Comparaison of methods

Scan statistique - Méthode de Monte Carlo et calcul de p-value

Import libraries

```
library("localScore")
```

```
## Warning: package 'localScore' was built under R version 4.0.5
```

```
library("latex2exp")
```

```
## Warning: package 'latex2exp' was built under R version 4.0.5
```

```
library("Rcpp")
```

```
## Warning: package 'Rcpp' was built under R version 4.0.5
```

```
library("caret")
```

```
## Warning: package 'caret' was built under R version 4.0.5
## Loading required package: ggplot2
## Loading required package: lattice
```

1. Proposition for simulations under $\mathcal{H}_1$

In this part, we propose a method that simulates a Poisson process under the hypothesis $\mathcal{H}_1$. The idea is to simulate a sample under $\mathcal{H}_0$, and add randomly a subsequence under the alternative hypothesis in this sequence.

```
PoissonProcess <- function(lambda,T) {
  return(sort(runif(rpois(1,lambda*T),0,T)))
}

SimulationH1 <- function(lambda0, lambda1,T,tau){
    ppH0=PoissonProcess(lambda0,T)
    ppH1.segt=PoissonProcess(lambda1,tau)
    dbt=runif(1,0,T-tau)
    ppH0bis=PoissonProcess(lambda0,T)
    ppH1.repo=dbt+ppH1.segt
    ppH0_avant=ppH0bis[which(ppH0bis<ppH1.repo[1])]
    ppH0_apres=ppH0bis[which(ppH0bis>ppH1.repo[length(ppH1.repo)])]
    ppH1=c(ppH0_avant,ppH1.repo,ppH0_apres)
    return (ppH1)
}
```

```
TimeBetweenEvent <- function(pp){
    n=length(pp)
    tbe=pp[2:n]-pp[1:n1-1]
    tbe=c(0,tbe)
```

```
    return (tbe)
}

DataFrame <- function(pp,tbe){
    list=data.frame(ProcessusPoisson=pp, TimeBetweenEvent=tbe)
}
```

2. Simulation of the sequences under $\mathcal{H}_0$ via a Monte Carlo Method

In this part, we will try to simulate, using a Monte Carlo method, a set of 10^5 independant samples, under the assumption that $\lambda = \lambda_0$, hence, that we are under the null hypothesis $\mathcal{H}_0$.

```
ScanStat <- function(pp, T, tau){
    n=length(pp)
    stop=n-length(which(pp>(T-tau)))
    ScanStat=0
    for (i in (1:stop)) {
        x=which((pp>=pp[i])&(pp<=(pp[i]+tau)))
        scan=length(x)
        if (scan>ScanStat) {ScanStat=scan}
  }
    return (c(i,ScanStat))
}
```

We test the scan statistic method for different values of λ_0. The method of scan statistic we implemented will allow us to have access to the scan test statistic and where it happens in the sequence.

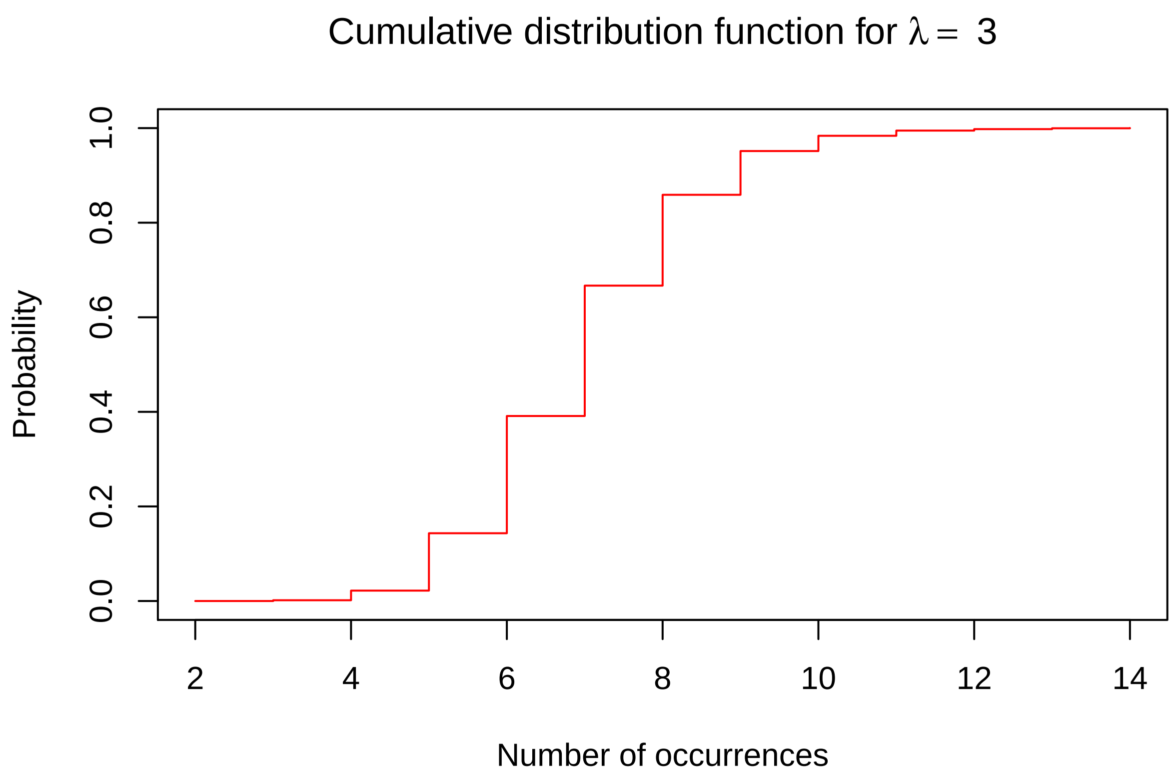
```
EmpDistrib <- function(lambda, n_sample,T,tau){
    pp=PoissonProcess(lambda,T)
    scan=c(ScanStat(pp,T, tau)[2])
    index=c(ScanStat(pp,T, tau)[1])
    for (i in 2:(n_sample)){
        pp=PoissonProcess(lambda,T)
        scan=rbind(scan,ScanStat(pp,T, tau)[2])
        index=rbind(index,ScanStat(pp,T, tau)[1])
    }
    min_scan=min(scan)-1
    max_scan=max(scan)
    table1=table(factor(scan, levels = min_scan:max_scan))
    EmpDis=data.frame(cdf=cumsum(table1)/sum(table1), proba=table1/sum(table1), index_scan=min_scan:max_
    EmpDis<-EmpDis[,-2]
    return(EmpDis)
    }
```

```
Plot_CDF <- function(lambda,n_sample,T,tau){
    Emp=EmpDistrib(lambda,n_sample,T,tau)
    title=TeX(paste(r'(Cumulative distribution function for $\lambda=$)', lambda))
    plot(Emp$index_scan, Emp$cdf,type="s",xlab="Number of occurrences",ylab="Probability", main=title, 
    return(Emp)
}
```

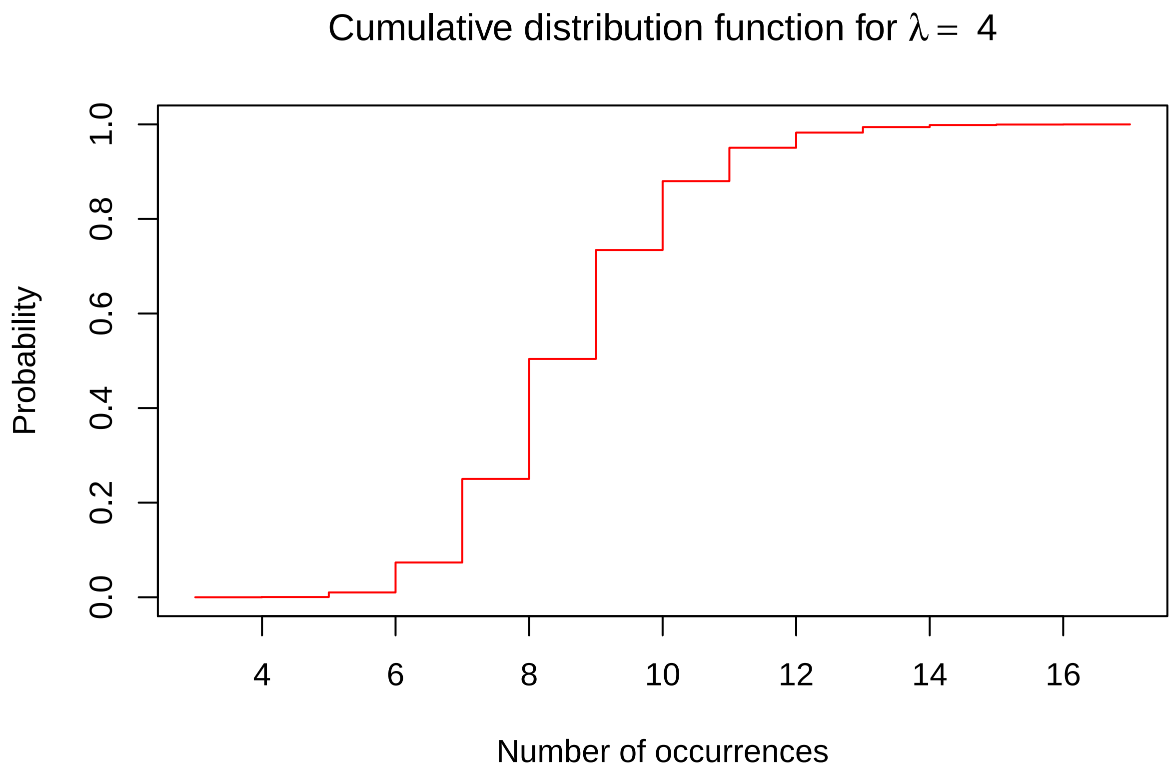
2.1 Test of $\mathcal{H}_0 : \lambda = \lambda_0$ against $\mathcal{H}_0 : \lambda = \lambda_1$, where $\lambda_1 > \lambda_0$

In this part, we will test different values for λ_0 and λ_1, and compute the probability of occurrence of a certain scan statistic.

```
#Empiricial distribution under H0
n_sample=10**4
lambda0=3
T=10
tau=1
ppH0=PoissonProcess(lambda0,T)
CDF=Plot_CDF(lambda0,n_sample,T,tau)
```




```
n_sample=10**4
lambda1=4
T=10
tau=1
ppH0=PoissonProcess(lambda1,T)
CDF=Plot_CDF(lambda1,n_sample,T,tau)
```




```
PValue <- function(Emp,ppH1, T, tau){
    scanH1=ScanStat(ppH1,T,tau)[2]
    index_scanH1=ScanStat(ppH1,T,tau)[1]
    index=Emp$index_scan
    n=length(index)
    if (scanH1< min(Emp$index_scan)){
        return (c(scanH1,1,index_scanH1))
        } else{
            if(min(Emp$index_scan)<scanH1 && scanH1<=max(Emp$index_scan)){
                return(c(scanH1,1-Emp$cdf[scanH1-min(Emp$index_scan)+1],index_scanH1))
            } else{return (c(scanH1,0,index_scanH1))}}
}
```

2.2. Simulation under $\mathcal{H}_0$ and computation of p-values

On simule des séquences sous $\mathcal{H}_0$, que l'on stocke. On calcule la valeur de la scan stat et de la p-value, que l'on stocke aussi. On a une séquence de p-valeur des scans et une séquence de score local.

```
NbSeqH0=10000
NbSeqH1=NbSeqH0
DataH0=vector("list")
DataH1=vector("list")
lambda0=4
lambda1=10
T=10
tau=1
```

```r
#Creation of a sequence that contains the sequence simulated under the null hypothesis
for (i in 1:NbSeqH0) {
    ppi=PoissonProcess(lambda0,T)
    DataH0[[i]]=ppi
}

#Creation of a sequence that contains the sequence simulated under the alternative hypothesis
seqH1begin=c()
for (i in 1:NbSeqH1) {
    pphi=SimulationH1(lambda0, lambda1,T,tau)
    DataH1[[i]]=pphi
}

#Computation of the time between events
TimeBetweenEventList <- function(list,n_list){
    TBE=vector("list",length=n_list)
    for (i in (1:n_list)) {
        ppi=list[[i]]
        ni=length(ppi)
        tbei=ppi[2:ni]-ppi[1:ni-1]
        TBE[[i]]=tbei
    }
    return (TBE)
}
tbe0=TimeBetweenEventList(DataH0,NbSeqH0)
```

We compute the p-value associated to all 5 sequences, and stock them in a vector.

```r
#We start by computing the empirical distribution for lambda0
Emp = EmpDistrib(lambda0,n_sample,T,tau)
scan = c()
pvalue = c()
index_scan = c()

#Then, we stock the p-value and the
for (i in 1:NbSeqH0){
    ppi = DataH0[[i]]
    result = PValue(Emp,ppi,T,tau)
    scan = c(scan,result[1])
    pvalue = c(pvalue,result[2])
    index_scan = c(index_scan,result[3])
}

ScS_H0=data.frame(num=(1:NbSeqH0), scan_stat=scan, pvalue_scan=pvalue,class=c(pvalue<0.05))
sum(ScS_H0$class[which(ScS_H0$class==TRUE)])/NbSeqH0
```

```
## [1] 0.1164
```

```r
#We start by computing the empirical distribution for lambda0
scan=c()
pvalue=c()
index_scan=c()

#Then, we stock the p-value and the
for (i in 1:NbSeqH1){
```

```
    ppi=DataH1[[i]]
    result=PValue(Emp,DataH1[[i]],T,tau)
    scan=c(scan,result[1])
    pvalue=c(pvalue,result[2])
    index_scan=c(index_scan,result[3])
}
ScS_H1=data.frame(num=1:NbSeqH1, scan_stat=scan, pvalue_scan=pvalue, class=(pvalue<0.05), begin_scan=inc
sum(ScS_H1$class[which(ScS_H1$class==TRUE)])/NbSeqH1
```

```
## [1] 0.6585
```

```
ScanStatMC <- function(NbSeq, T, tau, Emp, pp0){
    scan=c()
    pvalue=c()
    index_scan=c()

    for (i in 1:NbSeq){
        ppi=pp0[[i]]
        result=PValue(Emp,ppi,T,tau)
        scan=c(scan,result[1])
        pvalue=c(pvalue,result[2])
        index_scan=c(index_scan,result[3])
    }

    ScS_H0=data.frame(num=(1:NbSeq), scan_stat=scan, pvalue_scan=pvalue,class=c(pvalue<0.05))
    return(ScS_H0)
}
```

3. Local score

Distribution of scores via Monte Carlo

```
ComputeE <- function(lambda0, lambda1){
    E = 1
    maxXk = floor(E*(log(lambda1/lambda0)))
    while (maxXk < 3) {
        E = E+1
        maxXk = floor(E*(log(lambda1/lambda0)))
    }

    return (E)
}
```

```
ScoreDistribEmpiric <- function(lambda0, lambda1, n_sample, T){
    E = ComputeE(lambda0, lambda1)
    Score = c()

    for (i in 1:n_sample){
        ppH0 = PoissonProcess(lambda0,T)
        n1 = length(ppH0)
        tbe0 = ppH0[2:n1]-ppH0[1:n1-1]
        X = floor(E*(log(lambda1/lambda0)+(lambda0-lambda1)*tbe0))
        Score=c(Score,X)
    }
```

```r
    min_X = min(Score)
    max_X = max(Score)

    P_X = table(factor(Score, levels = min_X:max_X))/sum(table(Score))
    df = data.frame("Score_X" = min(Score):max(Score), "P_X" = P_X)
    df <- df[,-2]

    return (df)
}
```

```r
ScoreDistribElisa <- function(lambda0, lambda1, T){
    E = ComputeE(lambda0, lambda1)

    score_max = floor(E*log(lambda1/lambda0))

    ## score_min compute
    score_min_c = floor(E*log(lambda1/lambda0)+E*(lambda0-lambda1)*T)

    l = seq(score_min_c,score_max,1)
    borne_inf = (l-E*log(lambda1/lambda0))/(E*(lambda0-lambda1))
    borne_sup = (l+1-E*log(lambda1/lambda0))/(E*(lambda0-lambda1))
    proba.l = pexp(rate=lambda0,borne_inf)-pexp(rate=lambda0,borne_sup)
    S = sum(proba.l)
    new.proba.s = proba.l/S
    df = data.frame("Score_X" = l, "P_X" = new.proba.s)

    return (df)
}
```

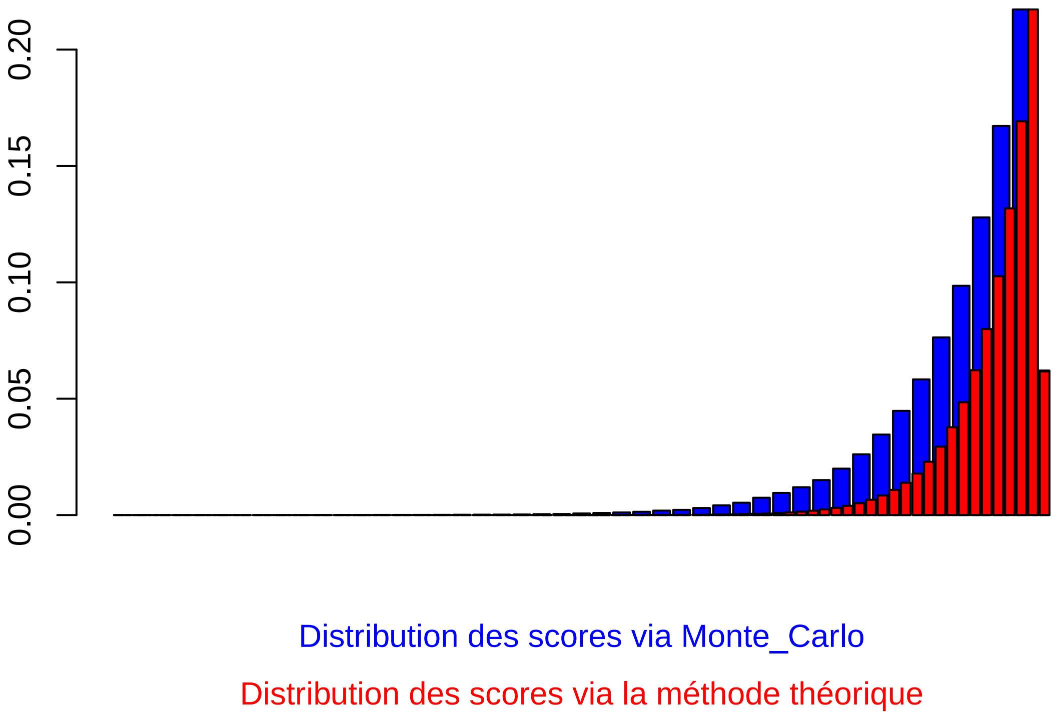
```r
distrib_score_mc=ScoreDistribEmpiric(2,3,10000,T)

distrib_score_theo=ScoreDistribElisa(2,3,T)


length(distrib_score_mc[,2])
```

```
## [1] 47
```

```r
length(distrib_score_theo[,2])
```

```
## [1] 81
```

```r
#diff_distrib_score=abs(distrib_score_mc[,2]-distrib_score_theo[,2])

#par(mfrow = c(1,2))
barplot(distrib_score_mc[,2],col="blue",axes=F)
mtext("Distribution des scores via Monte_Carlo",side=1,line=2.5,col="blue")
axis(2, ylim=c(0,10))
par(new = T)
barplot(distrib_score_theo[,2],col="red",axes=F)
mtext("Distribution des scores via la méthode théorique",side=1,line=4,col="red")
```



Local score calculation

```
LocalScoreMC <- function(lambda0, lambda1, NbSeq, T, X_seq, P_X, tbe0){
  E = ComputeE(lambda0, lambda1)

  pvalue = c()
  X = c()

  min_X = min(X_seq)
  max_X = max(X_seq)

  for (i in 1:NbSeq){
      x = floor(E*log(dexp(tbe0[[i]], rate = lambda1)/dexp(tbe0[[i]], rate = lambda0)))
      X = c(X,x)
      LS = localScoreC(x)$localScore[1]

      daudin_result = daudin(localScore = LS, score_probabilities = P_X, sequence_length = length(x), s
      options(warn = -1) # Disable warnings print

      pvalue = c(pvalue, daudin_result)
  }
  LS_H0=data.frame(num=1:NbSeq, pvalue_scan=pvalue, class=(pvalue<0.05))
  return(LS_H0)
}
```

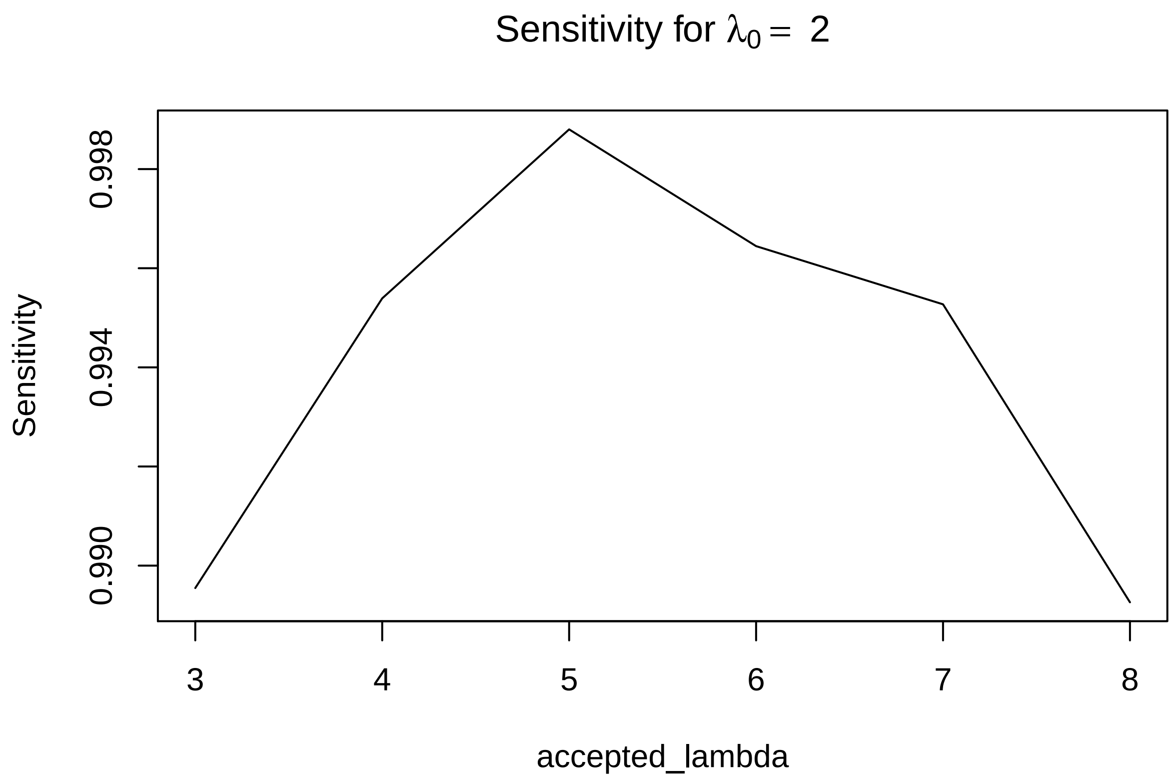
4. Experience plan for comparaison

```
NbSeq = 10**3
T = 10
for (lambda0 in (2:5)){
  Sensitivity = c()
  Specificity = c()
  accepted_lambda = c()

  for (lambda1 in c(3:8)){
    if (lambda0 < lambda1){
      accepted_lambda=c(accepted_lambda,lambda1)
      cat("For T = ", T, ", Nb = ", NbSeq, ", lambda0 = ", lambda0, " and lambda1 = ", lambda1, ":\n", s
      tbe0=vector("list",length=NbSeq)
      pp0 =  vector("list", length = NbSeq)
      for (i in (1:NbSeq)) {
        ppi = PoissonProcess(lambda0,T)
        ni=length(ppi)
        pp0[[i]] = ppi
        tbei=ppi[2:ni]-ppi[1:ni-1]
        tbe0[[i]]=tbei
        }

      #cat("- Empiric version:\n")
      Score = ScoreDistribEmpiric(lambda0, lambda1, NbSeq, T)
      Emp = EmpDistrib(lambda0,n_sample,T,tau)

      X_seq = Score$Score_X
      P_X = Score$P_X

      LS_H0 = LocalScoreMC(lambda0, lambda1, NbSeq, T, X_seq, P_X, tbe0)
      options(warn = -1) # Disable warnings print
      SS_H0 = ScanStatMC(NbSeq, T, tau, Emp, pp0)

      #cat("Local Score:\n")
      #print(summary(LS_H0))
      #cat("Scan Statistics:\n")
      #print(summary(SS_H0))
      #cat("Confusion Matrix:\n")
      #print(confusionMatrix(factor(LS_H0$class), factor(SS_H0$class)))

      #cat("- Elisa version:\n")
      Score = ScoreDistribElisa(lambda0, lambda1, T)
      Emp = EmpDistrib(lambda0,n_sample,T,tau)

      X_seq = Score$Score_X
      P_X = Score$P_X

      LS_H0 = LocalScoreMC(lambda0, lambda1, NbSeq, T, X_seq, P_X, tbe0)
      options(warn = -1) # Disable warnings print

      SS_H0 = ScanStatMC(NbSeq, T, tau, Emp, pp0)

      #cat("Local Score:\n")
```

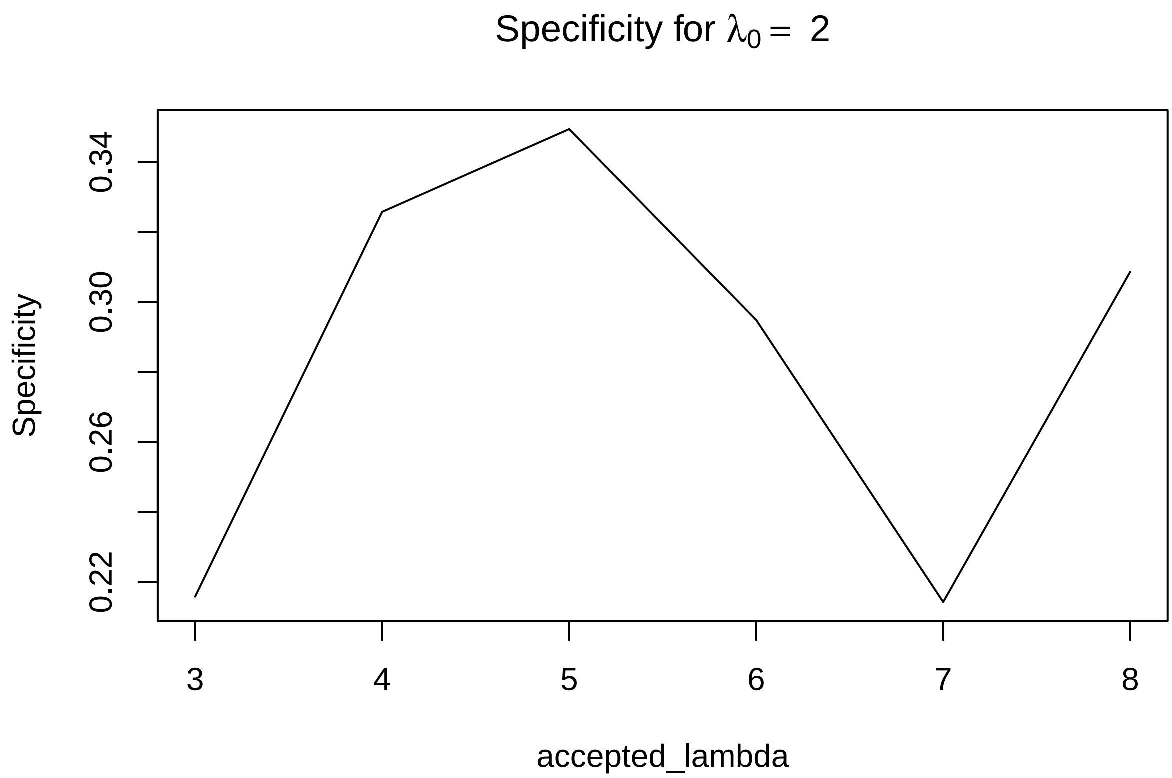
```r
      #print(summary(LS_H0))
      #cat("Scan Statistics:\n")
      #print(summary(SS_H0))
      #cat("Confusion Matrix:\n")
      print(confusionMatrix(factor(LS_H0$class), factor(SS_H0$class))$table)
      Sensitivity = c(Sensitivity,confusionMatrix(factor(LS_H0$class), factor(SS_H0$class))$byClass[1])
      Specificity = c(Specificity,confusionMatrix(factor(LS_H0$class), factor(SS_H0$class))$byClass[2])

      cat("---\n")

    }
  }
  titleSens=TeX(paste(r'(Sensitivity for $\lambda_0=$)', lambda0))
  plot(x=accepted_lambda,y=Sensitivity, type='l', main = titleSens)

  titleSpec=TeX(paste(r'(Specificity for $\lambda_0=$)', lambda0))
  plot(x=accepted_lambda,y=Specificity, type='l', main = titleSpec)

}
```

```
## For T = 10, Nb = 1000, lambda0 = 2 and lambda1 = 3:
##           Reference
## Prediction FALSE TRUE
##      FALSE   852  109
##      TRUE      9   30
## ---
## For T = 10, Nb = 1000, lambda0 = 2 and lambda1 = 4:
##           Reference
## Prediction FALSE TRUE
##      FALSE   864   89
##      TRUE      4   43
## ---
## For T = 10, Nb = 1000, lambda0 = 2 and lambda1 = 5:
##           Reference
## Prediction FALSE TRUE
##      FALSE   833  108
##      TRUE      1   58
## ---
## For T = 10, Nb = 1000, lambda0 = 2 and lambda1 = 6:
##           Reference
## Prediction FALSE TRUE
##      FALSE   841  110
##      TRUE      3   46
## ---
## For T = 10, Nb = 1000, lambda0 = 2 and lambda1 = 7:
##           Reference
## Prediction FALSE TRUE
##      FALSE   842  121
##      TRUE      4   33
## ---
## For T = 10, Nb = 1000, lambda0 = 2 and lambda1 = 8:
##           Reference
## Prediction FALSE TRUE
##      FALSE   829  112
```

```
##       TRUE       9   50
## ---
```

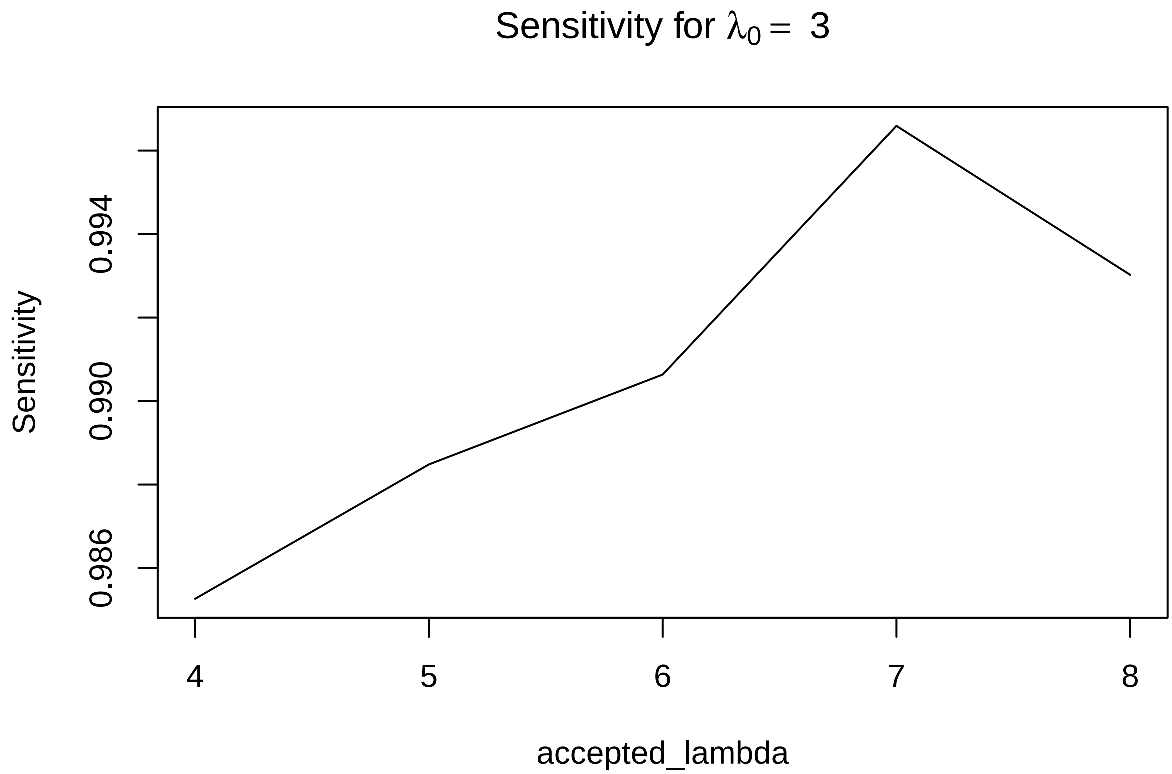


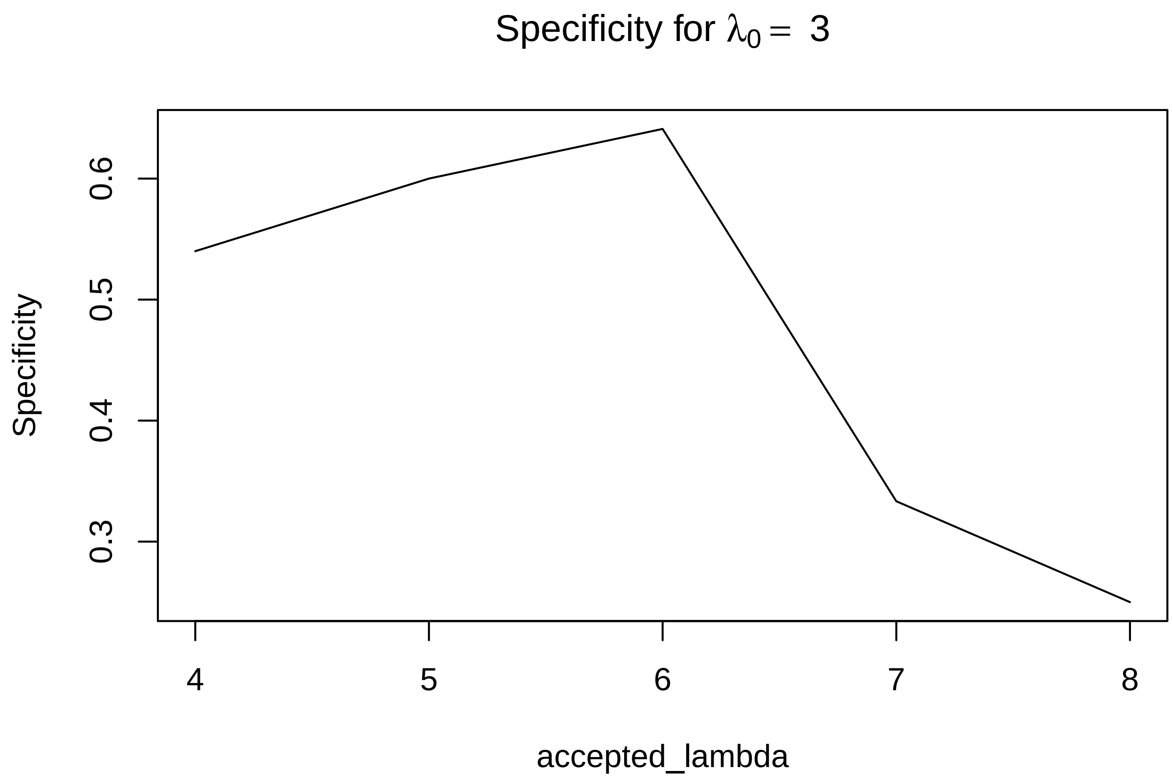
Sensitivity for $\lambda_0 =$ 2
Sensitivity
0.990
0.994
0.998
3
4
5
6
7
8
accepted_lambda



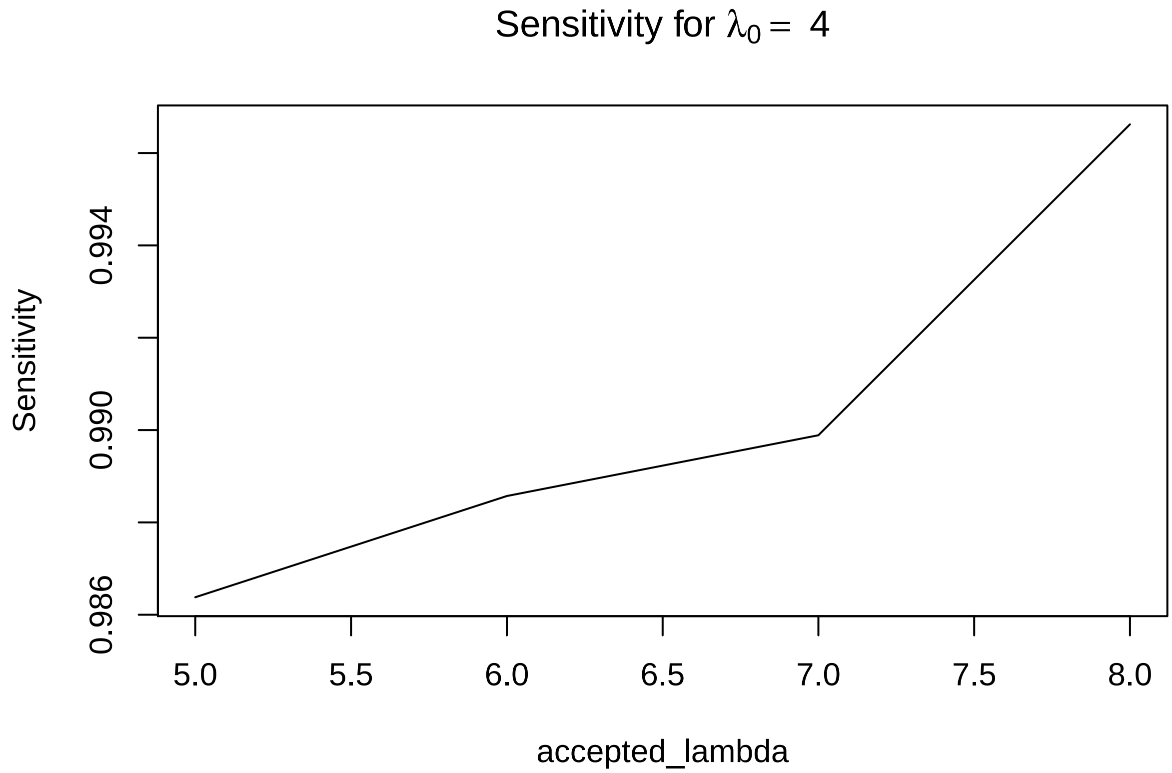

```
## For T = 10, Nb = 1000, lambda0 = 3 and lambda1 = 4:
##           Reference
## Prediction FALSE TRUE
##      FALSE   936   23
##      TRUE     14   27
## ---
## For T = 10, Nb = 1000, lambda0 = 3 and lambda1 = 5:
##           Reference
## Prediction FALSE TRUE
##      FALSE   944   18
##      TRUE     11   27
## ---
## For T = 10, Nb = 1000, lambda0 = 3 and lambda1 = 6:
##           Reference
## Prediction FALSE TRUE
##      FALSE   952   14
##      TRUE      9   25
## ---
## For T = 10, Nb = 1000, lambda0 = 3 and lambda1 = 7:
##           Reference
## Prediction FALSE TRUE
##      FALSE   877   80
##      TRUE      3   40
## ---
## For T = 10, Nb = 1000, lambda0 = 3 and lambda1 = 8:
##           Reference
```

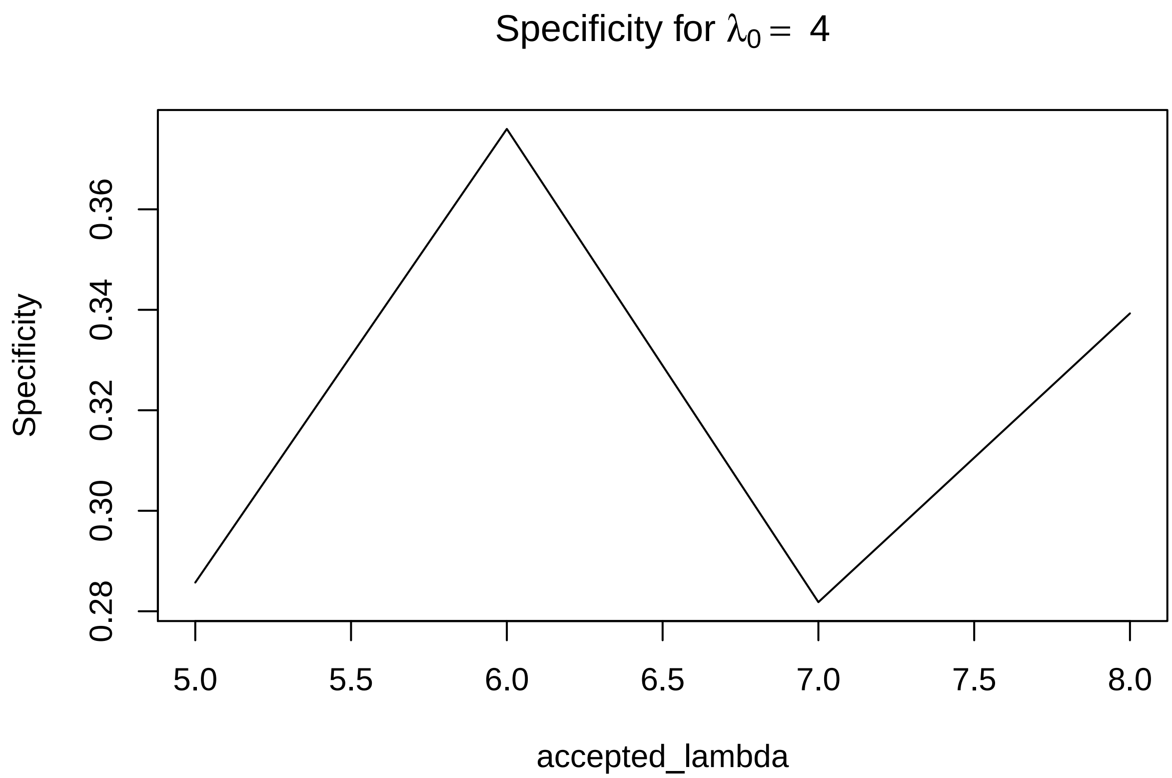
```
## Prediction FALSE TRUE
##      FALSE   854  105
##      TRUE      6   35
## ---
```



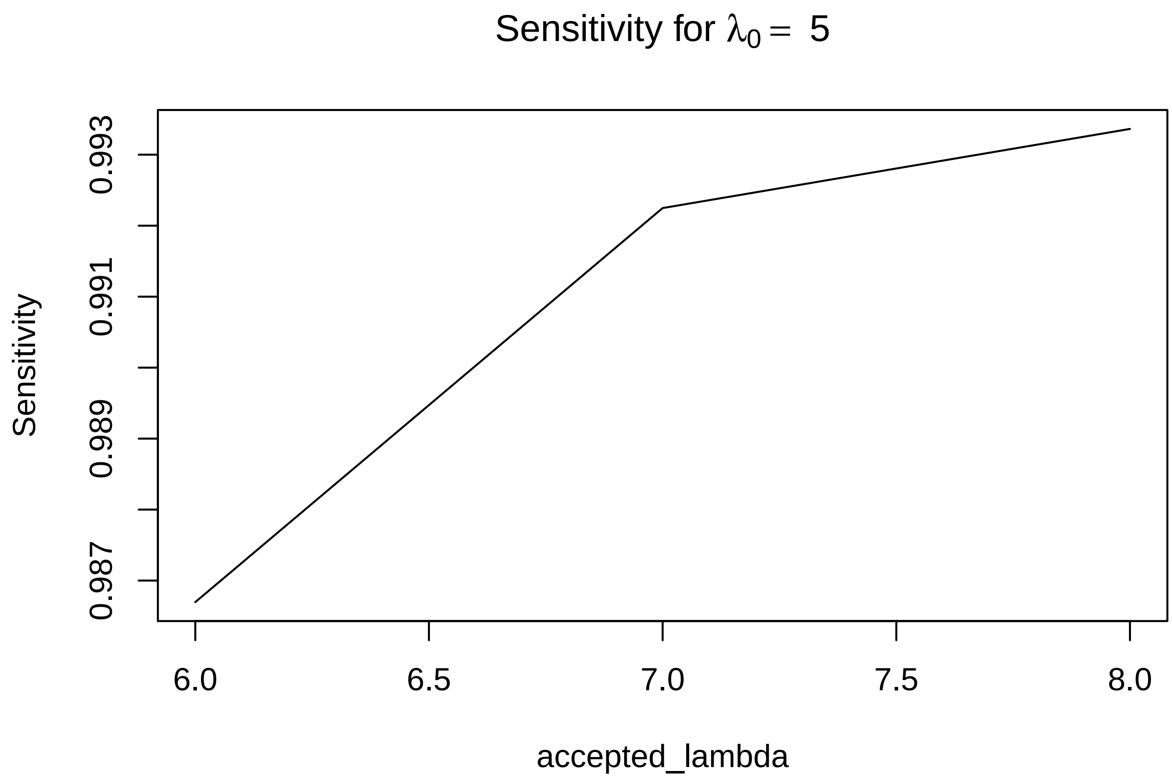


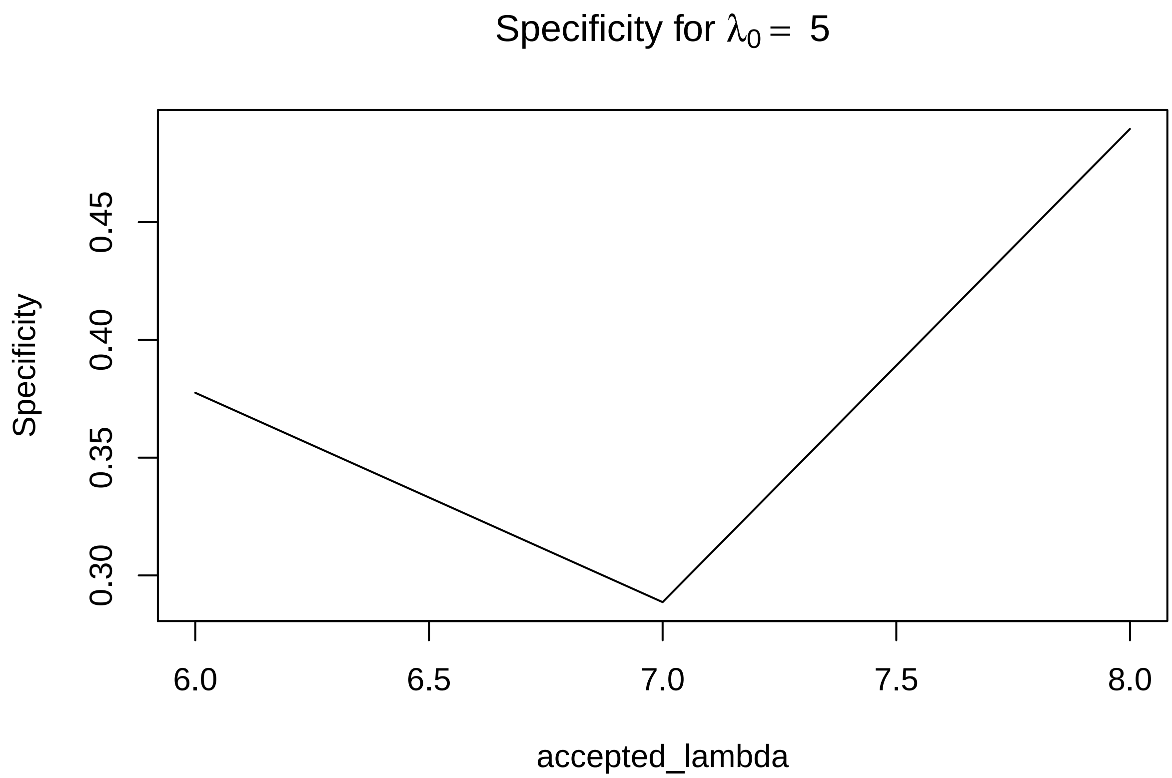

```
## For T = 10, Nb = 1000, lambda0 = 4 and lambda1 = 5:
##           Reference
## Prediction FALSE TRUE
##      FALSE   869   85
##      TRUE     12   34
## ---
## For T = 10, Nb = 1000, lambda0 = 4 and lambda1 = 6:
##           Reference
## Prediction FALSE TRUE
##      FALSE   865   78
##      TRUE     10   47
## ---
## For T = 10, Nb = 1000, lambda0 = 4 and lambda1 = 7:
##           Reference
## Prediction FALSE TRUE
##      FALSE   881   79
##      TRUE      9   31
## ---
## For T = 10, Nb = 1000, lambda0 = 4 and lambda1 = 8:
##           Reference
## Prediction FALSE TRUE
##      FALSE   885   74
##      TRUE      3   38
## ---
```


Sensitivity for $\lambda_0 = 4$
Sensitivity
0.986
0.990
0.994
5.0
5.5
6.0
6.5
7.0
7.5
8.0
accepted_lambda




```
## For T = 10, Nb = 1000, lambda0 = 5 and lambda1 = 6:
##           Reference
## Prediction FALSE TRUE
##      FALSE   890   61
##      TRUE     12   37
## ---
## For T = 10, Nb = 1000, lambda0 = 5 and lambda1 = 7:
##           Reference
## Prediction FALSE TRUE
##      FALSE   896   69
##      TRUE      7   28
## ---
## For T = 10, Nb = 1000, lambda0 = 5 and lambda1 = 8:
##           Reference
## Prediction FALSE TRUE
##      FALSE   898   49
##      TRUE      6   47
## ---
```


Sensitivity for λ0 = 5
Sensitivity
0.987
0.989
0.991
0.993
6.0
6.5
7.0
7.5
8.0
accepted_lambda



Specificity for $\lambda_0 =$ 5
Specificity
0.30
0.35
0.40
0.45
6.0
6.5
7.0
7.5
8.0
accepted_lambda