440

```r
MC.CI = function(delta){
  n = length(delta)
  se = sqrt(1/n)*sd(delta)

  list(mean=mean(delta), CI=mean(delta)+c(-1,1)*se*1.96, se=se)
}
```

1 simple monte carlo

```r
M=1000
x = matrix(rpois(25*M, 2), nrow = M, ncol = 25)
xbar = rowMeans(x)
deltax = (xbar-2)/(sqrt(2/25)) >= 1.645
# estimate of type 1 error rate
MC.CI(deltax)
```

```
## $mean
## [1] 0.061
##
## $CI
## [1] 0.04615873 0.07584127
##
## $se
## [1] 0.007572076
```

comment: easy to implement, but having the largest standard error and not very accurate

importance sampling

```r
x = matrix(rpois(25*M, 2.4653), nrow = M, ncol = 25)
xbar = rowMeans(x)
deltax = (xbar-2)/(sqrt(2/25)) >= 1.645
dfx = dpois(x,2)
dgx = dpois(x,2.4653)
fx=rep(1,M)
gx=rep(1,M)
for (j in 1:M){
  for(k in 1:25){
    fx[j]=dfx[j,k]*fx[j]
    gx[j]=dgx[j,k]*gx[j]
  }
}
deltax = (deltax*fx/gx)
MC.CI(deltax)
```

```
## $mean
## [1] 0.05462187
##
## $CI
```

```
## [1] 0.04957082 0.05967292
##
## $se
## [1] 0.002577068
```

comment: smallest standard error, useful when it's hard to sample nontrivial delta(x) from distribution of f(x), seems to overestimate the result for a bit

antithetic

```r
hM = round(M/2)
x = matrix(runif(25*hM), nrow = hM, ncol = 25)
x1 = qpois(x,2)
x2 = qpois(1-x,2)

x1bar = rowMeans(x1)
x2bar = rowMeans(x2)

deltax1 = (x1bar-2)/(sqrt(2/25)) >= 1.645
deltax2 = (x2bar-2)/(sqrt(2/25)) >= 1.645

deltax = (deltax1+deltax2)/2

MC.CI(deltax)
```

```
## $mean
## [1] 0.048
##
## $CI
## [1] 0.03507604 0.06092396
##
## $se
## [1] 0.006593858
```

merit: easy to implement (given when can generate x from its quantile), successfully decreased standard error by certain amount. overall, seems to produce accurate result (close to 0.05)

control variate

```r
x = matrix(rpois(25*M, 2), nrow = M, ncol = 25)
xbar = rowMeans(x)
deltax = (xbar-2)/(sqrt(2/25)) >= 1.645

gx = rowSums(x)


cov.hat = sum((deltax-mean(deltax))*(gx-mean(gx))/(M*(M-1)))
var.hat = sum((gx-mean(gx))^2)/(M*(M-1))
vardelta.hat = sum((deltax-mean(deltax))^2)/(M*(M-1))

alpha = -cov.hat/var.hat

real.star=50

delta.cv = deltax + alpha*(gx-real.star)

var.cv = vardelta.hat+alpha^2*var.hat+2*cov.hat*alpha
```

```r
theta.cv = mean(delta.cv)

theta.cv
```

```
## [1] 0.0620431
```

```r
se = sqrt(var.cv)
se
```

```
## [1] 0.006589048
```

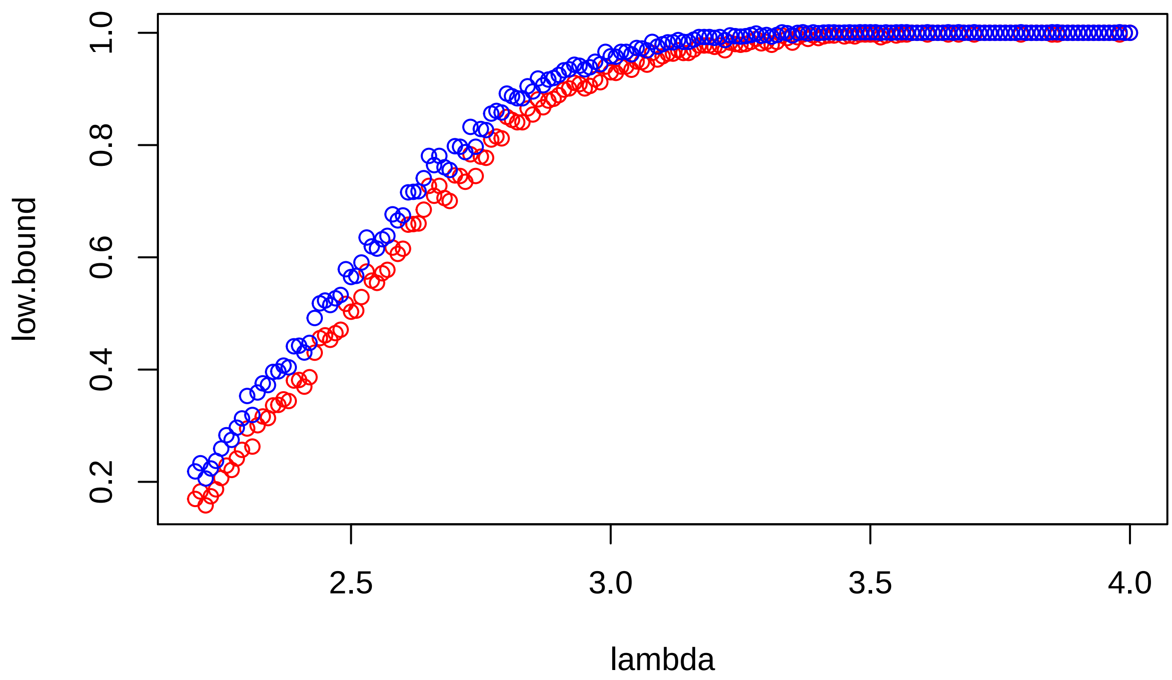
```r
theta.cv+c(1,-1)*se*1.96
```

```
## [1] 0.07495763 0.04912857
```

effect of variance reduction is slightly better than antithetic sampling, most complicated to implement in these four cases.

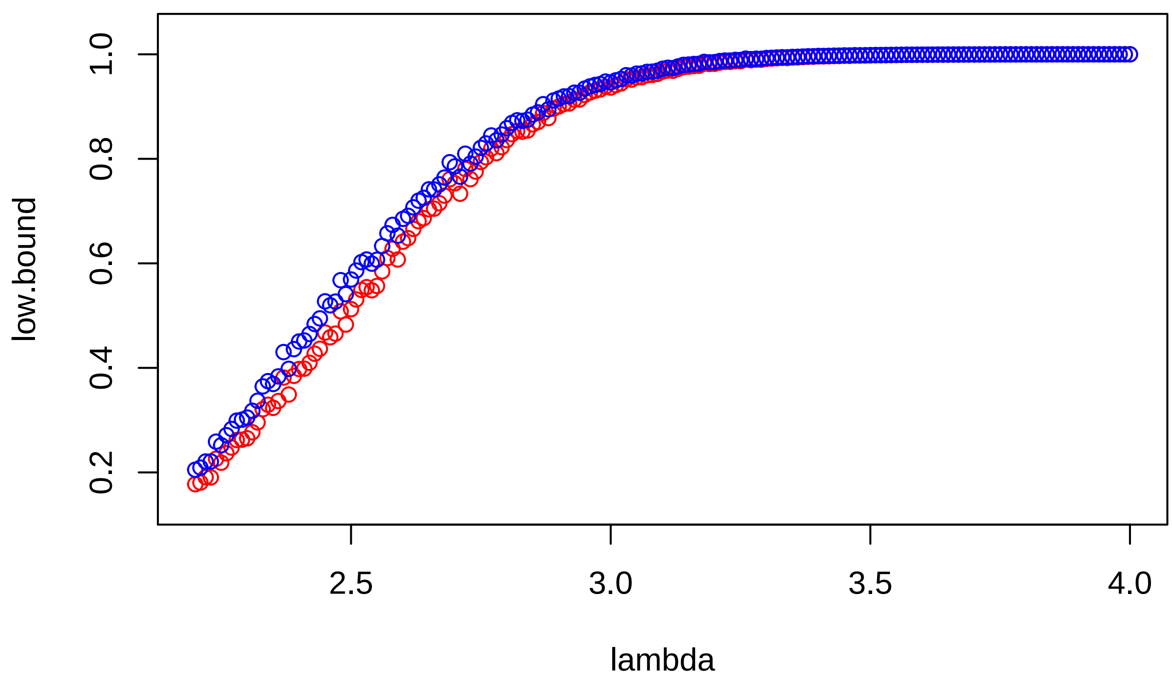
b

```r
# simple MC
lambda = seq(2.2,4,0.01)
low.bound=c()
up.bound=c()

for (i in lambda){
  x = matrix(rpois(25*M, i), nrow = M, ncol = 25)
  xbar = rowMeans(x)
  deltax = (xbar-2)/(sqrt(2/25)) >= 1.645
  low.bound=c(low.bound, MC.CI(deltax)$CI[1])
  up.bound=c(up.bound, MC.CI(deltax)$CI[2])
}

plot(lambda, low.bound,  col='red')
points(lambda, up.bound, col='blue')
```




```
#importance sampling
low.bound=c()
up.bound=c()
for (i in lambda){
  newlambda = 2.4653
  x = matrix(rpois(25*M, newlambda), nrow = M, ncol = 25)
  xbar = rowMeans(x)
  fx = rep(0,nrow(x))
  gx = rep(0,nrow(x))
  # if lambda>2.4653, we cannot sample acceptation well with f(x)
  if(i>2.4653){
    deltax = (xbar-2)/(sqrt(2/25)) <= 1.645
    dfx=dpois(x ,i)
    for (j in 1:nrow(x)){
      fx[j]=prod(dfx[j,])
    }
    dgx=dpois(x , newlambda)
    for (j in 1:nrow(x)){
      gx[j]=prod(dgx[j,])
    }
    ipx = deltax*fx/gx
    ipx = 1-ipx
  }else{
  # if lambda<2.4653, we cannot sample rejection well with f(x)
    deltax = (xbar-2)/(sqrt(2/25)) >= 1.645
    dfx=dpois(x ,i)
```

```
    for (j in 1:nrow(x)){
      fx[j]=prod(dfx[j,])
    }
    dgx=dpois(x , newlambda)
    for (j in 1:nrow(x)){
      gx[j]=prod(dgx[j,])
    }
    ipx = deltax*fx/gx
  }
  low.bound=c(low.bound, MC.CI(ipx)$CI[1])
  up.bound=c(up.bound, MC.CI(ipx)$CI[2])
}
plot(lambda, low.bound, col='red', ylim = extendrange(c(low.bound,up.bound)))
points(lambda, up.bound, col='blue')
```

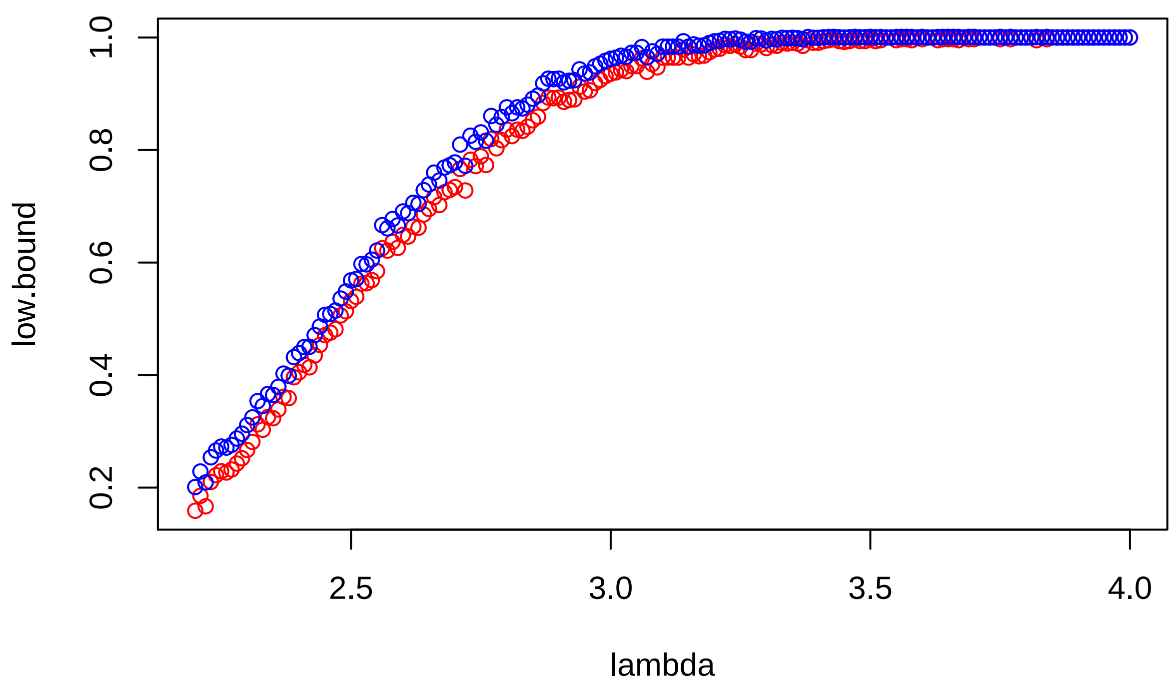


again simple montecarlo is easiest to implement, but yields largest CI and roughest curve. But it works.

importance sampling yields smoothest curve, and shortest confidence interval especially when lambda > 2.5, when it's hard to sample delta(x)

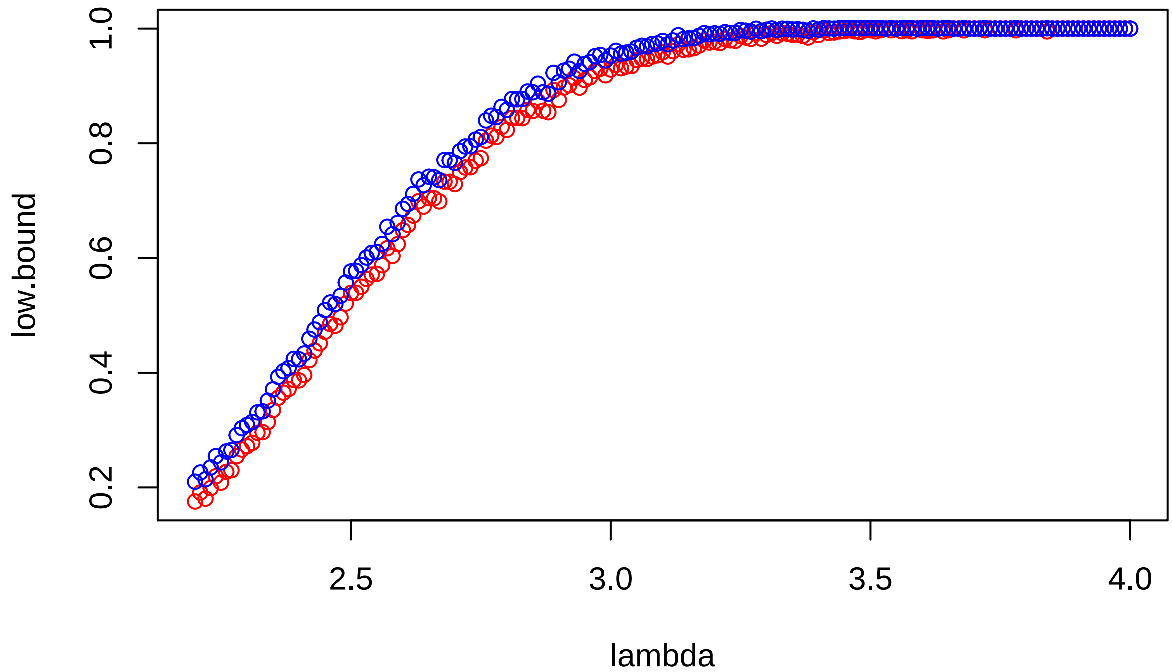
```
# antithetic
low.bound=c()
up.bound=c()

for (i in lambda){
  hM = round(M/2)
  x = matrix(runif(25*hM), nrow = hM, ncol = 25)
```

```
  x1 = qpois(x,i)
  x2 = qpois(1-x,i)
  x1bar = rowMeans(x1)
  x2bar = rowMeans(x2)

  deltax1 = (x1bar-2)/(sqrt(2/25)) >= 1.645
  deltax2 = (x2bar-2)/(sqrt(2/25)) >= 1.645

  deltax = (deltax1+deltax2)/2

  #concatenate 2 groups of data
  low.bound=c(low.bound, MC.CI(deltax)$CI[1])
  up.bound=c(up.bound, MC.CI(deltax)$CI[2])
}

plot(lambda, low.bound, col='red')
points(lambda, up.bound, col='blue')
```




```
# control variate
low.bound=c()
up.bound=c()

for (i in lambda){
  x = matrix(rpois(25*M, i), nrow = M, ncol = 25)
  xbar = rowMeans(x)
  deltax = (xbar-2)/(sqrt(2/25)) >= 1.645
```

```r
  gx = rowSums(x)


  cov.hat = sum((deltax-mean(deltax))*(gx-mean(gx))/(M*(M-1)))
  var.hat = sum((gx-mean(gx))^2)/(M*(M-1))
  vardelta.hat = sum((deltax-mean(deltax))^2)/(M*(M-1))

  alpha = -cov.hat/var.hat

  real.star=25*i

  delta.cv = deltax + alpha*(gx-real.star)

  var.cv = vardelta.hat+alpha^2*var.hat+2*cov.hat*alpha

  theta.cv = mean(delta.cv)

  theta.cv
  CIs = theta.cv+c(-1,1)*sqrt(var.cv)*1.96

  low.bound=c(low.bound, CIs[1])
  up.bound=c(up.bound, CIs[2])
}

plot(lambda, low.bound, col='red')
points(lambda, up.bound, col='blue')
```




```
###
```

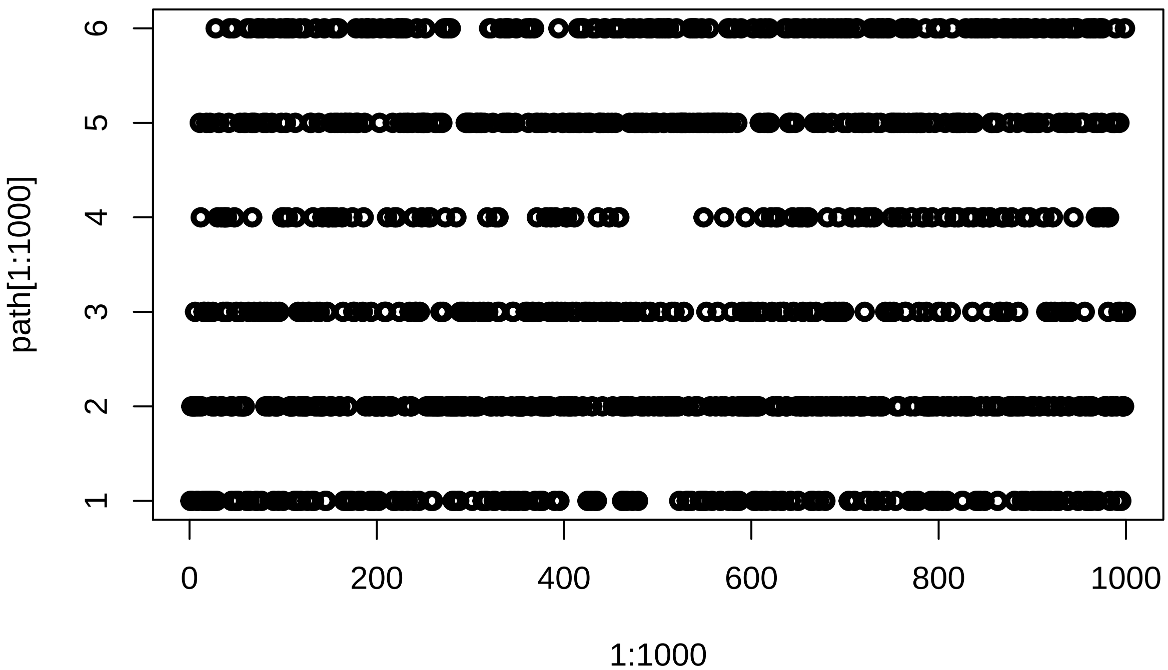
Control variance has slightly smaller CI and smoother curve than antithetic. They work better than simple montecarlo, worse than importance sampling when lambda is large. When lambda is small, they work better than importance sampling.

2

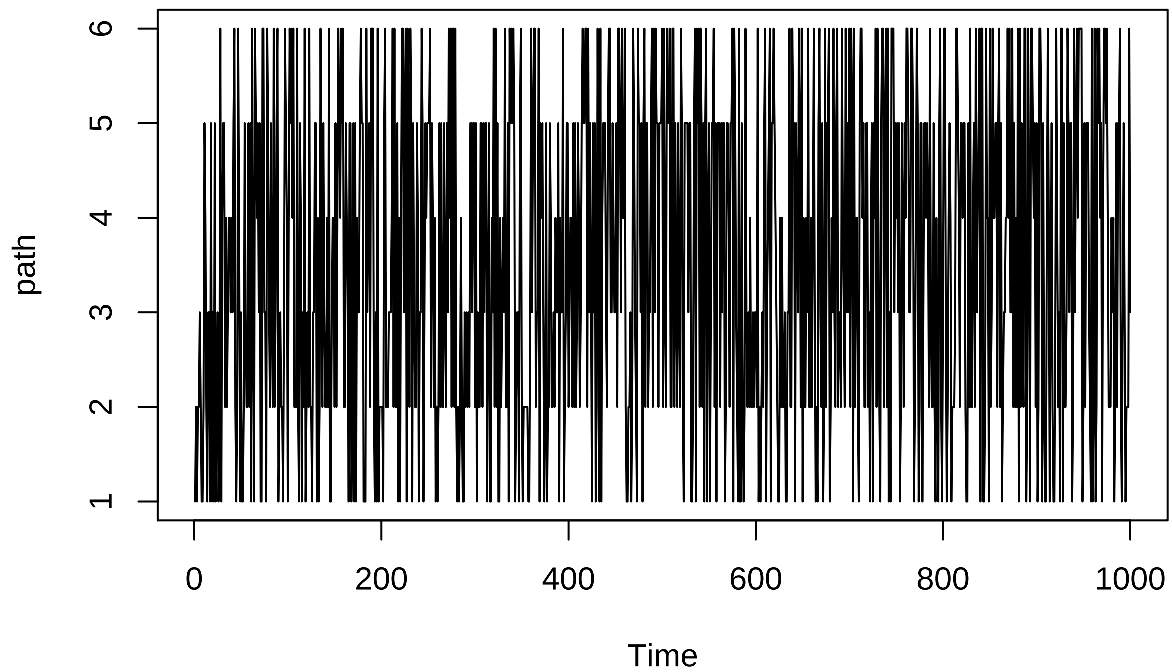
```r
P=diag(c(0.5,0.5,0.25,0.25,0.5,0.5))
P[1,2]=P[6,5]=0.5
P[2,1]=P[2,3]=P[5,4]=P[5,6]= 0.25
P[3,1]=P[3,2]=P[3,4]=P[4,6]=P[4,5]=P[4,3]=0.25
P
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.50 0.50 0.00 0.00 0.00 0.00
## [2,] 0.25 0.50 0.25 0.00 0.00 0.00
## [3,] 0.25 0.25 0.25 0.25 0.00 0.00
## [4,] 0.00 0.00 0.25 0.25 0.25 0.25
## [5,] 0.00 0.00 0.00 0.25 0.50 0.25
## [6,] 0.00 0.00 0.00 0.00 0.50 0.50
```

```r
getrand.state=function(x){
  ret=-1
  r=runif(1)
  temp=0
  for(sta in 1:length(x)){
    temp=temp+x[sta]
    if (r <= temp){
      ret = sta
      break
    }
  }
  ret
}

x=c(1,rep(0,5))
path=rep(0,1000)
for (i in 1:1000){
  path[i]=getrand.state(x)
  x= t(P) %*% x
}

plot(1:1000, path[1:1000] ,lwd=3)
```




```r
ts.plot(path)
```




```r
#relative frequency
table(path)/1000
```

```
## path
##     1     2     3     4     5     6
## 0.152 0.226 0.141 0.096 0.212 0.173
```

```r
# we guess stationary distribution is c(1/6,2/9,1/9,1/9,2/9,1/6)

t(P)%*%c(1/6,2/9,1/9,1/9,2/9,1/6)
```

```
##           [,1]
## [1,] 0.1666667
## [2,] 0.2222222
## [3,] 0.1111111
## [4,] 0.1111111
## [5,] 0.2222222
## [6,] 0.1666667
```