c()
for(s in 1:S){
  time.five[s]=sum(rexp(5,1/10))
}

min.x=round(min(time.five),2)
max.x=round(max(time.five),2)
x=seq(min.x, max.x, 0.01)
shape.est=5
scale.est=10
hist(time.five,breaks=50,freq=FALSE)
#理論值
lines(x,dgamma(x,shape=shape.est,scale=scale.est),co
l='red',lwd=3)


x=seq(1,30,0.1)
plot(x,dexp(x,0.2),type='l',lwd=2)
lines(x,dgamma(x,shape=1,scale=1/0.2),col='red',lty=
2)
lines(x,dgamma(x,shape=2,scale=1/0.2),col='blue')
lines(x,dgamma(x,shape=5,scale=1/0.2),col='green')




##Project duration Simulation
taskt.mean=c(20,50,60,15,65,35,30,10)
taskt.stdev=c(7,10,12,3,30,15,5,3)
#Assuming task time as a RV X ~ gamma(shape, scale)
#E[X]=shape*scale & Var[X]=shape*scale^2
shape.est=c()
scale.est=c()
scale.est=(taskt.stdev)^2/taskt.mean
scale.est

shape.est=taskt.mean/scale.est
shape.est
#Need to check both parameters>0 or not

#Define BT:Begin Time & FT: Finish Time

S=10000
#遲一天罰金
penaltyperday=100000
B.reduced=1
simDays.temp=c()
simPenalty.temp=c()
for(i in 1:S){
#i:index for the ith simulation
    taskt.i=c()
    for(j in 1:length(taskt.mean)){
    #j:activity index
        shape.j=shape.est[j]
        scale.j=scale.est[j]
        taskt.i.j=rgamma(1,shape=shape.j,
                         scale=scale.j)
        taskt.i[j]=round(taskt.i.j,0)
    }
    ##Assuming NO reduction in task B time
    #BT beginning time, ET end time
    BT.A=BT.C=BT.E=0
    ET.A=BT.A+taskt.i[1]
    ET.C=BT.C+taskt.i[3]
    ET.E=BT.E+taskt.i[5]
    #
    BT.B=ET.A
    #不花錢找外包
    if(B.reduced==0){
       ET.B=BT.B+taskt.i[2]
    }
    ##花錢找外包
    if(B.reduced==1){
      ET.B=BT.B+round(taskt.i[2]*0.8,0)
    }
    #
    BT.D=max(ET.B,ET.C)
    ET.D=BT.D+taskt.i[4]
    #
    BT.F=ET.E
    ET.F=BT.F+taskt.i[6]
    #
    BT.G=ET.D
    ET.G=ET.D+taskt.i[7]
    #
    BT.H=ET.G
    ET.H=BT.H+taskt.i[8]
    #H or F結束
    simDays.temp[i]=max(ET.H, ET.F)
    #算有沒有delay
    delay.i=max(simDays.temp[i]-130,0)
    simPenalty.temp[i]=delay.i*penaltyperday
}

simDays.base=simDays.temp
simPenalty.base=simPenalty.temp
#
summary(simDays.base)
sd(simDays.base)
sum(simDays.base<=130)/S
#
summary(simPenalty.base)


simDays.reduced=simDays.temp
simPenalty.reduced=simPenalty.temp
```

```
summary(simDays.reduced)
sd(simDays.reduced)
sum(simDays.reduced<=130)/S
#
summary(simPenalty.reduced)
```

1015_Optimization of Decision Variables

```
#Below is a function that simulates random samples
from
#the Myerson distribution in section 4.4 of Roger
Myerson's 2005 book
#This is a generalized version of normal & lognormal
distribution
rMyerson <- function(n,q1,q2,q3,lower=-Inf,
upper=Inf,tl=0.5){
  #n: the number of random samples
  #q1: xx percentile, xx<50 & usually xx=25
  #q2: 50 percentile
  #q3: xx percentile, xx>50 & usually xx=75
  #lower: minimum possible value
  #upper: maximum possible value
  #tl: tail probability = P(X<q1)+P(X>q3)
  ######################################
  x=runif(n)
  x[which(x>0.999999)]=0.999999
  x[which(x<0.000001)]= 0.000001
  #Above avoids sampling values that are too extreme
  norml=qnorm(x)/qnorm(1-tl/2)
  br=(q3-q2)/(q2-q1)
  if(br==1){
    res=q2+(q3-q2)*norml
  } else {
    res=q2+(q3-q2)*(br^norml-1)/(br-1)
  }
  pmin(pmax(res, lower), upper)
}


##Case: Scotia Snowboards
#P(weather is cold)
p.cold=1/3
#Demand parameters
q1.normal=60000
q2.normal=75000
q3.normal=90000

q1.cold=80000
q2.cold=100000
q3.cold=125000

#Cost parameters
unitcost=20
unitprice=48
salvage=8


#x as the order quantity. Try different ordering
decisions
x.val=seq(50000,150000,1000)

#Number of simulation runs
S=15000

#Simulate random demand
#1: cold; 0:weather
#模擬各個需求的可能
sim.weather=sample(c(1,0),S,replace=TRUE,
                prob=c(p.cold,1-p.cold))

sim.demand=rep(0,S)
for(s in 1:S){
  #1: cold; 0:weather
  if(sim.weather[s]==1){

sim.demand[s]=rMyerson(1,q1.cold,q2.cold,q3.cold,
                       lower=0)
  }else{

sim.demand[s]=rMyerson(1,q1.normal,q2.normal,q3.norm
al,
                       lower=0)
  }
  sim.demand[s]=round(sim.demand[s],0)
}
sim.demand

#算利潤
profit=function(x=80000,d){
  #d: demand realizations
  profit.val=c()
  for(i in 1:length(d)){
    profit.val[i]=(unitprice-unitcost)*min(x, d[i])+
      (salvage-unitcost)*max(x-d[i], 0)
  }
  profit.val
}


sim.profit=matrix(0,nrow=S,ncol=length(x.val))
avg.profit=c()
sd.profit=c()

start.time <- Sys.time()

for(i in 1:length(x.val)){
  sim.profit[,i]=profit(x.val[i],d=sim.demand)
  avg.profit[i]=mean(sim.profit[,i])
  sd.profit[i]=sd(sim.profit[,i])
  cat("production quantity:", x.val[i], "\n")
}

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken

x11(width=12,height=5)
par(mfrow=c(1,2))
plot(x.val,avg.profit,type='l',xlab="production
quantity",lwd=3)
plot(x.val[which(avg.profit>1800000)],
    avg.profit[which(avg.profit>1800000)],
    type='l',xlab="production quantity",lwd=3)


x.val[which.max(avg.profit)]
max(avg.profit)



##Assess the value of perfect information
cu=unitprice-unitcost
co=unitcost-salvage
cu
co
#Calculate the critical fractile
frac=cu/(cu+co)
frac

#Compute Q* for cold weather
#冬天可能的需求
dcold=rMyerson(S,q1.cold,q2.cold,q3.cold,lower=0)
hist(dcold)
x.cold=quantile(dcold,frac,names=FALSE)
x.cold=round(x.cold,0)
x.cold


#Compute Q* for normal weather
dnormal=rMyerson(S,q1.normal,q2.normal,q3.normal,low
er=0)
hist(dnormal)
x.normal=quantile(dnormal,frac,names=FALSE)
x.normal=round(x.normal,0)
```

```
x.normal

profit.x.cold=profit(x.cold,dcold)
profit.x.normal=profit(x.normal,dnormal)
mean(profit.x.cold)
mean(profit.x.normal)

frac
#知道是否偏冷
profit.perfectinfo=c()

for(s in 1:S){
  #1: cold; 0:weather
  if(sim.weather[s]==1){
    x=x.cold
  }else{
    x=x.normal
  }
  demand=sim.demand[s]
  profit.perfectinfo[s]=
    (unitprice-unitcost)*min(x, demand)+
    (salvage-unitcost)*max(x-demand, 0)
}

mean(profit.perfectinfo)

max(avg.profit)

mean(profit.perfectinfo)-max(avg.profit)



##A Simple Biding Problem
#cost parameters
cost.q1=86
cost.q2=100
cost.q3=120

#parameters of opponents' bids
oppbid.q1=120
oppbid.q2=140
oppbid.q3=180

#Number of simulation runs
S=20000

opponents=sample(c(1,2,3,4,5),S,prob=c(0.2,0.3,0.3,0
.1,0.1),
                 replace=TRUE)
oppbids=list()
for(s in 1:S){

oppbids.s=rMyerson(opponents[s],oppbid.q1,oppbid.q2,
oppbid.q3,
                   lower=0)
  oppbids[[s]]=round(oppbids.s,0)
}
sim.cost=rMyerson(S,cost.q1,cost.q2,cost.q3,lower=0)


#b: the bidding value 我出的價格
bidding=function(b,oppb,c){
  win=all(oppb>b)
  profit=win*(b-c)
  c(win,profit)
}

#Try different bidding values
b.val=seq(100,200,5)

sim.win=matrix(0,nrow=S,ncol=length(b.val))
sim.profit=matrix(0,nrow=S,ncol=length(b.val))
avg.profit=c()
sd.profit=c()
wins=c()

start.time <- Sys.time()
```

```
for(i in 1:length(b.val)){
  for(s in 1:S){

temp=bidding(b=b.val[i],opp=oppbids[[s]],c=sim.cost[
s])
    sim.win[s,i]=temp[1]
    sim.profit[s,i]=temp[2]
  }
  wins[i]=sum(sim.win[,i])
  avg.profit[i]=mean(sim.profit[,i])
  sd.profit[i]=sd(sim.profit[,i])
  #
  cat("bidding value:", b.val[i], "\n")
}

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken

x11(width=8,height=5)
plot(b.val,avg.profit,type='l',xlab="bidding
value",lwd=3)
abline(h=0,lty=2,col='red',lwd=2)

b.val[which.max(avg.profit)]
max(avg.profit)

wins[which.max(avg.profit)]/S



##The Winners' Curse
#Capture stochastic dependencies between C and A
#corrleation coefficient
#corr假設0.4
corr.opp.cost=0.4

#Below simulates bivariate standard normal variates
#both have standard deviation of 1
cov.opp.cost=corr.opp.cost*1*1
#variance-covariance matrix
covmat.opp.cost=matrix(c(1,cov.opp.cost,
                         cov.opp.cost,1),nrow=2)
covmat.opp.cost
library(MASS)
binorm=mvrnorm(S,c(0,0),covmat.opp.cost)
#check the correlation of simulated values
cor(binorm[,1],binorm[,2])

#Transform the simulated standard normal variates
#into uniform (0, 1) variates
birand1=pnorm(binorm[,1])
birand2=pnorm(binorm[,2])
birand=cbind(birand1,birand2)
#check correlation between two uniform(0, 1)
variates
cor(birand[,1],birand[,2])

apply(birand,2,summary)
x11(width=12, height=5)
par(mfrow=c(1,2))
hist(birand[,1])
hist(birand[,2])

##Pre-draw indepedent oppbids & costs
sim.oppbids.ind=rep(0,S)
sim.cost.ind=rMyerson(S,cost.q1,cost.q2,cost.q3,lowe
r=0)
for(s in 1:S){

oppbids=rMyerson(opponents[s],oppbid.q1,oppbid.q2,op
pbid.q3,lower=0)
  sim.oppbids.ind[s]=round(min(oppbids),0)
}

start.time <- Sys.time()
```

```
##Pre-draw correlated oppbids & costs
#有相關性的 藍線
sim.oppbids.corr=rep(0,S)
sim.cost.corr=rep(0,S)
for(s in 1:S){

sim.oppbids.corr[s]=round(quantile(sim.oppbids.ind,b
irand[s,1]),0)

sim.cost.corr[s]=quantile(sim.cost.ind,birand[s,2])
}

cor(sim.oppbids.corr,sim.cost.corr)

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken

sim.win.corr=matrix(0,nrow=S,ncol=length(b.val))
sim.profit.corr=matrix(0,nrow=S,ncol=length(b.val))
avg.profit.corr=rep(0,length(b.val))
sd.profit.corr=rep(0,length(b.val))
wins.corr=rep(0,length(b.val))


for(i in 1:length(b.val)){
  oppbids.i=sim.oppbids.corr
  if(any(b.val[i]<oppbids.i)){
    winnings=which(b.val[i]<oppbids.i)
    sim.win.corr[winnings,i]=1
    wins.corr[i]=sum(sim.win.corr[,i])
    #
    cost.i=sim.cost.corr
    sim.profit.corr[winnings,i]=b.val[i]-
cost.i[winnings]
    avg.profit.corr[i]=mean(sim.profit.corr[,i])
    sd.profit.corr[i]=sd(sim.profit.corr[,i])
  }
  #
  cat("bidding value:", b.val[i], "\n")
}



y.max=max(c(avg.profit,avg.profit.corr))
y.min=min(c(avg.profit,avg.profit.corr))

x11(width=8,height=5)
plot(b.val,avg.profit,type='l',lwd=3,
     xlab="bidding value",
     ylim=c(y.min,y.max))
abline(h=0,lty=2,col='red',lwd=2)
lines(b.val,avg.profit.corr,lty=3,col='blue',lwd=3)

b.val[which.max(avg.profit.corr)]
max(avg.profit.corr)

wins.corr[which.max(avg.profit.corr)]/S

b.val[which.max(avg.profit)]
max(avg.profit)

wins[which.max(avg.profit)]/S
```

1022_Stevens

```
##Supreme Court Cases
#大法官決策預測
# Read in the data
Stevens =
read.csv("/Users/julieyao/Documents/nccu/2019fall/De
cisionSciences/R Codes/Stevens.csv")
str(Stevens)

# Split the data
library(caTools)
library(rpart)
library(rpart.plot)
library(partykit)
set.seed(9527)
spl = sample.split(Stevens$Reverse, SplitRatio =
0.7)
Train = subset(Stevens, spl==TRUE)
Test = subset(Stevens, spl==FALSE)


##Classification trees
Train.df=Train
Train.df$Reverse=ifelse(Train.df$Reverse==1,"yes","n
o")
Test.df=Test
Test.df$Reverse=ifelse(Test.df$Reverse==1,"yes","no"
)

# Build a decision tree model
StevensTree = rpart(Reverse ~ Circuit + Issue +
Petitioner +
                      Respondent + LowerCourt + Unconst,
                    data = Train.df, method="class",
minbucket=15,
                    parms = list(split="information"))

plot(as.party(StevensTree),type="simple")
plot(as.party(StevensTree),type="extended")

#隨機森林
library(randomForest)
# Build a random forest model
StevensForest = randomForest(as.factor(Reverse) ~
Circuit + Issue +
                               Petitioner + Respondent +
LowerCourt +
                               Unconst, data = Train,
ntree=200,
                             nodesize=25)




StevensForest = randomForest(as.factor(Reverse) ~
Circuit + Issue +
                               Petitioner + Respondent +
LowerCourt + Unconst,
                             data = Train, ntree=500,
nodesize=25,
                             importance=TRUE)
varImpPlot(StevensForest)



##Logistic regression
StevensLogit=glm(Reverse ~ Circuit + Issue +
Petitioner +
                   Respondent + LowerCourt + Unconst,
                 data=Train,
family=binomial(link="logit"))
summary(StevensLogit)

# ROC curve
library(ROCR)
PredictLogit=predict(StevensLogit,newdata=Test,
type="response")
round(PredictLogit,3)
#閥值/決策臨界值
t=0.5
yhat=ifelse(PredictLogit>=t,1,0)
actual=Test$Reverse
table(actual,yhat)
predLogit = prediction(PredictLogit, Test$Reverse)
#
x11(width=8,height=5)
plot(performance(predLogit,
"tpr","fpr"),col='blue',lty=3,lwd=3)
abline(0,1)
```

```
performance(predLogit, "auc")

ROCRperf=performance(predLogit, "tpr","fpr")

x11(width=8,height=5)
#threshold on the right
#Add threshold labels
plot(ROCRperf, colorize=TRUE,
print.cutoffs.at=seq(0,1,by=0.1))


# ROC curve
library(ROCR)
PredictLogit=predict(StevensLogit,newdata=Test,
type="response")
PredictTree = predict(StevensTree, newdata =
Test.df, type = "prob")
PredictForest = predict(StevensForest, newdata =
Test, type = "prob")
predLogit = prediction(PredictLogit, Test$Reverse)
predTree = prediction(PredictTree[,2], Test$Reverse)
predForest = prediction(PredictForest[,2],
Test$Reverse)
#
x11(width=8,height=5)
plot(performance(predLogit,
"tpr","fpr"),col='blue',lty=3,lwd=3)
plot(performance(predTree, "tpr",
"fpr"),col='green',add=T,lty=4,lwd=3)
plot(performance(predForest, "tpr",
"fpr"),col='red',add=T,lty=3,lwd=3)
abline(0,1,lty=2)

performance(predLogit, "auc")
performance(predTree, "auc")
performance(predForest, "auc")
```