a2q56

```
getalpha= function(z,znew, inversetheta1=1.5, inversetheta2=2){
  alpha = znew^(-3/2)*exp(-inversetheta1*znew -inversetheta2/znew)/
    (z^(-3/2)*exp(-inversetheta1*z -inversetheta2/z))
  return(alpha)
}

getalpha2 = function(z,znew, inversetheta1, inversetheta2){
  alpha = znew^(-3/2)*exp(-inversetheta1*znew -inversetheta2/znew+2*sqrt(inversetheta1*inversetheta2)+ l
    (z^(-3/2)*exp(-inversetheta1*z -inversetheta2/z+2*sqrt(inversetheta1*inversetheta2)+ log(sqrt(2*thet
  return(alpha)
}

ind.met = function(gshape, grate){
    M=1000
    x = rep(1,M)
    theta1=1.5
    theta2=2
    for (i in 1:(M-1)){
      xnew = rgamma(1,gshape,grate)
      alpha = getalpha(x[i],xnew, theta1,theta2)*dgamma(x[i],gshape, grate)/dgamma(xnew,gshape, grate)
      accept = runif(1)<alpha
      x[i+1] = x[i] * (1-accept )+ xnew*(accept)
    }
    return(list(mean=mean(x), meaninv=mean(1/x), var=var(x) ))
}


rand.walk = function(){
    M=1000
    x = rep(0,M)
    theta1=1.5
    theta2=2
    # position in our normal random walk
    posnorm = 0
    for (i in 1:(M-1)){
      randnorm = rnorm(1)
      newpos = posnorm + randnorm
      accept.rate = exp(newpos-posnorm)*getalpha(exp(posnorm), exp(newpos))
      accept = runif(1)<accept.rate
      x[i+1] = (1-accept)* x[i] + accept * newpos
    }
    x = x[10:M]
    x = exp(x)
    return(list(data=x, mean=mean(x), meaninv=mean(1/x), var=var(x) ))
}
```

```r
theta1= 1.5
theta2= 2

# E(Z)
sqrt(theta2/theta1)
```

```
## [1] 1.154701
```

```r
# E(1/Z)
sqrt(theta1/theta2)+1/(2*theta2)
```

```
## [1] 1.116025
```

```r
ind.met(1,1)
```

```
## $mean
## [1] 1.170875
##
## $meaninv
## [1] 1.098203
##
## $var
## [1] 0.414498
```

```r
ind.met(10,10)
```

```
## $mean
## [1] 1.213924
##
## $meaninv
## [1] 1.046794
##
## $var
## [1] 0.364579
```

```r
ind.met(10,20)
```

```
## $mean
## [1] 0.7275384
##
## $meaninv
## [1] 1.519805
##
## $var
## [1] 0.0429998
```

```r
ind.met(10,20)
```

```
## $mean
## [1] 0.7644551
##
## $meaninv
## [1] 1.436873
##
## $var
## [1] 0.04195873
```

```r
ind.met(10,20)
```

```
## $mean
## [1] 0.8719903
##
## $meaninv
## [1] 1.279675
##
## $var
## [1] 0.05999467
```

```r
ind.met(20,10)
```

```
## $mean
## [1] 1.041517
##
## $meaninv
## [1] 1.061639
##
## $var
## [1] 0.2113589
```

```r
ind.met(20,10)
```

```
## $mean
## [1] 1.560813
##
## $meaninv
## [1] 0.7062689
##
## $var
## [1] 0.2748112
```

```r
ind.met(20,10)
```

```
## $mean
## [1] 1.361345
##
## $meaninv
## [1] 0.8060194
##
## $var
## [1] 0.2308663
```

```r
ind.met(10,1)
```

```
## $mean
## [1] 1
##
## $meaninv
## [1] 1
##
## $var
## [1] 0
```

```r
ind.met(1,10)
```

```
## $mean
```

```
## [1] 1
##
## $meaninv
## [1] 1
##
## $var
## [1] 0
```

```r
r = rand.walk()
r$mean
```

```
## [1] 1.132289
```

```r
r$meaninv
```

```
## [1] 1.09331
```

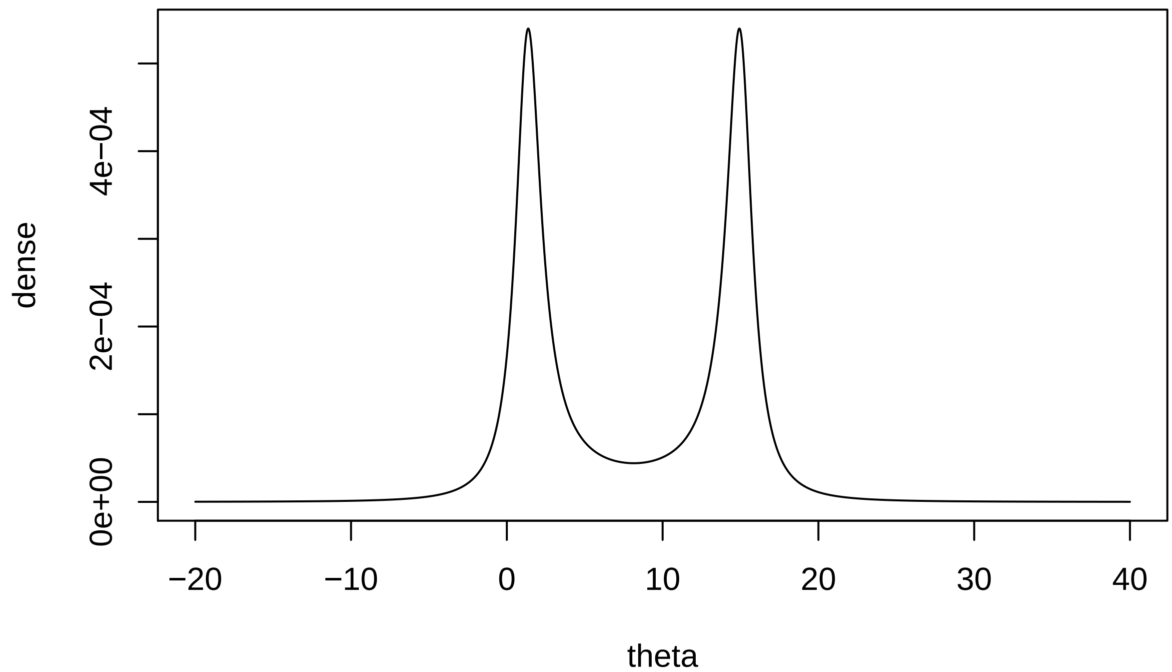
we observe for initial value of gammarate and gammashape, if the resulting expectation is far from the starting value x0, we will keep rejecting it until a miracle. so it might be needed to initialze our x0 based on our gamma parameters.

also, we tend to overestimate mean and underestimate inversemean if shape>rate, and underestimate mean and overestimate inversemean if shape<rate

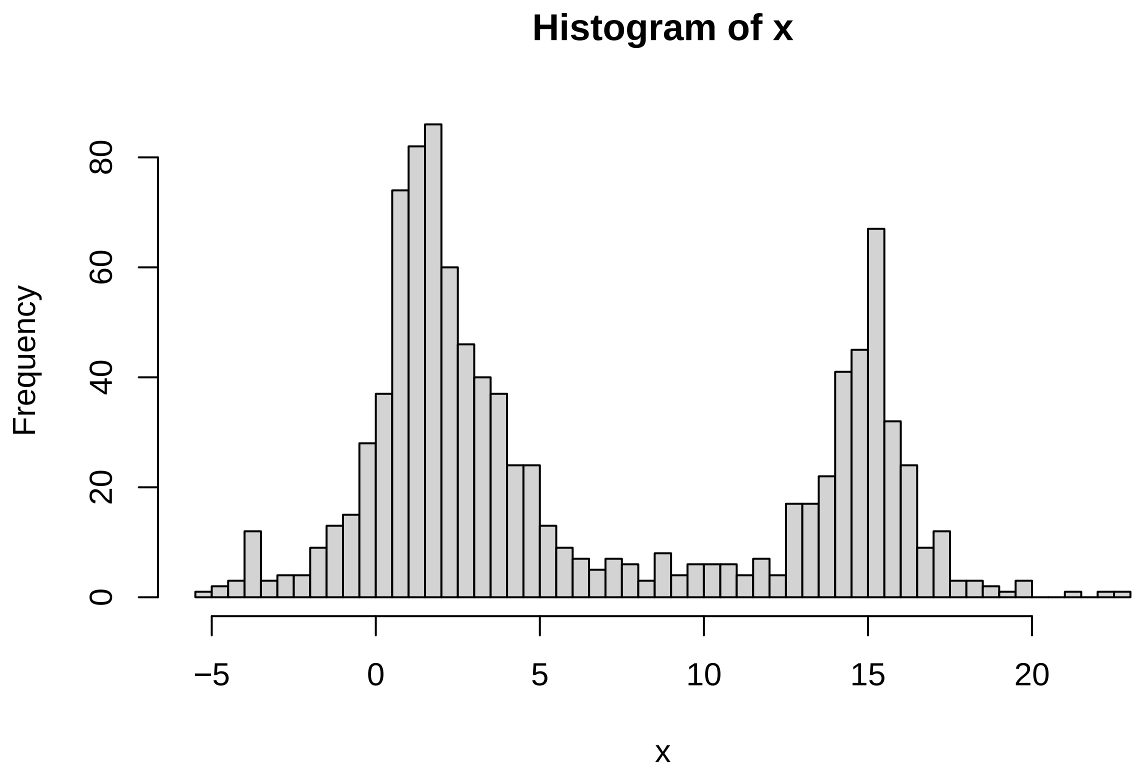
when rate=shape, the approximation is relatively more accurate.

the variance of our approximation is smallest when shape<rate.

```r
theta = seq(-20,40,0.01)
n=length(theta)
dense = rep(0, n)

for (i in 1:n){
  dense[i] = dcauchy(1.3,theta[i],1)*dcauchy(15,theta[i],1)
}
plot(theta, dense, type = 'l')
```




```r
M=1000
x=rep(1,M)
metro = function(){
  for(i in 1:(M-1)){
    newx = rcauchy(1,x[i],1)
    ratio = dcauchy(1.3,newx,1)/dcauchy(1.3,x[i],1)*dcauchy(15,newx,1)/dcauchy(15,x[i],1)
    if (ratio >= runif(1)){
      x[i+1] = newx
    }else{
      x[i+1] = x[i]
    }
  }
  x
}

x=metro()
hist(x,breaks=50)
```




```r
#simulated tempering
gettemp = function(oldt){
  oldt = round(oldt, 2)
  if (oldt == 0.1){
    return(0.2)
  }else if(oldt==1){
    return(0.9)
  }else{
    if (runif(1)>0.5){
      return(oldt+0.1)
    }else{
      return(oldt-0.1)
    }
  }
}


M=1000
# transitional matrix
m = matrix(0, nrow = 10, ncol = 10)
for (i in 2:9){
  m[i,i+1]=m[i,i-1]=0.5
}
m[1,2]=m[10,9]=1
m
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
```

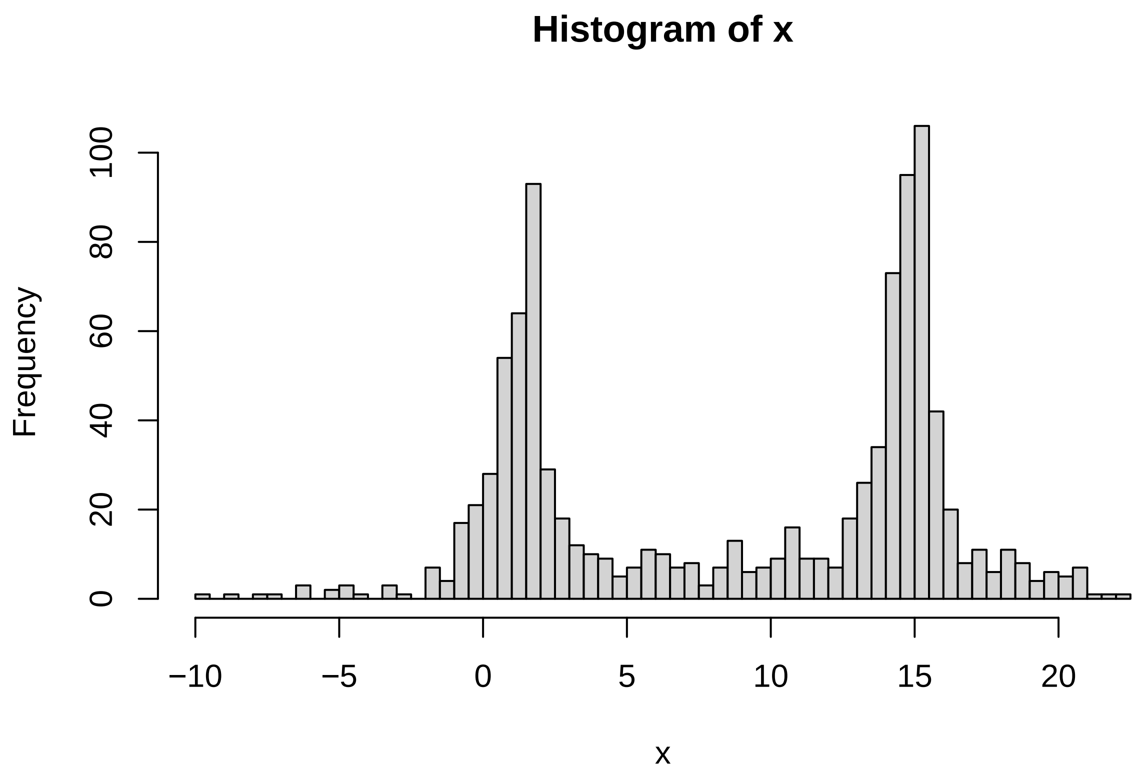
```
##  [1,]  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0
##  [2,]  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0  0.0   0.0
##  [3,]  0.0  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0   0.0
##  [4,]  0.0  0.0  0.5  0.0  0.5  0.0  0.0  0.0  0.0   0.0
##  [5,]  0.0  0.0  0.0  0.5  0.0  0.5  0.0  0.0  0.0   0.0
##  [6,]  0.0  0.0  0.0  0.0  0.5  0.0  0.5  0.0  0.0   0.0
##  [7,]  0.0  0.0  0.0  0.0  0.0  0.5  0.0  0.5  0.0   0.0
##  [8,]  0.0  0.0  0.0  0.0  0.0  0.0  0.5  0.0  0.5   0.0
##  [9,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.5  0.0   0.5
## [10,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0   0.0
```

```r
v = c(1,rep(2,8),1)

#limiting distribution
v=v/sum(v)
t(m) %*% v
```

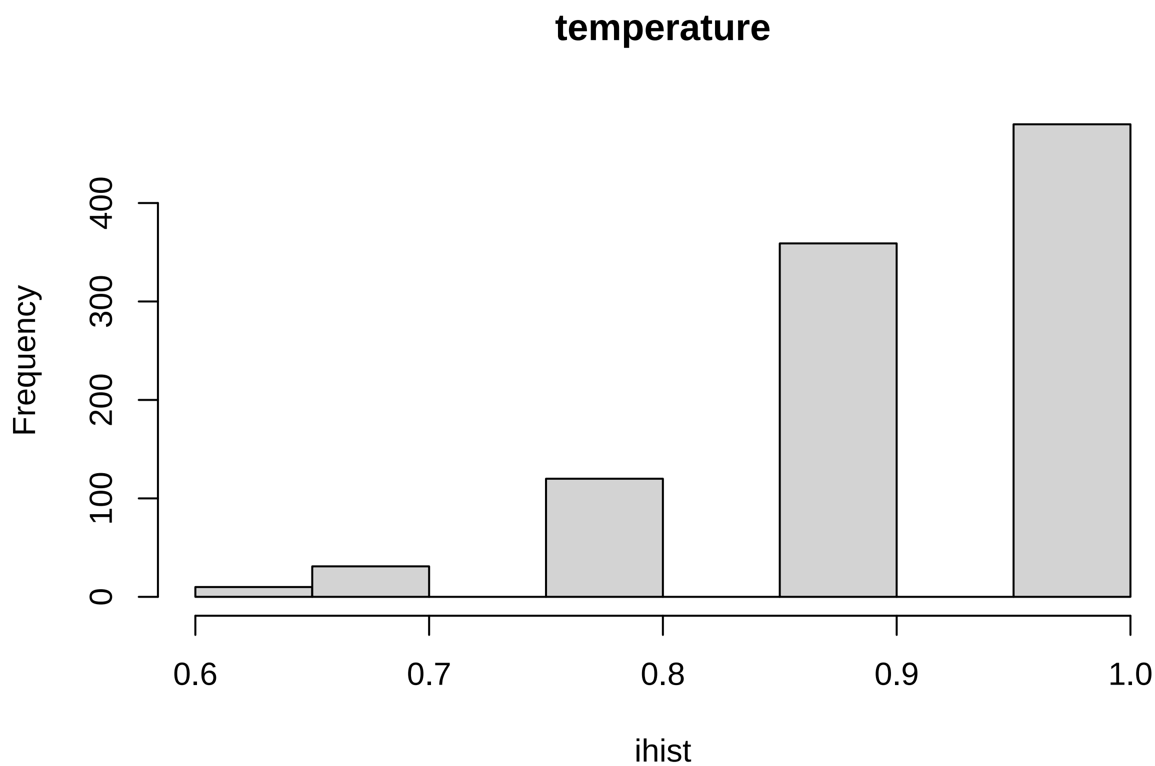
```
##             [,1]
##  [1,] 0.05555556
##  [2,] 0.11111111
##  [3,] 0.11111111
##  [4,] 0.11111111
##  [5,] 0.11111111
##  [6,] 0.11111111
##  [7,] 0.11111111
##  [8,] 0.11111111
##  [9,] 0.11111111
## [10,] 0.05555556
```

```r
# we see p(i)*q(i,j) = p(j) * q(j,i) for i,j in 1:10, |i-j|=1
# as p(1), p(10) are half other  p(i), but q(1,2), q(10,9) are twice q(i,j) for i,j not in {1,10}

# then for our metropolis hasting, r(i,inew,xnew) = pi_inew(xnew)/pi_i(xnew)
# as
M=1000
x=rep(1,M)
i = 1
ihist = rep(1,M)
for (t in 1:(M-1)){
  # use old i to generate new x
  newx = rcauchy(1,x[t],1)
  ratio = (dcauchy(1.3,newx,1)/dcauchy(1.3,x[t],1)*dcauchy(15,newx,1)/dcauchy(15,x[t],1))^(1/i)
  if (ratio > runif(1)){
    x[t+1] = newx
  }else{
    x[t+1] = x[t]
  }
  #generate new i
  newi = gettemp(i)

  # i accept ratio
  ratio = (dcauchy(1.3,x[t+1],1)*dcauchy(15,x[t+1],1))^(1/newi)/
    (dcauchy(15,x[t+1],1)*dcauchy(1.3,x[t+1],1))^(1/i)

  if (ratio>runif(1)){
```

```r
    i = newi
  }
  ihist[t+1]=i

}

hist(x,breaks = 50)
```




```r
hist(ihist, main = 'temperature')
```



we observe while metropolis algorithum tend to let output be trapped in some region (because of the tiny probability to travel from one cluster to another), simulated tempering allow us to go to travel to other cluster. the data in c resembles the original density function more.