705604096_stats102b_hw1

Jade Gregory

2024-04-15

Question 1

a) Dawn's cost function:

$$c(r,t) = r + t,\ 0 \leq r \leq 5,\ 10r < t$$

b) Nando's cost function:

- let b be the number of gym badges
- let r be the number of ribbons

$$c(b,r) = b + 2r,\ 0 \leq b + r \leq 12$$

c) My cost function:

- let t represent the time it takes to walk to class leaving 30 minutes before my class starts
- let s represent the amount of sweat produced when walking to class

$$c(t,s) = t + s,\ 0 \leq t \leq 30$$

Question 2

```
aem <- function(g, n, N, LB = -1, UB = 1, method = "deterministic", min = TRUE){

  # starting time
  st_time <- Sys.time()
  if(method == "deterministic"){

    # sample points
    x1 <- seq(LB, UB, length = N)

    # eval g at sample points
    y1 <- g(x1)

    # making grid for evaluation matrix
    mygrid <- lapply(1:n, function(x) x1)
    myp <- do.call(expand.grid, mygrid)
  }
  else if(method == "stochastic"){

    # sample points
    myp <- as.data.frame(matrix(runif(n * N, LB, UB), ncol = n))
  }

  # eval g at sampled points
  my_vals <- apply(myp, 1, g)
```

```
  # determining min or max
  if(min){
    idx_all <- which.min(my_vals)
  }else{
    idx_all <- which.max(my_vals)
  }

  # ending time
  fin_time <- Sys.time()

  # total time
  tot_time <- fin_time - st_time

  # returning a list
  final <- list(
    index = myp[idx_all,, drop = FALSE],
    val = my_vals[idx_all],
    eval_matrix = cbind(points, g = my_vals),
    time = tot_time
  )
  return(final)
  }
```

```
# test case one
f <- function(v){
  v[1]^2 + v[2]^2
}
test_1 <- aem(f, n = 2, N = 100)
test_1$index
```

```
##             Var1        Var2
## 4950 -0.01010101 -0.01010101
```

```
test_1$val
```

```
## [1] 0.0002040608
```

```
# test case two
set.seed(24601)
test_2 <- aem(f, n = 2, N = 100, method = "stochastic")
test_2$index
```

```
##            V1          V2
## 42 -0.1682719 -0.03904734
```

```
test_2$val
```

```
## [1] 0.02984014
```

```
# test case three
h <- function(v) {
exp(abs(v[1]^2 - 1) + abs(v[2]^3 + abs(v[3])))
}

test_1 <- aem(h, n = 3, N = 100, LB = -2, UB = 2)
test_1$index
```

```
##            Var1       Var2       Var3
## 473875 0.989899 -0.4646465 -0.1010101
```

```
test_1$val
```

```
## [1] 1.021012
```

```
# test case four
set.seed(24601)
test_2 <- aem(h, n = 3, N = 100, LB = -2, UB = 2, method = "stochastic")
test_2$index
```

```
##           V1          V2         V3
## 85 0.8820057 -0.04737154 0.07202588
```

```
test_2$val
```

```
## [1] 1.341764
```

```
# test case five
j <- function(v) {
log(abs(sin(v[1]) + cos(v[2])) + 1)
}

set.seed(24601)
test_2 <- aem(j, n = 2, N = 100, method = "stochastic")
test_2$index
```

```
##           V1         V2
## 44 -0.907459 -0.6993643
```

```
test_2$val
```

```
## [1] 0.02243654
```

```
# test case six
set.seed(24601)
test_2 <- aem(j, n = 2, N = 100, method = "stochastic", min = FALSE)
test_2$index
```

```
##           V1         V2
## 29 0.8163274 0.01638802
```

```
test_2$val
```

```
## [1] 1.003752
```

Question 3

a)

$$f(x|\lambda) = \lambda e^{-\lambda x},\ x \geq 0 L(\lambda) = \Pi(\lambda e^{-\lambda x_i}) lnL(\lambda) = \Sigma(ln(\lambda) - \lambda x_i) \frac{d}{d\lambda}(\Sigma(ln(\lambda) - \lambda x_i)) = 0 \Sigma(\frac{1}{\lambda} - x_i) = 0 \therefore \lambda_{mle} = \frac{n}{\Sigma x_i}$$

b)

```
# 10 total observations
n <- 10

# calculating mle
```

```r
mle <- n / ( 0.368 + 0.714 + 0.126 + 0.006 + 0.525 + 0.36 + 0.095 + 0.212 + 0.107 + 0.058)
mle
```

```
## [1] 3.889537
```

c)

```r
# log likelihood function
l <- function(lambda){
  10 * log(lambda[1])  -  log(lambda[1]) * (0.368 + 0.714 + 0.126 + 0.006 + 0.525 + 0.36 + 0.095 + 0.21
}

# test case with log likelihood
set.seed(24601)
lltest <- aem(l, n = 1, N = 1000, LB = 0, UB = 5, method = "stochastic", min = FALSE)
lltest$index
```

```
##           V1
## 827 4.997914
```

```r
lltest$val
```

```
## [1] 11.95341
```

```r
lltest$index - mle
```

```
##           V1
## 827 1.108377
```

In this case, our maximum likelihood estimator has a value of 4.997914 that produces an estimate value of 11.95341. Our actual MLE value is 3.889537, so these two estimates have a difference of 1.108377. If this value is small compared to the spread of our data points, then we could classify these two estimates as comparable. But, since the bounds are [0,5], I am not confident that they are.

Question 4

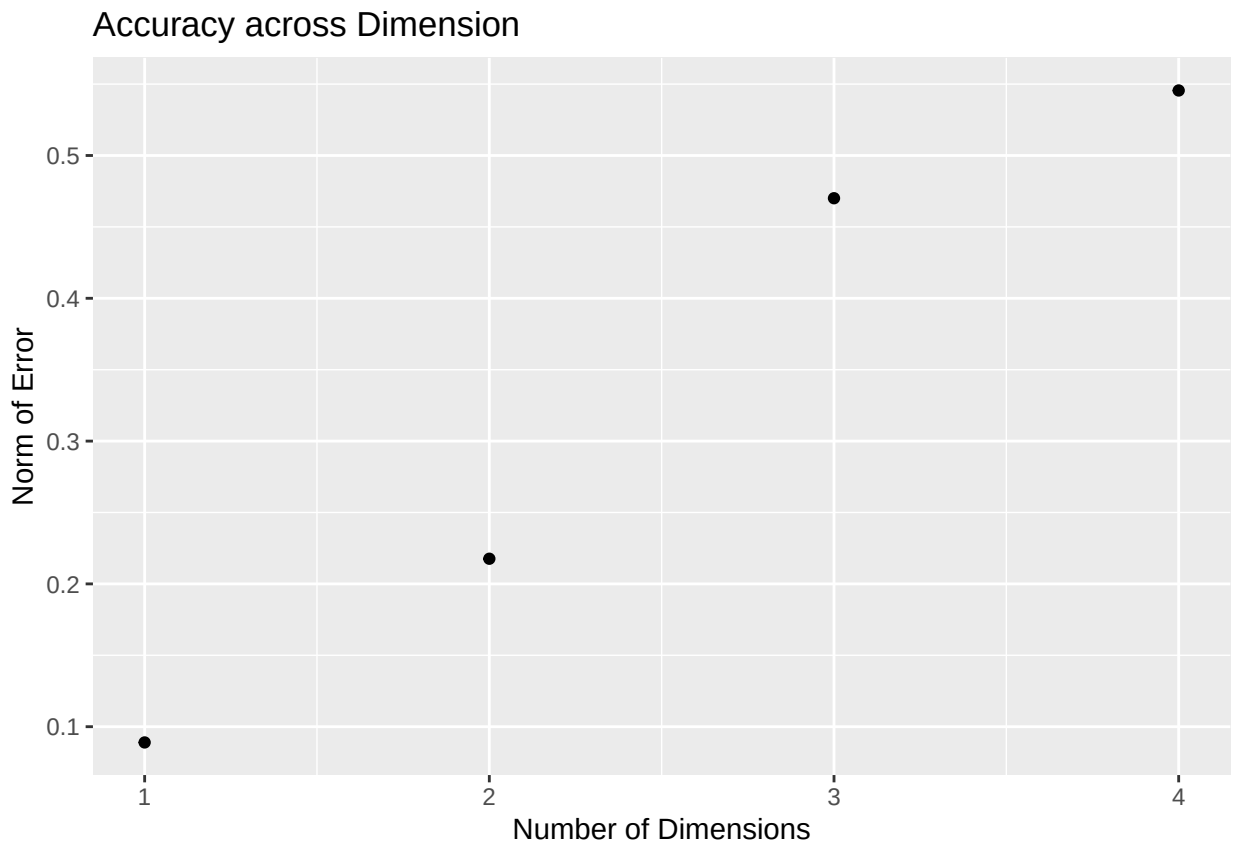
```r
q4 <- function(x){
  sum(x^2)
}

# all aem functions with n from 1 to 4
set.seed(24601)
n1 <- aem(q4, n = 1, N = 30, LB = -1, UB = 1, method = "stochastic", min = TRUE)
n2 <- aem(q4, n = 2, N = 30, LB = -1, UB = 1, method = "stochastic", min = TRUE)
n3 <- aem(q4, n = 3, N = 30, LB = -1, UB = 1, method = "stochastic", min = TRUE)
n4 <- aem(q4, n = 4, N = 30, LB = -1, UB = 1, method = "stochastic", min = TRUE)
```

```r
# taking euclidean norm of error
n_error <- c(sqrt(sum(n1$index^2)), sqrt(sum(n2$index^2)), sqrt(sum(n3$index^2)), sqrt(sum(n4$index^2))
time_taken <- c(n1$time, n2$time, n3$time, n4$time)
ns <- 1:4

# print euclidean norm of error results to prove it is increasing in value across dimension
n_error
```

```
## [1] 0.08896783 0.21762000 0.47008358 0.54553851
```

```r
# creating data frame of value data
myvaluedat <- data.frame(ns, n_error)

# creating data frame of time data to plot
mytimedat <- data.frame(ns, time_taken)
```

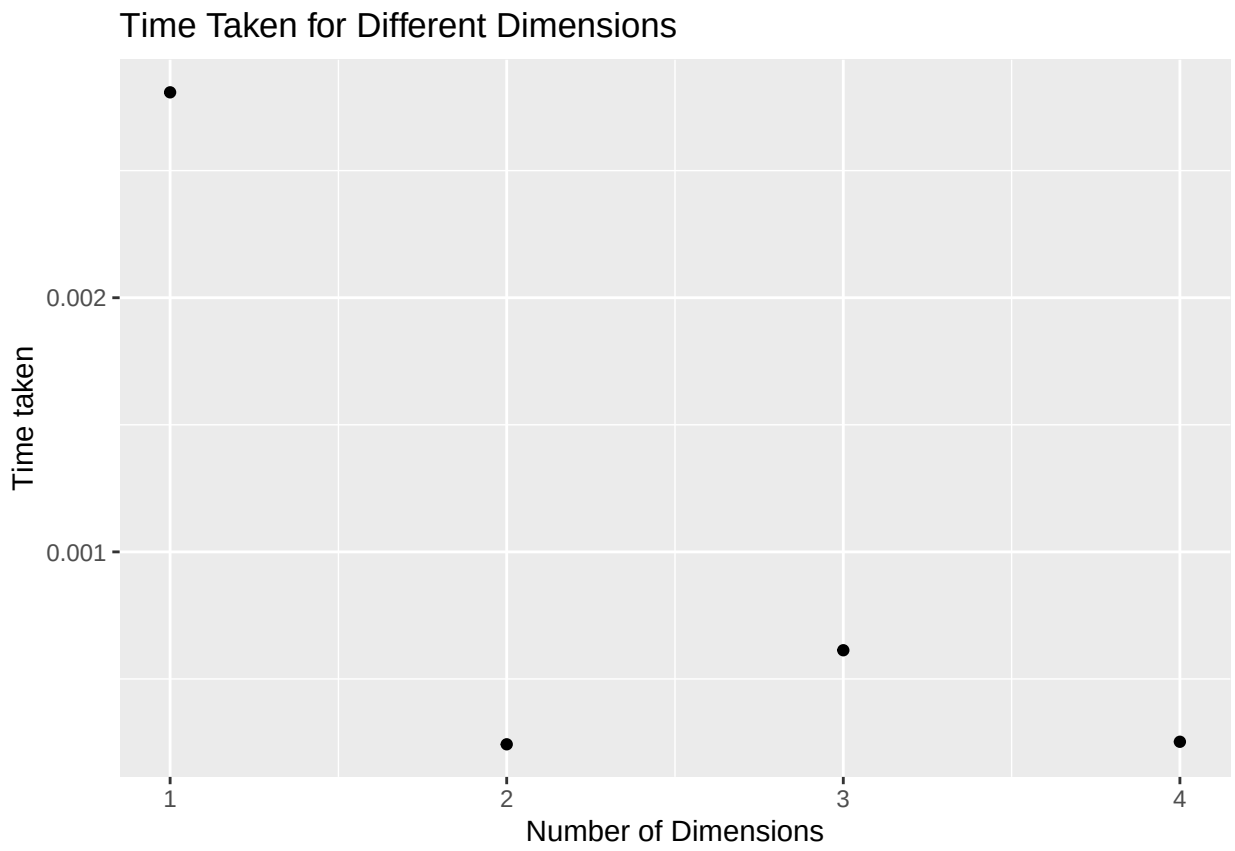
```r
# plotting each value of $val for each n
ggplot(myvaluedat, aes(x = ns, y = n_error)) +
  geom_point() +
  labs(x = "Number of Dimensions",
       y = "Norm of Error",
       title = "Accuracy across Dimension")
```



We can see that as our dimension increases, our euclidean norm of our error value from our stochastic process increases as well.

```r
# plotting the data
set.seed(24601)
ggplot(mytimedat, aes(x = ns, y = time_taken)) +
  geom_point() +
  labs(x = "Number of Dimensions",
       y = "Time taken",
       title = "Time Taken for Different Dimensions")
```

```
## Don't know how to automatically pick scale for object of type <difftime>.
## Defaulting to continuous.
```



Question 5

a) In one dimension, the random search, coordinate search, and coordinate descent algorithms all only have the option of stepping forward or backward on a line. This condenses these algorithms down to being practically the same in one dimension.

b)

```
zom <- function(g, alpha, w0, K){
  for(k in 1:K){
    forward <- g(w0 + alpha)
    back <- g(w0 - alpha)

    if(forward < g(w0)){
      w0 <- w0 + alpha
    } else if (back < g(w0)){
      w0 <- w0 - alpha
    }
  }
  list(
    index = w0,
    val = g(w0)
  )
}
```

c)

```r
# verifying zom function
gw <- function(w){
  sin(3*w) + 0.3* w^2
}

# test case with w0 = 4.5
zom(gw, 0.1, 4.5, 10)
```

```
## $index
## [1] 3.5
##
## $val
## [1] 2.795304
```

```r
# test case with w0 = -1.5
zom(gw, 0.1, -1.5, 10)
```

```
## $index
## [1] -0.5
##
## $val
## [1] -0.922495
```

d)

```r
# making negative log likelihood function
neg_l <- function(lambda){
 -1 * ( 10 * log(lambda[1])  -  log(lambda[1]) * (0.368 + 0.714 + 0.126 + 0.006 + 0.525 + 0.36 + 0.095
}

# testing using negative log likelihood function
zom(neg_l, 0.01, 1, 1000)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

e)

```r
# testing with alpha in {1, 0.1, 0.01, 0.001} and K in {10, 100, 1000, 10000}
zom(neg_l, 1, 1, 10)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

```r
zom(neg_l, 0.1, 1, 100)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

```
zom(neg_l, 0.01, 1, 1000)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

```
zom(neg_l, 0.001, 1, 10000)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

```
zom(neg_l, 1, 1, 10)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

```
zom(neg_l, 1, 1, 100)
```

```
## $index
## [1] 101
##
## $val
## [1] -34.28573
```

```
zom(neg_l, 1, 1, 1000)
```

```
## $index
## [1] 1001
##
## $val
## [1] -51.32514
```

```
zom(neg_l, 0.1, 1, 1000)
```

```
## $index
## [1] 101
##
## $val
## [1] -34.28573
```

```
zom(neg_l, 0.01, 1, 1000)
```

```
## $index
## [1] 11
##
## $val
## [1] -17.81396
```

```
zom(neg_l, 0.001, 1, 1000)
```

```
## $index
```

```
## [1] 2
##
## $val
## [1] -5.14939
```

We can see as we hold alpha at 1, increasing the step size increases the output index values and decreases the output val values. When holding the number of iterations constant, we can see that decreasing out alpha value decreases our index output and increases the value of our val output. If we increase the iteration count by the same augmentation as we decrease our step size value by, we get the same output values over and over again.