two-group model

Mengqi LIn

9/4/2021

Settings

n_g : vector for number of hypothesis in each group. n_g = c(1000,2000)

mu1_g : vector for mu1 in each group. mu1_g = c(1.5, 4)

pi1_g : vector for pi1 in each group. pi1_g = c(0.1, 0.05)

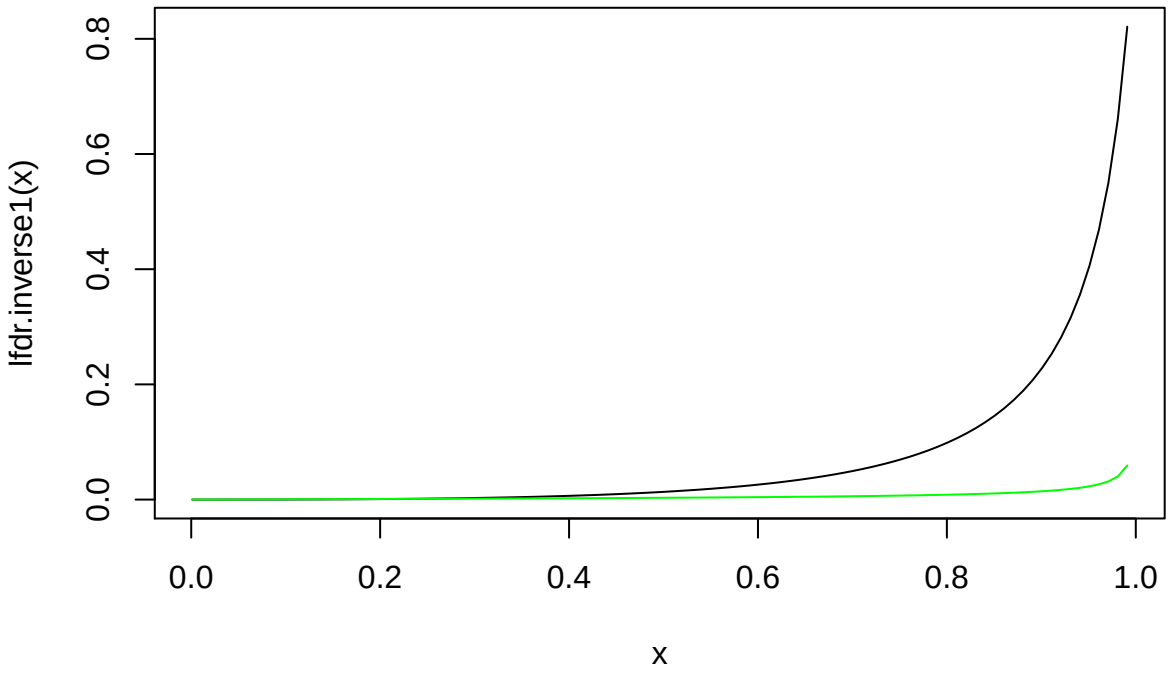
```
n_g = c(1000,2000)
mu1_g = c(1.5, 4)
pi1_g = c(0.1, 0.05)
pi0_g = 1-pi1_g
alpha <- 0.05
```

the lfdr.inverse function in each group

```
## lfdr.inverse function
lfdr.inverse <- function(lfdr, pi0, mu1){
  1 - pnorm(1/mu1*log((pi0/lfdr - pi0)/(1-pi0)) + mu1/2)
}

## lfdr.inverse function in group1
lfdr.inverse1 <- function(lfdr) {
  lfdr.inverse(lfdr, pi0 = pi0_g[1], mu1 = mu1_g[1])
}

## lfdr.inverse function in group2
lfdr.inverse2 <- function(lfdr) {
  lfdr.inverse(lfdr, pi0 = pi0_g[2], mu1 = mu1_g[2])
}
```

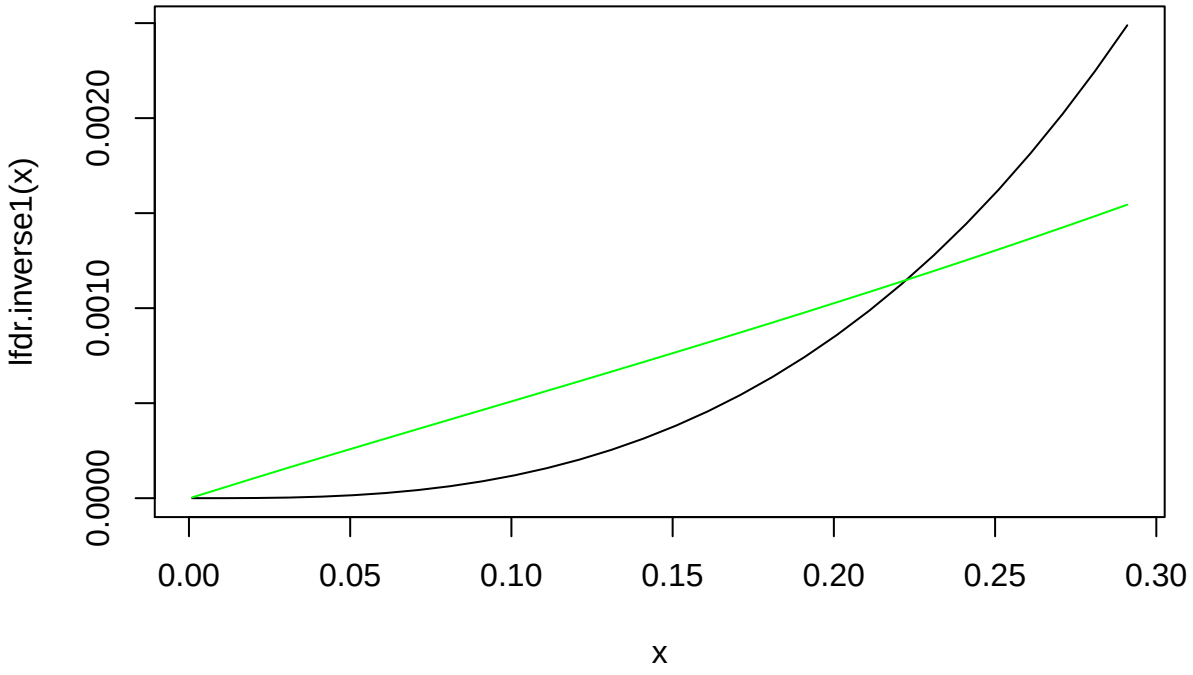
the plots of lfdr.inverse function in each group:

```
x = seq(0.001,0.9999,0.01)
plot(x = x, y = lfdr.inverse1(x), type = "l")
lines(x = x, y = lfdr.inverse2(x), type = "l",col = "green")
```



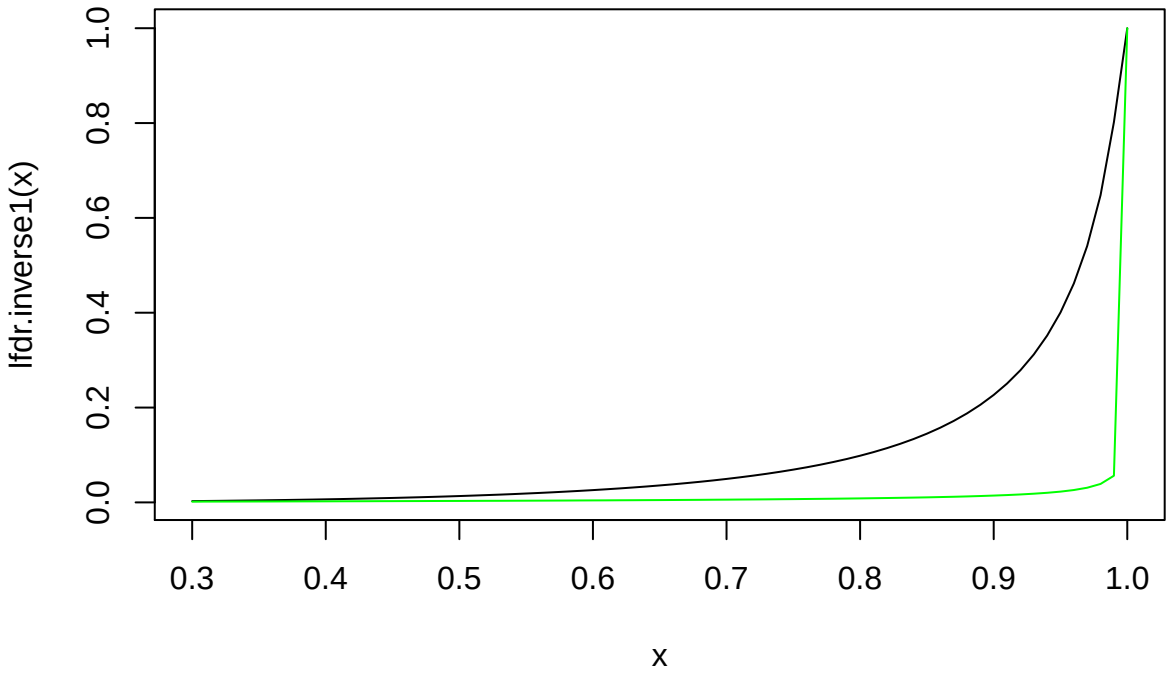
See lfdr.inverse in interval [0, 0.3]

```
thr <- 0.3
x = seq(0.001,thr,0.01)
plot(x = x, y = lfdr.inverse1(x), type = "l")
lines(x = x, y = lfdr.inverse2(x), type = "l",col = "green")
```



See lfdr.inverse in interval [0.3, 1]

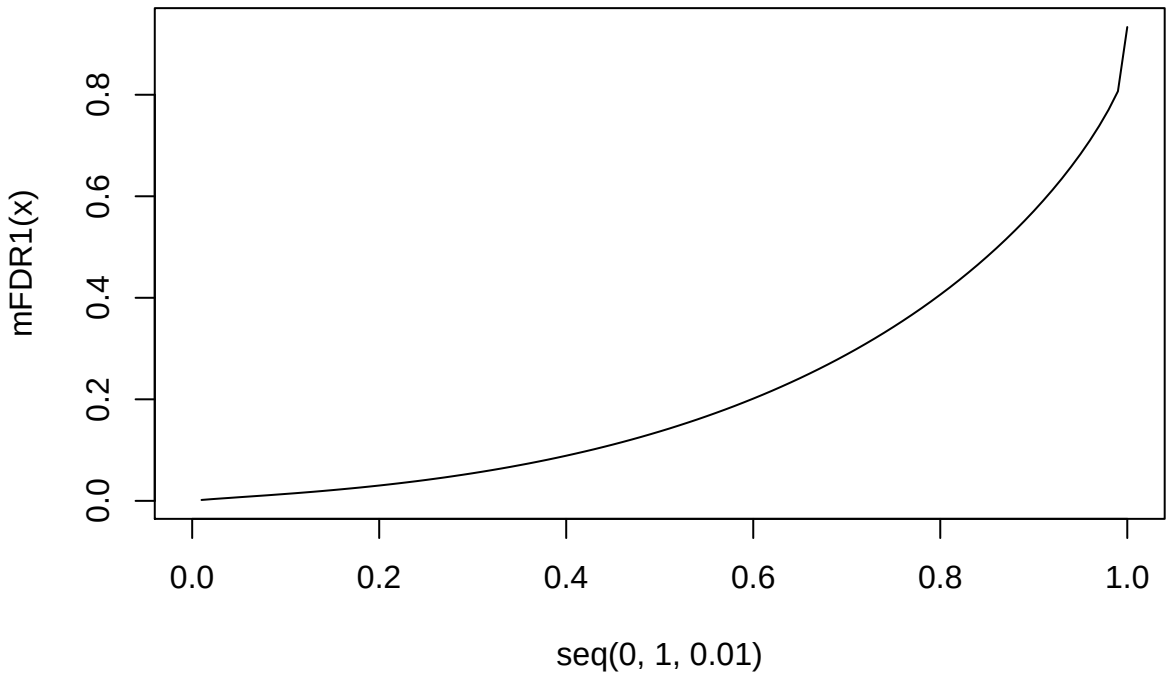
```r
x = seq(thr,1,0.01)
plot(x = x, y = lfdr.inverse1(x), type = "l")
lines(x = x, y = lfdr.inverse2(x), type = "l",col = "green")
```



mFDR function and mFDR plots versus alpha

```r
## to calculate mFDR, we need the cdf of nonnull p-values
nonnull.cdf <- function(p, mu1) {
  pnorm(qnorm(1-p) - mu1,lower.tail = F)
}

## the mFDR function gives the mFDR, when we reject lfdr < c.
mFDR <- function(c, n_g, pi0_g, mu1_g) {
  V <- c()
  R <- c()
  for(i in 1:length(pi0_g)) {
    pi0 = pi0_g[i]
    mu1 = mu1_g[i]
    n = n_g[i]
    t = lfdr.inverse(c, pi0, mu1)
    V[i] = n*pi0*t
    R[i] = V[i] + n*(1-pi0)*nonnull.cdf(t, mu1)
  }
  return(sum(V)/sum(R))
}

## the mFDR function for our example
mFDR1 <- function(c) {
  mFDR(c, n_g = n_g, pi0_g = pi0_g, mu1_g = mu1_g)
```

```
}
mFDR1 <- Vectorize(mFDR1)
x = seq(0,1,0.01)
plot(x = seq(0,1,0.01), y = mFDR1(x), type = "l")
```



Find lambda_alpha

```
lambda_alpha <- vecbinsolv(zf = alpha, fun = mFDR1 , tlo = 0, thi = 1, nits = 30)
lambda_alpha
```

```
## [1] 0.2843686
```

Compute oracle weights

```
oracle.weights <- function(alpha, n_g, pi0_g, mu1_g) {
  pi_g <- n_g/sum(n_g)
  #lambda_alpha <- solve.mFDR(alpha, n_g, pi0_g, mu1_g)
  mFDR1 <- function(c) {
    V <- c()
    R <- c()
    for(i in 1:length(pi0_g)) {
      pi0 = pi0_g[i]
      mu1 = mu1_g[i]
      n = n_g[i]
      t = lfdr.inverse(c, pi0, mu1)
      V[i] = n*pi0*t
      R[i] = V[i] + n*(1-pi0)*nonnull.cdf(t, mu1)
```

```
    }
    return(sum(V)/sum(R))
  }
  lambda_alpha <- vecbinsolv(zf = alpha, fun = mFDR1 , tlo = 0, thi = 1, nits = 30)
  w <- c()
  t <- c()
  for(i in 1:length(n_g)) {
    t[i] <- lfdr.inverse(lambda_alpha, pi0 = pi0_g[i], mu1 = mu1_g[i])
  }
  t <- t/sum(t*pi0_g*pi_g)
  for(i in 1:length(n_g)) {
    w <- c(w, rep(t[i], n_g[i]))
  }
  return(w)
}
oracle_weights <- oracle.weights(alpha, n_g, pi0_g, mu1_g)
oracle_weights <- unique(oracle_weights)
oracle_weights
```

```
## [1] 1.409474 0.911302
```

Generate data based on our setting

```
set.seed(1)

genmu <- function(n, pi1, mu1){
  m <- ceiling(n * pi1)
  mu <- rep(0, n)
  altmu <- rep(1, m)
  mu[1:m] <- mu1 * altmu
  mu
}

gen_data <- function(n_g, mu1_g, pi1_g,
                     rho, Sigma_type,
                     side,
                    nreps){
  n <- sum(n_g)
  ngroups <- length(n_g)
  Sigma <- diag(n)
  sqrtSigma <- Sigma
  mu <- c()
  groups <- c()
  for (j in 1: ngroups) {
    mu <- c(mu, genmu(n_g[j], pi1_g[j], mu1_g[j]))
    groups <- c(groups, rep(j , n_g[j]))
  }
  H0 <- mu == 0
    zvals <- list()
  for (i in 1:nreps){
    zvals[[i]] <- as.numeric(mu + sqrtSigma %*% rnorm(n))
```

```
  }
  return(list(zvals = zvals, groups = groups, H0 = H0, Sigma = Sigma))
}
data <- gen_data(n_g = n_g, mu1_g = mu1_g, pi1_g = pi1_g,
                 Sigma_type = "iid",
                 side = "right",
                 nreps = 100)
zvals <- data$zvals
groups <- data$groups
H0 <- data$H0
```

Calculate the adaptive optimal weights based on the package locfdr

```
adaptive_optimal.weights <- function(groups, zvals, alpha, side, pi0Est) {
  weights <- c()
  lfdr_g <- list()
  ngroups <- length(unique(groups))
  pi0_g <- c()
  n_g <- c()
  for(i in 1: ngroups) {
    g <- unique(groups)[i]
    zvals_g <- zvals[groups == g]
    n_g[i] <- sum(groups == g)
    lfdrres <- locfdr(zvals_g, plot = 0)
    lfdr_g[[i]] <- lfdrres$fdr
    pi0_g[i] <- min(lfdrres$fp0[1,3], lfdrres$fp0[3,3], lfdrres$fp0[5,3], 1) ###????
  }
  n <- sum(n_g)
  pi_g <- n_g/n

  lfdr <- unlist(lfdr_g, F, F)
  lfdr.order <- order(lfdr)
  st.lfdr <- lfdr[lfdr.order]
  k = max(which(cumsum(st.lfdr)/(1:n) <= alpha))
  # k = 1
  # while(k < n && (1/k)*sum(st.lfdr[1:k]) < alpha) {
  #   k = k + 1
  # }
  # k <- k-1
  thr <- st.lfdr[k]

  st.groups <- groups[lfdr.order]
  st.pvals <- zvals_pvals(zvals, side)[lfdr.order]

  t_g <- c()
  for(j in 1:ngroups) {
    g <- unique(groups)[j]
    st.lfdr_g <- st.lfdr[st.groups == j]

    if(!any(which(st.lfdr_g < thr))) {
      t_g[j] <- 0
```

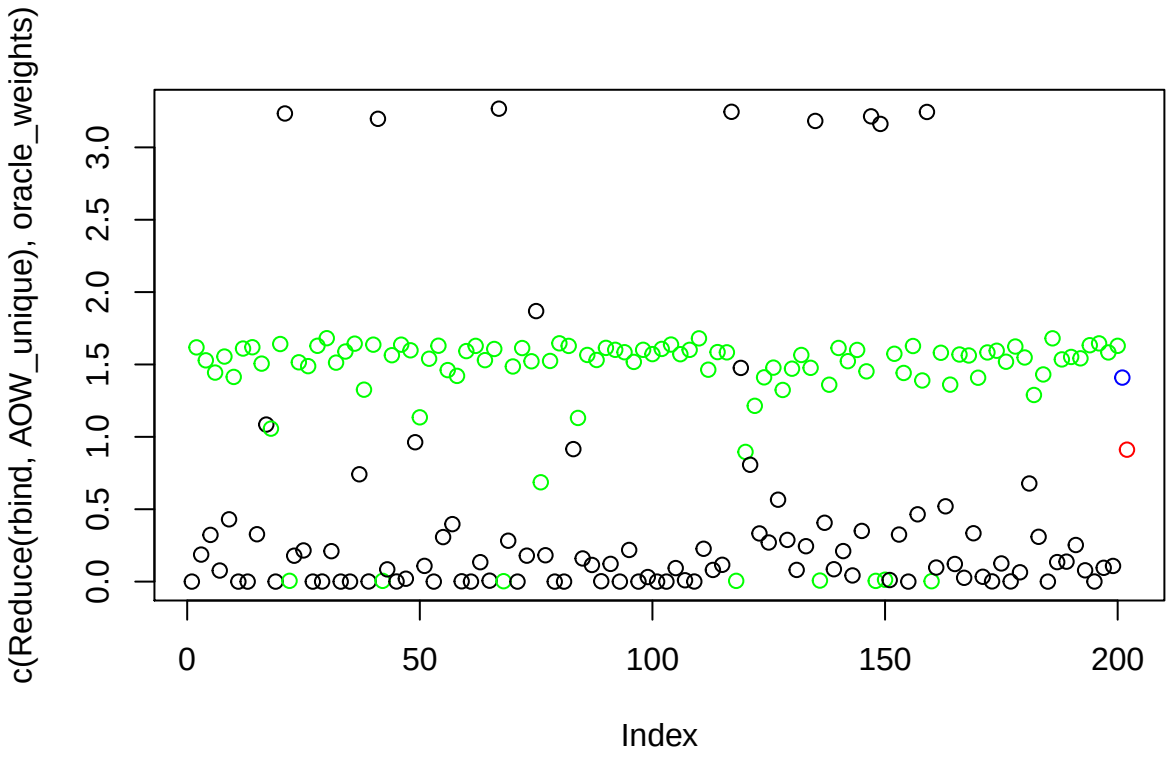
```
      #weights[groups == g] <- 1
    } else {
      pvals.order_g <- st.pvals[st.groups == j]
      t_g[j] <- pvals.order_g[max(which(st.lfdr_g < thr))]
      #weights[groups == g] <- pvals.order_g[max(which(st.lfdr_g < thr))]
    }
  }
  # if(pi0Est == F) {
  #   pi0_g <- rep(1, ngroups)
  # }

  if(sum(t_g*pi_g*pi0_g) < 1e-6) {
    return(rep(1/sum(pi_g*pi0_g), length(zvals)))
  }
  #?
  t_g.init <- t_g

  t_g <- t_g/sum(t_g*pi_g*pi0_g)

  for(j in 1: ngroups) {
    g <- unique(groups)[j]
    weights[groups == g] <- t_g[j]
  }

  return(list(weights = weights, thr = thr, t_g = t_g, t_g.init = t_g.init))
}
side <- "one"
adaptive_optimal_weights <- lapply(zvals, function(zv){
  adaptive_optimal.weights(groups, zv, alpha, side)
})
```

plot the adaptive weights versus the oracle ones

```
AOW_unique <- sapply(adaptive_optimal_weights, function(w){
  unique(w$weights)
})
rowMeans(AOW_unique)
```

```
## [1] 0.4522596 1.3925049
```

```
plot(c(Reduce(rbind, AOW_unique), oracle_weights), col = c(rep(c("black", "green"), length(AOW_unique)/
```

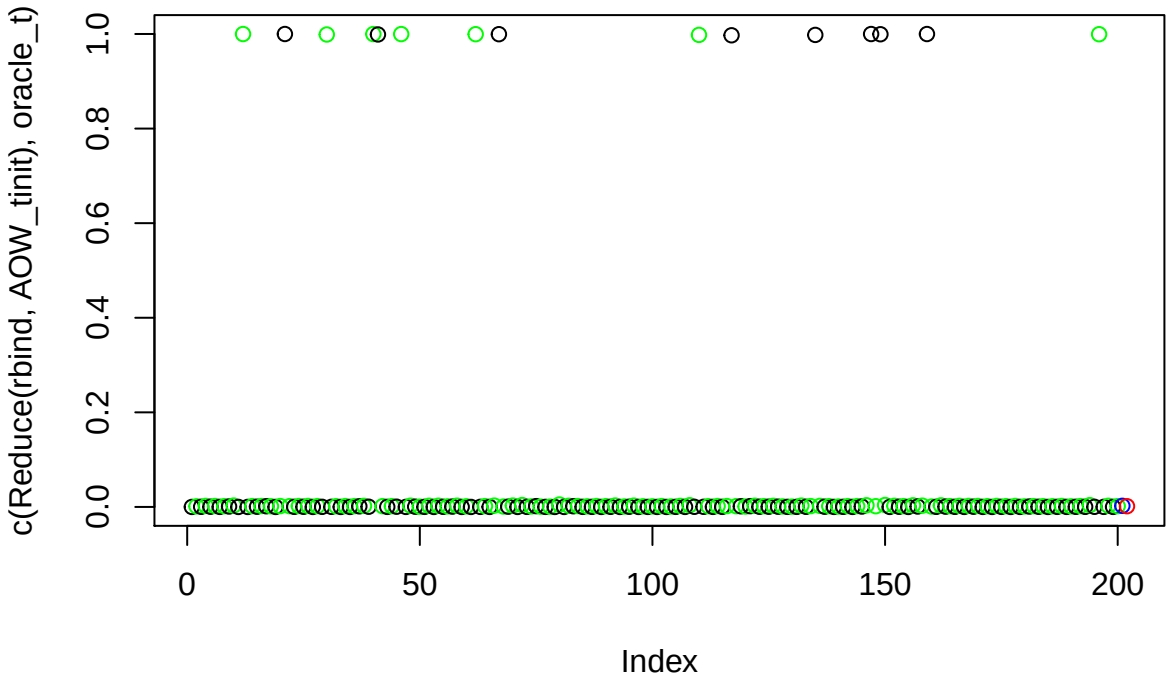


hist the thr(lambda_alpha) of our adaptive procedure

```
AOW_thr <- sapply(adaptive_optimal_weights, function(w){
  unique(w$thr)
})
hist(AOW_thr)
abline(v = lambda_alpha, col = "red")
```

plot the adaptive t_g versus the oracle ones

```
AOW_tinit <- sapply(adaptive_optimal_weights, function(w){
  unique(w$t_g.init)
})
oracle_t <- c(lfdr.inverse1(lambda_alpha), lfdr.inverse2(lambda_alpha))

## plot the AOW_init_t versus its oracle t
plot(c(Reduce(rbind, AOW_tinit), oracle_t), col = c(rep(c("black", "green"), length(AOW_tinit)/2), c("bl
```




```
## remove outliers and plot
AOW_tinit_clean <- AOW_tinit[, -ceiling(which(AOW_tinit > 0.5)/2)]
plot(c(Reduce(rbind, AOW_tinit_clean), oracle_t), col = c(rep(c("black", "green"), length(AOW_tinit_clea
```