Homework 3

M1399.000100, Seoul National University, Spring 2023

Due 23:00 Wed, 2024-05-22

Append your answer below each question. Submit the modified version of this `Rmd` file and the output `pdf` file, together with other necessary files such as images and R source code. The submitted version of this `Rmd` file should be `knitted` to an `pdf` file ideally identical to the submitted one.

When writing your own R code, do NOT use R packages that implement the functions you are asked to write. i.e., you must write your own code from scratch.

No late submission is accepted.

Q1. Iterative method

1. Write an R function `jacobi` implementing Jacobi's method for solving linear equation $\mathbf{Ax} = \mathbf{b}$, with interface

```
library(Matrix)
jacobi <- function(A, b, x0,  maxit=100, tol=1.0e-6, history=FALSE, verbose=FALSE) {
    nbrows <- ncol(A)
    x <- x0
    numit <- 0
    nrmdx <- Inf
    converged <- FALSE
    xhist <- NULL
    if (history)  {
        xhist <- vector("list", maxit)
    }
    while (numit < maxit) {
        numit <- numit + 1
        deltax <- ## PUT YOUR CODE HERE
        ## PUT YOUR CODE HERE
        x <- x + deltax
        nrmdx <- ## PUT YOUR CODE HERE
        if (verbose) {
            ## PUT YOUR CODE HERE
        }
        if (history) ## PUT YOUR CODE HERE
        if (nrmdx < tol) {
            ## PUT YOUR CODE HERE
        }
    }
```

```
    if (history) xhist <- xhist[seq_len(numit)]
    ret <- ## PUT YOUR CODE HERE
    ret
}
```

where `x0` is the initial iterate, `maxit` is the maximum number of iteration to run, `tol` is the tolerance for convergence test, and `history` and `verbose` are debugging flags; if `history == TRUE`, then your function shoud store each iterate x (excluding `x0`) in a list; if `verbose == TRUE`, then your function should print the error between the current and previous iterates as mesuared by the Euclidean norm.

The return value should be a `list` having four fields:

- `sol`: `numeric` object of length `ncol(A)` containing the last iterate of the method.
- `iter`: number of iterations ran
- `hist`: if `history == TRUE`, list of length `iter` containing all the iterates; otherwise `NULL`
- `converged`: `logical` variable indicating whether the final error is below `tol`

Assume that $\mathbf{A}$ is square and its diagonal elements are nonzero. Vectorize your code as much as possible.

Using the `get2DPoissonMatrix()` function defined in the Lecture Note on Iterative Methods for Solving Linear Equations (also provided below), generate the coefficient matrix as follows.

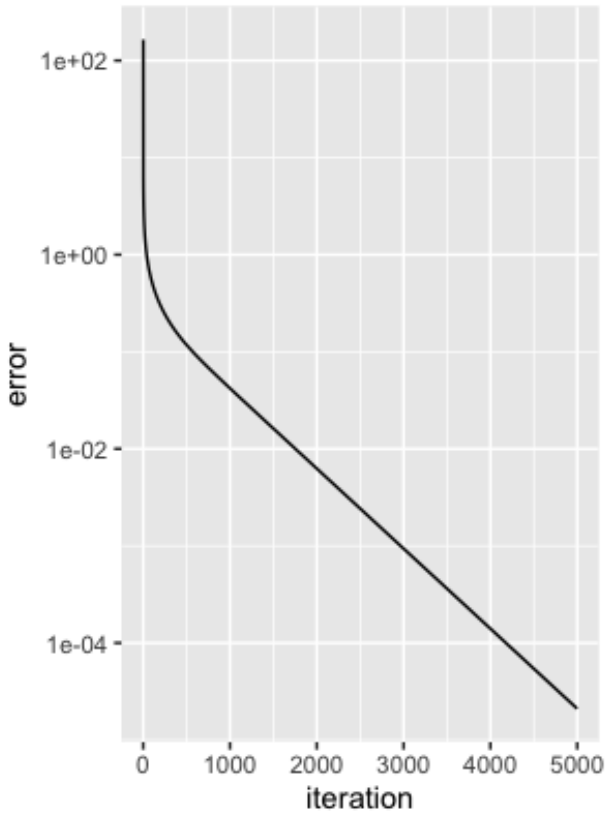
```
get1DPoissonMatrix <- function(n) {
  Matrix::bandSparse(n, n, #dimensions
                     (-1):1, #band, diagonal is number 0
                     list(rep(-1, n-1),
                     rep(4, n),
                     rep(-1, n-1)))
}

# Generate 2-dimensional discrete Poisson matrix
#
#' @param n Grid size. Output will be a n^2 by n^2 matrix.
get2DPoissonMatrix <- function(n) {  # n^n by n^n
    T <- get1DPoissonMatrix(n)
    eye <- Matrix::Diagonal(n)
    N <- n * n  ## dimension of the final square matrix
    ## construct two block diagonal matrices
    D <- bdiag(rep.int(list(T), n))
    O <- bdiag(rep.int(list(-eye), n - 1))

    ## augment O and add them together with D
    D +
     rbind(cbind(Matrix(0, nrow(O), n), O), Matrix(0, n, N)) +
     cbind(rbind(Matrix(0, n, ncol(O)), O), Matrix(0, N, n))
}

n <- 50
A <- get2DPoissonMatrix(n)
set.seed(456) # seed
b <- rnorm(n^2)
```

Then find the solution to `A %*% x = b` using your `jacobi()` function. In doing this, use the `hist` field of your return object to produce a convergence plot like the following.



2. Repeat the above for the successive over-relaxation (SOR) method, with interface

```
sor <- function(A, b, x0, w=1.1, maxit=100, tol=1.0e-6, history=FALSE, verbose=FALSE) {
    ## PUT YOUR CODE HERE
    while (numit < maxit) {
        ## PUT YOUR CODE HERE
    }
    ## PUT YOUR CODE HERE
}
```

Experiment with various values of the relaxation parameter `w`.

Discuss the difference between Jacobi's method and SOR (which includes the Gauss-Seidel) in terms of number of iterations to converge, wall-clock time of your code, etc. What do you think is the main reasons for the differences?

3. (PageRank) Now let us implement the PageRank algorithm using the SOR method. The Wikipedia Vote Network dataset, available at https://snap.stanford.edu/data/wiki-Vote.html, describes the voting network among the administrators of Wikipedia. The compressed text file `Wiki-Vote.txt.gz` in the SNAP website encodes this network in an edgelist format. The number of edges is 103689.

a) Convert the edgelist into an adjacency matrix. This matrix should be a sparse `dgCMatrix` provided in the `Matrix` package. Use the `readr::read_tsv()` function to read the edgelist into a `tbl`, a tidyverse dataframe. Then convert it into an `igraph` graph structure by using

```
vote <- igraph::graph_from_edgelist(as.matrix(df), directed=TRUE)
```

where `df` is the `tbl` for the imported edgelist. Finally,

```
A <- igraph::as_adjacency_matrix(vote, sparse=TRUE)
```

will give you the desired adjacency matrix. The resulting matrix should have a dimension 8297 by 8297.

b) Implement the PageRank algorithm using SOR. Since the `A` matrix is sparse and large, your code in Problem 2 is likely very inefficient. Recall that we need to solve

$$(\mathbf{I} - \mathbf{P}^T)\mathbf{x} = \mathbf{0}$$

where $\mathbf{P} = (p_{ij})$ with

$$p_{ij} = \begin{cases} (1-p)/n + pa_{ij}/r_i, & r_i > 0, \\ 1/n, & r_i = 0; \end{cases}$$

r_i is the out-degree of node i, `A`$= (a_{ij})$, and p is the probability of surfing from node i to j. Directly constructing the coefficient matrix from `A`, e.g.,

```
B <- Matrix::Diagonal(n)
for (i in which(outdeg > 0)) {
    for (j in seq_len(ncol(A))) {
        if (A[i,j] > 0) B[i,j] <- B[i,j] - (1 - p) / n - p * A[i,j] / outdeg[i]
    }
}
for (i in which(outdeg == 0)) {
    B[i,] <- B[i,] - 1/n
}
```

won't terminate easily in most of laptops. (This is because modifying a sparse matrix is not efficient.) Instead, you should implement multiplication of each row of $\mathbf{I} - \mathbf{P}^T$ with $\mathbf{x}$ *within* the SOR loop, by using only `A` and `p`.

Write an R function `pagerank` with interface

```
pagerank <- function(A, x0, p=0.85,  w=1.5, maxit=100, tol=1.0e-6, history=FALSE, verbose=FALSE) {
    nbrows <- ncol(A)
    r <- ## PUT YOUR CODE HERE. Compute out-degrees
    dangling <- which(r == 0) # dangling nodes
    ## PUT YOUR CODE HERE
    while (numit < maxit) {
        ## PUT YOUR CODE HERE
    }
    ## PUT YOUR CODE HERE
}
```

implementing this idea. Your return object should be the same as `sor()` above, but should additionally check nonnegativity of your solution to generate a warning "Method converged but the solution has negative elements." Write your function by completing the skeleton code above.

Apply your `pagerank()` to the Wikipedia Vote Network dataset with $p = 0.85$, convert your solution to probability, and report the top 10 nodes in terms of the PageRank score.

Q2. Contraction map

Recall that function $F : [a, b] \to [a, b]$ is *contractive* if there exists a nonnegative constant $L < 1$ such that

$$|F(x) - F(y)| \leq L|x - y| \tag{Lip}$$

for all $x, y \in [a, b]$.

1. Show that if F is differentiable on $[a, b]$, then condtion (Lip) is equivalent to

   $$|F'(x)| \leq L$$

   for all $x \in [a, b]$.

Now suppose we want to find a root of a differentiable function $f(x)$ on (a, b). Consider the following iteration

$$x^{(t+1)} = x^{(t)} + \alpha f(x^{(t)}) \quad (\alpha \neq 0). \tag{Iter}$$

2. When does iteration (Iter) converge?
3. Discuss the advantage of introducing the α.
4. Relate iteration (Iter) with the gradient descent method for minimization of a twice differentiable function.

Q3. Poisson regression

1. In the lecture note on optimization, we used step-halving in the Fisher scoring of the Poisson regression analysis of the quarterly count data of AIDS deaths in Australia. Repeat this using the **Armijo rule**. Use the interface

```
poissonreg <- function(x, y, maxiter=10, tol=1e-6, alpha=.2, gamma=.6) {
    ## YOUR CODE HERE
    while ((!stop) && (iter < maxiter)) {
        eta <- beta[1] + x * beta[2]
        w <- exp(eta)  # lambda
        # line search by Armijo rule
        s <- 1.0
        while (TRUE) {
            ## YOUR CODE HERE
        }
        print(c(as.numeric(beta), inneriter[iter], curlik))
        betadiff <- beta - betaold
        if (norm(betadiff, "F") < tol) stop <- TRUE
        likold <- curlik
        betaold <- beta
        iter <- iter + 1
    }
    return(list(val=as.numeric(beta), iter=iter, inneriter=inneriter[1:iter]))
}
```

   where `alpha` and `gamma` corresponds to the Armijo rule parameters α and β, respectively. Also, execute the following code.

```r
# Test code
deaths <- c(0, 1, 2, 3, 1, 4, 9, 18, 23, 31, 20, 25, 37, 45)
quarters <- seq_along(deaths)
poissonreg(quarters, deaths)$val
```

2. In the same lecture note, it is stated that Poisson regression has the objective function $f(\beta) = -\sum_{i=1}^{n} \left[y_i \mathbf{x}_i^T \beta - \exp(\mathbf{x}_i^T \beta) - \log(y_i!)\right]$ and its gradient

$$\begin{aligned}\nabla f(\beta) &= -\sum_{i=1}^{n} \left(y_i \mathbf{x}_i - \exp(\mathbf{x}_i^T \beta)\mathbf{x}_i\right) \\ &= -\sum_{i=1}^{n} (y_i - \lambda_i)\mathbf{x}_i = -\mathbf{X}^T(\mathbf{y} - \boldsymbol{\lambda})\end{aligned}$$

where $\mathbf{X}$ is the design matrix, $\mathbf{y}$ is the response

is *not* Lipschitz continuous. Show this.

Q4. Gradient descent

1. In K-class logistic regression, we observe n independent categorical variables with $K > 2$ categories, each associated with covariates. Let $\mathbf{y}_i$ be the ith observation and $\mathbf{x}_i \in \mathbf{R}^p$ be the associated covariate vector. We use a dummy variable encoding for the response so that $\mathbf{y}_i$ is a K-dimensional binary vector with only one component being 1. Let p_k be the probability of the kth category. We want to model the logit transform of p_k as a linear function of the covariate $\mathbf{x}_i$. Due to the constraint $\sum_{k=1}^{K} p_k = 1$, we set

$$\log \frac{p_k}{p_K} = \mathbf{x}_i^T \boldsymbol{\beta}_k, \quad k = 1, \ldots, K-1$$

and $\boldsymbol{\beta}_K = 0$.

a. Express the log likelihood of the data in terms of the matrices $\mathbf{Y} = (y_{ik}) \in \mathbb{R}^{n \times K}$ and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$ or their components. You may ignore terms irrelevant to $\boldsymbol{\beta}_k$.
b. Write down a gradient ascent step for estimating coefficients $\mathbf{B} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{K-1}]$.

Q5. IRLS

1. A researcher is interested in how variables, such as GRE (Graduate Record Exam scores), GPA (grade point average) and prestige of the undergraduate institution, effect admission into graduate school. The response variable, admit/don't admit, is a binary variable.
The data is available at https://stats.idre.ucla.edu/stat/data/binary.csv. How to analyze these data can be found in the website https://stats.idre.ucla.edu/r/dae/logit-regression/. You can use the following code to load the data.

```r
mydata <- read.csv(url("https://stats.idre.ucla.edu/stat/data/binary.csv"))
```

Implement the iteratively reweighted least squares (IRLS) algorithm for fitting a logistic regression model, and apply your algorithm to the admission data above. Use the interface below.

```r
logistic <- function(y, X, beta0=NULL, maxiter=500, tol=1e-8) {
    ## PUT YOUR CODE HERE
    stop <- FALSE
    while ((!stop) && (iter < maxiter)) {
        ## PUT YOUR CODE HERE
    }
    ## PUT YOUR CODE HERE
}
```

The return object should be a list comprised of `coef`, containing the fitted regression coefficient, and `iter`, the number of iterations ran. Note that the variable `rank` is categorical data.

Compare your result with `glm()` function in R. You can use the following code.

```r
logistic(mydata$admit,
         cbind(1, mydata$gre, mydata$gpa, mydata$rank), maxiter=500)$coef
glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")$coefficients
```