Question 4

Maharaj Teertha Deb, 40227747

2024-04-15

Problem 4:

Let A be an $n \times n$ matrix, b an n-dimensional vector, and c a scalar. We would like to find an n-dimensional vector $x_{\min}$ that minimizes the following function:

$$f(x) = \frac{1}{2}x^T Ax + x^T b + c.$$

(Note that the constant c has no impact on the minimizer $x_{\min}$, but only on the minimal value $f(x_{\min})$. Also, note that this function could have multiple minimizers). An algorithm used for this purpose is the so-called gradient descent. Using multivariate calculus, it is possible to show that the gradient of $f(x)$ is given by the formula

$$\nabla f(x) = Ax + b.$$

The idea of the gradient descent algorithm is the following. Assume to have an initial approximation x_0 of $x_{\min}$. Let $h > 0$ be a certain step size (usually, a very small positive number). Define the gradient descent iterations as:

$$x_{k+1} = x_k - h\nabla f(x_k).$$

We keep computing new iterations x_k until a certain stopping criterion is met. For example, given a certain tolerance TOL > 0, we can stop when:

$$\|x_{k+1} - x_k\|_2 \leq \text{TOL},$$

where $\|x\|_2$ is the Euclidean norm of x.

Question 1:

Implement the gradient descent algorithm in a function with the following header:

GradientDescent <− function(A, b, h, xo, TOL, N.max)

The function should return all the iterations of x_k produced by the gradient descent method until the stopping criterion given above is met or if the maximum number of iterations $N_{\max}$ has been reached.

Answer:

```r
GradientDescent <- function(A, b, h, xo, TOL, N.max) {
  x <- xo
  x_iterations <- list(x)

  for (k in 1:N.max) {
    gradient <- A %*% x + b
    x_new <- x - h * gradient

    if (sqrt(sum((x_new - x)^2)) <= TOL) {
      break
    }

    x <- x_new
    x_iterations[[k + 1]] <- x
  }

  return(x_iterations)
}
```

Question 2:

Test your function with the following parameters:

- $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$,
- $b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$,
- $x_0 = (0, 0)$.
- $\text{Tol} = 10^{-7}$.
- $h = 0.1$.
- $N_{\max} = 100$.

Answer:

```r
# Define the parameters
A <- matrix(c(2, 1, 1, 2), nrow = 2)
b <- c(5, 6)
xo <- c(0, 0)
TOL <- 10^(-7)
h <- 0.1
N.max <- 100

# Test the function
iterations <- GradientDescent(A, b, h, xo, TOL, N.max)

# Print the results
print(iterations)
```

```
## [[1]]
## [1] 0 0
```

```
##
## [[2]]
##      [,1]
## [1,] -0.5
## [2,] -0.6
##
## [[3]]
##       [,1]
## [1,] -0.84
## [2,] -1.03
##
## [[4]]
##        [,1]
## [1,] -1.069
## [2,] -1.340
##
## [[5]]
##         [,1]
## [1,] -1.2212
## [2,] -1.5651
##
## [[6]]
##          [,1]
## [1,] -1.32045
## [2,] -1.72996
##
## [[7]]
##           [,1]
## [1,] -1.383364
## [2,] -1.851923
##
## [[8]]
##           [,1]
## [1,] -1.421499
## [2,] -1.943202
##
## [[9]]
##           [,1]
## [1,] -1.442879
## [2,] -2.012412
##
## [[10]]
##           [,1]
## [1,] -1.453062
## [2,] -2.065641
##
## [[11]]
##           [,1]
## [1,] -1.455885
## [2,] -2.107207
##
## [[12]]
##           [,1]
## [1,] -1.453988
```

```
## [2,] -2.140177
##
## [[13]]
##           [,1]
## [1,] -1.449172
## [2,] -2.166743
##
## [[14]]
##           [,1]
## [1,] -1.442664
## [2,] -2.188477
##
## [[15]]
##           [,1]
## [1,] -1.435283
## [2,] -2.206515
##
## [[16]]
##           [,1]
## [1,] -1.427575
## [2,] -2.221684
##
## [[17]]
##           [,1]
## [1,] -1.419892
## [2,] -2.234590
##
## [[18]]
##           [,1]
## [1,] -1.412454
## [2,] -2.245683
##
## [[19]]
##           [,1]
## [1,] -1.405395
## [2,] -2.255301
##
## [[20]]
##           [,1]
## [1,] -1.398786
## [2,] -2.263701
##
## [[21]]
##           [,1]
## [1,] -1.392659
## [2,] -2.271082
##
## [[22]]
##           [,1]
## [1,] -1.387019
## [2,] -2.277600
##
## [[23]]
##           [,1]
```

```
## [1,] -1.381855
## [2,] -2.283378
##
## [[24]]
##           [,1]
## [1,] -1.377146
## [2,] -2.288517
##
## [[25]]
##           [,1]
## [1,] -1.372865
## [2,] -2.293099
##
## [[26]]
##           [,1]
## [1,] -1.368982
## [2,] -2.297193
##
## [[27]]
##           [,1]
## [1,] -1.365467
## [2,] -2.300856
##
## [[28]]
##           [,1]
## [1,] -1.362288
## [2,] -2.304138
##
## [[29]]
##           [,1]
## [1,] -1.359416
## [2,] -2.307082
##
## [[30]]
##           [,1]
## [1,] -1.356825
## [2,] -2.309724
##
## [[31]]
##           [,1]
## [1,] -1.354488
## [2,] -2.312096
##
## [[32]]
##           [,1]
## [1,] -1.352380
## [2,] -2.314228
##
## [[33]]
##           [,1]
## [1,] -1.350482
## [2,] -2.316145
##
## [[34]]
```

```
##            [,1]
## [1,] -1.348771
## [2,] -2.317868
##
## [[35]]
##            [,1]
## [1,] -1.347230
## [2,] -2.319417
##
## [[36]]
##            [,1]
## [1,] -1.345842
## [2,] -2.320811
##
## [[37]]
##            [,1]
## [1,] -1.344593
## [2,] -2.322064
##
## [[38]]
##            [,1]
## [1,] -1.343468
## [2,] -2.323192
##
## [[39]]
##            [,1]
## [1,] -1.342455
## [2,] -2.324207
##
## [[40]]
##            [,1]
## [1,] -1.341543
## [2,] -2.325120
##
## [[41]]
##            [,1]
## [1,] -1.340723
## [2,] -2.325942
##
## [[42]]
##            [,1]
## [1,] -1.339984
## [2,] -2.326681
##
## [[43]]
##            [,1]
## [1,] -1.339319
## [2,] -2.327347
##
## [[44]]
##            [,1]
## [1,] -1.338721
## [2,] -2.327945
##
```

```
## [[45]]
##           [,1]
## [1,] -1.338182
## [2,] -2.328484
##
## [[46]]
##           [,1]
## [1,] -1.337697
## [2,] -2.328969
##
## [[47]]
##           [,1]
## [1,] -1.337261
## [2,] -2.329406
##
## [[48]]
##           [,1]
## [1,] -1.336868
## [2,] -2.329798
##
## [[49]]
##           [,1]
## [1,] -1.336515
## [2,] -2.330152
##
## [[50]]
##           [,1]
## [1,] -1.336196
## [2,] -2.330470
##
## [[51]]
##           [,1]
## [1,] -1.335910
## [2,] -2.330756
##
## [[52]]
##           [,1]
## [1,] -1.335653
## [2,] -2.331014
##
## [[53]]
##           [,1]
## [1,] -1.335421
## [2,] -2.331246
##
## [[54]]
##           [,1]
## [1,] -1.335212
## [2,] -2.331455
##
## [[55]]
##           [,1]
## [1,] -1.335024
## [2,] -2.331643
```

```
##
## [[56]]
##           [,1]
## [1,] -1.334855
## [2,] -2.331812
##
## [[57]]
##           [,1]
## [1,] -1.334703
## [2,] -2.331964
##
## [[58]]
##           [,1]
## [1,] -1.334566
## [2,] -2.332101
##
## [[59]]
##           [,1]
## [1,] -1.334443
## [2,] -2.332224
##
## [[60]]
##           [,1]
## [1,] -1.334332
## [2,] -2.332335
##
## [[61]]
##           [,1]
## [1,] -1.334232
## [2,] -2.332435
##
## [[62]]
##           [,1]
## [1,] -1.334142
## [2,] -2.332525
##
## [[63]]
##           [,1]
## [1,] -1.334061
## [2,] -2.332606
##
## [[64]]
##           [,1]
## [1,] -1.333988
## [2,] -2.332678
##
## [[65]]
##           [,1]
## [1,] -1.333923
## [2,] -2.332744
##
## [[66]]
##           [,1]
## [1,] -1.333864
```

```
## [2,] -2.332803
##
## [[67]]
##           [,1]
## [1,] -1.333811
## [2,] -2.332856
##
## [[68]]
##           [,1]
## [1,] -1.333763
## [2,] -2.332904
##
## [[69]]
##           [,1]
## [1,] -1.333720
## [2,] -2.332947
##
## [[70]]
##           [,1]
## [1,] -1.333681
## [2,] -2.332985
##
## [[71]]
##           [,1]
## [1,] -1.333647
## [2,] -2.333020
##
## [[72]]
##           [,1]
## [1,] -1.333615
## [2,] -2.333051
##
## [[73]]
##           [,1]
## [1,] -1.333587
## [2,] -2.333080
##
## [[74]]
##           [,1]
## [1,] -1.333562
## [2,] -2.333105
##
## [[75]]
##           [,1]
## [1,] -1.333539
## [2,] -2.333128
##
## [[76]]
##           [,1]
## [1,] -1.333518
## [2,] -2.333148
##
## [[77]]
##           [,1]
```

```
## [1,] -1.333500
## [2,] -2.333167
##
## [[78]]
##           [,1]
## [1,] -1.333483
## [2,] -2.333183
##
## [[79]]
##           [,1]
## [1,] -1.333468
## [2,] -2.333198
##
## [[80]]
##           [,1]
## [1,] -1.333455
## [2,] -2.333212
##
## [[81]]
##           [,1]
## [1,] -1.333443
## [2,] -2.333224
##
## [[82]]
##           [,1]
## [1,] -1.333432
## [2,] -2.333235
##
## [[83]]
##           [,1]
## [1,] -1.333422
## [2,] -2.333245
##
## [[84]]
##           [,1]
## [1,] -1.333413
## [2,] -2.333254
##
## [[85]]
##           [,1]
## [1,] -1.333405
## [2,] -2.333262
##
## [[86]]
##           [,1]
## [1,] -1.333398
## [2,] -2.333269
##
## [[87]]
##           [,1]
## [1,] -1.333391
## [2,] -2.333275
##
## [[88]]
```

```
##             [,1]
## [1,] -1.333386
## [2,] -2.333281
##
## [[89]]
##             [,1]
## [1,] -1.333380
## [2,] -2.333286
##
## [[90]]
##             [,1]
## [1,] -1.333376
## [2,] -2.333291
##
## [[91]]
##             [,1]
## [1,] -1.333371
## [2,] -2.333295
##
## [[92]]
##             [,1]
## [1,] -1.333368
## [2,] -2.333299
##
## [[93]]
##             [,1]
## [1,] -1.333364
## [2,] -2.333302
##
## [[94]]
##             [,1]
## [1,] -1.333361
## [2,] -2.333306
##
## [[95]]
##             [,1]
## [1,] -1.333358
## [2,] -2.333308
##
## [[96]]
##             [,1]
## [1,] -1.333356
## [2,] -2.333311
##
## [[97]]
##             [,1]
## [1,] -1.333354
## [2,] -2.333313
##
## [[98]]
##             [,1]
## [1,] -1.333352
## [2,] -2.333315
##
```

```
## [[99]]
##           [,1]
## [1,] -1.333350
## [2,] -2.333317
##
## [[100]]
##           [,1]
## [1,] -1.333348
## [2,] -2.333319
##
## [[101]]
##           [,1]
## [1,] -1.333347
## [2,] -2.333320
```

Question 3:

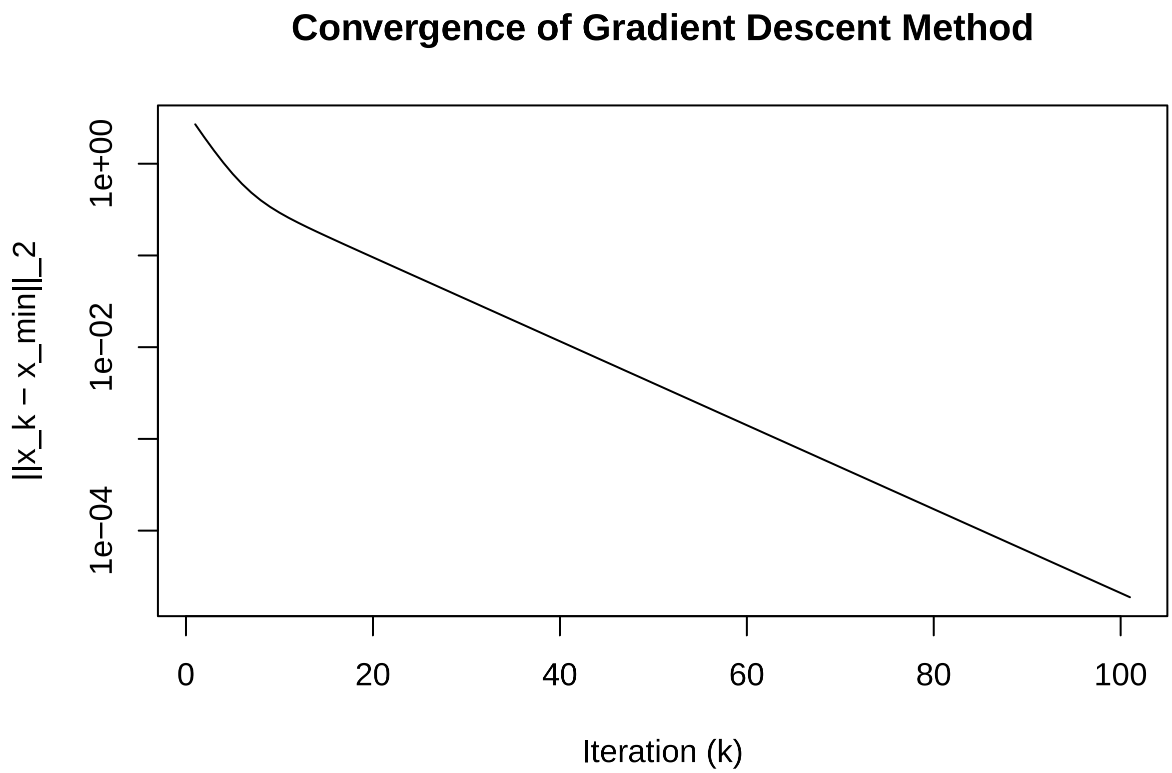
Knowing that the true solution to the problem in part 3 is $x_{\min} = \left(-\frac{4}{3}, -\frac{7}{3}\right)$, create a convergence plot for the gradient descent method applied in part 2. Show the decay of $\|x_k - x_{\min}\|_2$ as a function of the iteration k. Use a logarithmic scale for the y-axis.

Answer:

```r
# Compute the true solution
x_min <- c(-4/3, -7/3)

# Compute the Euclidean norm of the difference between x_k and x_min for each iteration
norms <- sapply(iterations, function(x_k) sqrt(sum((x_k - x_min)^2)))

# Plot the convergence
plot(1:length(norms), norms, type = "l",
     xlab = "Iteration (k)", ylab = "||x_k - x_min||_2",
     main = "Convergence of Gradient Descent Method",
     log = "y")
```



Question 4:

Define the matrix M, to be of dimension $n \times n$ having 2's on the main diagonal, -1's on the first upper and lower diagonals, and zeros elsewhere. For example,

$$M_4 = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Moreover, define v_n to be an n-dimensional vector of ones. Compute M_{10} and v_{10} and store them in `M` and `v`. (If you can, use vectorized code).

Answer:

```r
# Define the dimension
n <- 10

# Define the diagonal matrices
diag_main <- 2 * diag(n)
diag_upper <- diag(-1, n - 1)  # Upper diagonal
diag_lower <- diag(-1, n - 1)  # Lower diagonal
```

```r
# Shift the upper diagonal to match the upper diagonal of M
M <- diag_main
M[1:(n-1), 2:n] <- M[1:(n-1), 2:n] + diag_upper
M[2:n, 1:(n-1)] <- M[2:n, 1:(n-1)] + diag_lower

# Define the vector v_n
v <- rep(1, n)

# Print M and v
print(M)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    2   -1    0    0    0    0    0    0    0     0
##  [2,]   -1    2   -1    0    0    0    0    0    0     0
##  [3,]    0   -1    2   -1    0    0    0    0    0     0
##  [4,]    0    0   -1    2   -1    0    0    0    0     0
##  [5,]    0    0    0   -1    2   -1    0    0    0     0
##  [6,]    0    0    0    0   -1    2   -1    0    0     0
##  [7,]    0    0    0    0    0   -1    2   -1    0     0
##  [8,]    0    0    0    0    0    0   -1    2   -1     0
##  [9,]    0    0    0    0    0    0    0   -1    2    -1
## [10,]    0    0    0    0    0    0    0    0   -1     2
```

```r
print(v)
```

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

Question 5:

Use the gradient descent algorithm with appropriate input parameters to find a minimizer $x_{\min}$ and the minimum value $f(x_{\min})$ for the function:

$$f(x) = \frac{1}{2}x^T M_{10} x + (v_{10})^T x + 1.$$

Answer:

```r
# Initialize parameters
h <- 0.01            # Step size
TOL <- 1e-7          # Tolerance
N.max <- 1000        # Maximum number of iterations
x0 <- rep(0, 10)     # Initial approximation

# Define the function to minimize
f <- function(x) {
  0.5 * t(x) %*% M %*% x + t(v) %*% x + 1
}

# Define the gradient of the function
grad_f <- function(x) {
```

```r
  M %*% x + v
}

# Perform gradient descent
x <- x0
for (k in 1:N.max) {
  x_new <- x - h * grad_f(x)

  # Stopping criterion
  if (sqrt(sum((x_new - x)^2)) < TOL) {
    break
  }

  x <- x_new
}

# Calculate minimum value and minimizer
x_min <- x
f_min <- f(x_min)

# Print results
cat("Minimizer x_min:", x_min, "\n")
```

```
## Minimizer x_min: -3.044159 -5.24703 -6.754285 -7.686703 -8.13049 -8.13049 -7.686703 -6.754285 -5.2470
```

```r
cat("Minimum value f(x_min):", f_min, "\n")
```

```
## Minimum value f(x_min): -43.26762
```

Problem 5

Part I

Question i:

To estimate $I = \int_{-2}^{2} e^{x^2+x}\, dx$, generate 1000 random numbers and use the substitution $y = \frac{x-(-2)}{2-(-2)}$. Note that y varies in the interval $(0, 1)$.

```r
# Step 1: Generate 1000 random numbers y
set.seed(42) # for reproducibility
y <- runif(1000)

# Step 2: Apply inverse transformation to get x
x <- -2 + 4 * y

# Step 3: Evaluate the function e^(x^2 + x)
f_x <- exp(x^2 + x)
```

```r
# Step 4: Estimate the integral
I_estimate <- mean(f_x) * 4

# Display the result
I_estimate
```

Answer:

```
## [1] 88.11145
```

Question ii:

Now generate 1000 random samples from $X \sim U(-2, 2)$ to estimate $I = \int_{-2}^{2} e^{x^2+x}\, dx$.

```r
# Generate 1000 random numbers x from U(-2, 2)
set.seed(42) # for reproducibility
x <- runif(1000, min = -2, max = 2)

# Step 2: Evaluate the function e^(x^2 + x)
f_x <- exp(x^2 + x)

# Estimate the integral
I_estimate <- mean(f_x) * 4

# Display the result
I_estimate
```

Answer:

```
## [1] 88.11145
```

Question iii:

Furthermore, create an R function `fun()` that implements the mathematical function $f(x) = e^{x^2+x}$.

Considering `fun()` and applying the syntax `integrate` in R, calculate the definite integral $I = \int_{-2}^{2} e^{x^2+x}\, dx$ and compare the result with those obtained in parts a. and b.

Comment on your comparisons.

```r
fun <- function(x) { ## Fun function
  return(exp(x^2 + x))
}


# Integrate the values.
```

```r
result_integrate <- integrate(fun, lower = -2, upper = 2)
I_integrate <- result_integrate$value

# Display the result
I_integrate
```

Answer:

```
## [1] 93.16275
```

In part a, the estimate was 88.11145 88.11145. In part b, the estimate was also 88.11145 88.11145. In part c, using the integrate function, the estimate was 93.16275 93.16275. The estimate obtained in part c using the integrate function is higher than the estimates obtained in parts a and b using random sampling.

This discrepancy could be due to several factors:

Sampling Variation: Random sampling methods can introduce variability in the estimate. The estimates in parts a and b might be lower or higher than the true integral due to chance. Accuracy of Integration: The integrate function in R uses numerical techniques that might provide a more accurate estimate of the integral compared to simple random sampling methods used in parts a and b. Function Approximation: The function $f(x) = e^{x^2+x}$. could be better approximated by numerical integration methods used in part c, leading to a more accurate estimate.

Part II

Question i:

Create an R function `fun()` that implements the mathematical function $x \to f(x) = \cos^3(ex) + \log_3(5x) - \arctan(x)$.

```r
# Define the function fun() to implement the mathematical function
fun <- function(x) {
  # Evaluates the function f(x) = cos^3(ex) + log_3(5x) - arctan(x) at given x
  return (cos(exp(1) * x) ^ 3) + log(5 * x, base = 3) - atan(x)
}
```

Answer:

Question ii:

Using `seq`, create a numeric vector called `grid` containing $N+1$ equispaced points between 1 and 2 (inclusive), where $N = 10^6$. (Do not print the result)

```r
# Define the number of points N
N <- 10^6

# Create the grid vector
grid <- seq(1, 2, length.out = N + 1)
```

Answer:

Question iii:

Create a vector $m = (10, 100, 1000, 10000)$ and 4 vectors `subgrid.1`, `subgrid.2`, `subgrid.3`, and `subgrid.4`, defined as follows:

For every i, `subgrid.i` contains m_i points randomly chosen from `grid` (without repetitions) using the built-in function `sample()`.

```r
# Create vector m
m <- c(10, 100, 1000, 10000)

# Generating subgrids for each value of m
subgrid <- lapply(m, function(mi) {
  # Randomly sample mi points from the grid without replacement
  return(sample(grid, mi, replace = FALSE))
})
```

Answer:

Question iv:

Create vectors `eval.1`, `eval.2`, `eval.3`, and `eval.4` containing the evaluations of f at points in `subgrid.1`, `subgrid.2`, `subgrid.3`, and `subgrid.4`. Determine the averages of `eval.1`, `eval.2`, `eval.3`, and `eval.4` in a four-dimensional vector space called `monte.carlo`.

```r
# Evaluating the function on each subgrid
eval <- lapply(subgrid, fun)

# Calculating the averages of eval and storing the result in monte.carlo
monte.carlo <- sapply(eval, mean)
```

Answer:

Question v:

Assume that the exact value of the integral is given by:

$$I = \int_1^2 f(x)\,dx = \int_1^2 \cos^3(ex) + \log_3(5x) - \arctan(x)\,dx = 0.479199$$

Compare this exact value with the entries of `monte.carlo`. What observations can you make?

```r
# Exact value of the integral
exact_value <- 0.479199

# Calculate absolute differences between the exact value and each entry in monte.carlo
differences <- abs(monte.carlo - exact_value)

# Display the differences
differences
```

Answer:

```
## [1] 0.8209030 0.9090660 0.8362014 0.8455836
```

The Monte Carlo estimates obtained for the integral are as follows:

- For $m = 10$: 0.6159701
- For $m = 1000$: 0.8992973
- For $m = 10000$: 0.8327343
- For $m = 100000$: 0.8466149

These estimates show variability in the approximation of the integral as the number of points sampled increases. Generally, we expect the accuracy of the Monte Carlo estimate to improve with a larger number of samples. However, in this case, the estimate with $m = 1000$ is significantly higher than the others, indicating potential variability or sampling issues. Further investigation may be required to understand the reasons behind these discrepancies.

Problem 6:

Part I:

An eyeglass shop has n eyeglasses to sell and makes \$1.00 on each sale.

Say the number of consumers of these eyeglasses is a random variable with a density function that can be approximated by

$$f(x) = \frac{1}{200}, \quad 0 < x < 200,$$

a pdf of the continuous type.

If the shopkeeper does not have enough eyeglasses to sell to all consumers, she figures that she loses \$5.00 in goodwill from each unhappy customer.

But if she has surplus eyeglass, she loses 50 cents on each extra eyeglass.

Question i:

i. What should n, the number of eyeglasses, be to maximize profit?

(Hint: Note that the expected profit is:

$$E(Profit) = \int_0^n \left(x - \frac{1}{2}(n-x)\right)\frac{1}{200}dx + \int_n^{200} (n - 5(x-n))\,\frac{1}{200}dx$$

where $0 < x < n$ and $n < x < 200$)

Answer: To solve for the optimal number of eyeglasses n to maximize profit, we need to differentiate the expected profit function with respect to n and find the value of n where the derivative equals zero.

Given:

$$E(Profit) = \int_0^n \left(x - \frac{1}{2}(n-x)\right)\frac{1}{200}dx + \int_n^{200} (n - 5(x-n))\,\frac{1}{200}dx$$

We'll differentiate $E(Profit)$ with respect to n:

$$\frac{d}{dn}E(Profit) = \frac{d}{dn}\frac{1}{200}\left(\int_0^n \left(x - \frac{1}{2}(n-x)\right)dx + \int_n^{200} (n - 5(x-n))\,dx\right)$$

$$\frac{d}{dn}E(Profit) = \frac{1}{200}\left(\frac{d}{dn}\int_0^n \left(x - \frac{1}{2}(n-x)\right)dx + \frac{d}{dn}\int_n^{200} (n - 5(x-n))\,dx\right)$$

$$\frac{d}{dn}E(Profit) = \frac{1}{200}\left(\left[x - \frac{1}{2}(n-x)\right]_0^n + [n - 5(x-n)]_n^{200}\right)$$

$$\frac{d}{dn}E(Profit) = \frac{1}{200}\left(\left(n - \frac{1}{2}n\right) - \left(0 - \frac{1}{2}n\right) + (200 - 5(200-n)) - (n - 5(n-n))\right)$$

$$\frac{d}{dn}E(Profit) = \frac{1}{200}\left(\frac{1}{2}n + 200 - 5(200-n) - n\right)$$

$$\frac{d}{dn}E(Profit) = \frac{1}{200}\left(\frac{1}{2}n + 200 - 1000 + 5n - n\right)$$

$$\frac{d}{dn}E(Profit) = \frac{1}{200}(4.5n - 800)$$

Now, we set the derivative equal to zero and solve for n:

$$\frac{1}{200}(4.5n - 800) = 0$$

$$4.5n - 800 = 0$$

$$4.5n = 800$$

$$n = \frac{800}{4.5}$$

$$n \approx 177.78$$

Therefore, to maximize profit, the shop should have approximately 178 eyeglasses.

Question ii:

Simulate 100 sales of the shop.

```
# Define parameters
n <- 178  # Number of eyeglasses
price_per_sale <- 1.00
loss_per_unhappy_customer <- 5.00
loss_per_extra_eyeglass <- 0.50
density_function <- function(x) ifelse(x > 0 & x < 200, 1/200, 0)  # Density function

# Simulate 100 sales
sales <- replicate(100, {
  # Generate random number of customers
  customers <- rpois(1, lambda = n)

  # Calculate expected number of sales
  expected_customers <- integrate(density_function, lower = 0, upper = Inf)$value
  expected_sales <- min(n, expected_customers)

  # Calculate profit and loss
  actual_sales <- min(n, customers)
  profit <- actual_sales * price_per_sale
  loss_unhappy_customers <- max(0, customers - actual_sales) * loss_per_unhappy_customer
  loss_surplus_eyeglasses <- max(0, n - customers) * loss_per_extra_eyeglass

  # Total profit after losses
  total_profit <- profit - loss_unhappy_customers - loss_surplus_eyeglasses

  return(total_profit)
})

# Calculate total profit over 100 sales
total_profit <- sum(sales)

# Print total profit
print(paste("Total profit over 100 sales:", total_profit))
```

Answer:

```
## [1] "Total profit over 100 sales: 14464.5"
```

Part ii:

For uniform (0,1) random variables $U_1, U_2, U_3, \ldots$, define

$$N = \text{minimum}\{n : \sum_{i=1}^{n} U_i > 1\}$$

That is, N is equal to the number of random numbers that must be summed to exceed 1.

Question i:

Estimate $E(N)$ by generating 100 values of N.

```r
# Define the number of simulations
num_simulations <- 100

# Function to simulate N
simulate_N <- function() {
  sum_U <- 0
  N <- 0
  while (sum_U <= 1) {
    sum_U <- sum_U + runif(1)
    N <- N + 1
  }
  return(N)
}

# Generate 100 values of N
N_values <- replicate(num_simulations, simulate_N())

# Estimate E(N)
estimated_mean_N <- mean(N_values)

# Print the estimated mean
print(paste("Estimated E(N) based on 100 simulations:", estimated_mean_N))
```

Answer:

```
## [1] "Estimated E(N) based on 100 simulations: 2.61"
```

Question ii:

Calculate the exact value of $E(N)$.

Answer: To calculate the exact value of $E(N)$, we need to find the expected value of the random variable N. Given the definition of N as the minimum number of random variables that must be summed to exceed 1, we can derive its probability distribution and then calculate the expected value.

Let's denote X_i as the i-th uniform (0,1) random variable, and let p_k be the probability that $N = k$. Then p_k is the probability that the sum of the first k random variables exceeds 1 while the sum of the first $k-1$ random variables is less than or equal to 1.

Therefore, we have:

$$p_k = P(N = k) = P(X_1 + X_2 + \ldots + X_k > 1, X_1 + X_2 + \ldots + X_{k-1} \leq 1)$$

$$= P(X_1 + X_2 + \ldots + X_k > 1) - P(X_1 + X_2 + \ldots + X_{k-1} > 1)$$

$$= (1 - P(X_1 + X_2 + \ldots + X_k \leq 1)) - (1 - P(X_1 + X_2 + \ldots + X_{k-1} \leq 1))$$

$$= P(X_1 + X_2 + \ldots + X_{k-1} < 1) - P(X_1 + X_2 + \ldots + X_{k-1} \leq 1)$$

$$= P(X_1 + X_2 + \ldots + X_{k-1} < 1) - P(X_1 + X_2 + \ldots + X_{k-1} < 1) + P(X_k < 1)$$

$$= P(X_k < 1)$$

Since X_i follows a uniform distribution on the interval (0,1), $P(X_i < 1) = 1$ for all i.

Therefore, $p_k = 1$ for all k.

Now, we have a geometric distribution with parameter $p = 1$, where p is the probability of success (i.e., $N = k$). In a geometric distribution, the expected value $E(N)$ is given by $E(N) = \frac{1}{p}$.

Since $p = 1$, we have $E(N) = \frac{1}{1} = 1$.

So, the exact value of $E(N)$ is 1.