Homework 2

Steven Xu

Due @ 11:59pm on October 1, 2018

Part 1. Let's investigate the limiting behavior of the power method you implemented in Homework 1. Suppose we have a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ that is symmetric and has eigendecomposition $\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\mathsf{T}$ and that $\mathbf{A}$ possesses a **unique** largest in magnitude eigenvalue. The eigendecomposition has the following properties:

1. $\mathbf{U} \in \mathbb{R}^{n \times n}$ has orthonormal columns, i.e. $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$, and therefore the columns $\mathbf{u}_1, \ldots, \mathbf{u}_n$ form a basis for $\mathbb{R}^n$
2. Without loss of generality, $\boldsymbol{\Lambda}$ is a diagonal matrix such that $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$.

The power method iterates as follows.

$$\mathbf{x}^{(k+1)} \leftarrow \frac{\mathbf{A}\mathbf{x}^{(k)}}{\|\mathbf{A}\mathbf{x}^{(k)}\|_2}$$

Suppose we start with a vector $\mathbf{x}^{(0)}$ such that $\mathbf{u}_1^\mathsf{T}\mathbf{x}^{(0)} \neq 0$.

1. Argue that $\mathbf{x}^{(0)} = c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \cdots + c_n\mathbf{u}_n$ for some vector $\mathbf{c} \in \mathbb{R}^n$. Write the vector $\mathbf{c}$ as a function of $\mathbf{U}$ and $\mathbf{x}^{(0)}$. In other words, how do you compute $\mathbf{c}$ from $\mathbf{U}$ and $\mathbf{x}^{(0)}$?

Answer:

We know that the columns $\mathbf{u}_1, \ldots, \mathbf{u}_n$ form a basis for $\mathbb{R}^n$, i.e. $C(\mathbf{U}) = \mathbb{R}^n$. Since $\mathbf{x}^{(0)} \in \mathbb{R}^n$, we have $\mathbf{x}^{(0)} \in C(\mathbf{U})$, i.e. $\exists\ \mathbf{c} \in \mathbb{R}^n$ s.t. $\mathbf{x}^{(0)} = \mathbf{U}\mathbf{c}$ or $\mathbf{x}^{(0)} = \sum_{i=1}^n c_i\mathbf{u}_i$. Pre-multiply both sides by $\mathbf{U}^\mathsf{T}$ we have $\mathbf{c} = \mathbf{U}^\mathsf{T}\mathbf{x}^{(0)}$.

2. Show that

$$\mathbf{x}^{(k)} = \frac{c_1\mathbf{u}_1 + \sum_{j=2}^n c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u}_j}{\left\| c_1\mathbf{u}_1 + \sum_{j=2}^n c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u}_j \right\|_2} \mathbf{sign}\,(\lambda_1)^k,$$

where

$$\mathbf{sign}\,(\lambda) = \begin{cases} 1 & \text{if } \lambda > 0 \\ -1 & \text{if } \lambda < 0 \\ 0 & \text{if } \lambda = 0. \end{cases}$$

Answer:

Prove by induction:

When k=1,

$$\mathbf{x}^{(1)} = \frac{\mathbf{A}\mathbf{x}^{(0)}}{\|\mathbf{A}\mathbf{x}^{(0)}\|_2}$$

$$= \frac{\mathbf{A}\sum_{i=1}^{n} c_i \mathbf{u_i}}{\|\mathbf{A}\sum_{i=1}^{n} c_i \mathbf{u_i}\|_2}$$

$$= \frac{\sum_{i=1}^{n} c_i \mathbf{A}\mathbf{u_i}}{\|\sum_{i=1}^{n} c_i \mathbf{A}\mathbf{u_i}\|_2}$$

$$\because \mathbf{u}_1, \ldots, \mathbf{u}_n \text{ are eigenvectors of } \mathbf{A}$$

$$\therefore \mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i \ \forall \ i = 1, \ldots, n$$

$$\mathbf{x}^{(1)} = \frac{\sum_{i=1}^{n} c_i \lambda_i \mathbf{u}_i}{\left\|\sum_{i=1}^{n} c_i \lambda_i \mathbf{u}_i\right\|_2}$$

$$= \frac{c_1\lambda_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \lambda_j \mathbf{u}_j}{\left\|c_1\lambda_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \lambda_j \mathbf{u}_j\right\|_2}$$

$$= \frac{\frac{1}{\lambda_1}(c_1\lambda_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \lambda_j \mathbf{u}_j)}{\frac{1}{\lambda_1}\left\|c_1\lambda_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \lambda_j \mathbf{u}_j\right\|_2}$$

$$\text{By construction } \frac{1}{\lambda_1} = \mathbf{sign}\,(\lambda_1)|\frac{1}{\lambda_1}|, \text{ assuming } \lambda_1 \neq 0$$

$$\therefore\ \mathbf{x}^{(1)} = \frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)\mathbf{u}_j}{|\frac{1}{\lambda_1}|\left\|c_1\lambda_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \lambda_j \mathbf{u}_j\right\|_2}\mathbf{sign}\,(\lambda_1)$$

$$= \frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)\mathbf{u}_j}{\left\|c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)\mathbf{u}_j\right\|_2}\mathbf{sign}\,(\lambda_1)$$

So the equation holds when k=1.

Suppose it holds when k=m, i.e.

$$\mathbf{x}^{(m)} = \frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^m \mathbf{u}_j}{\left\| c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^m \mathbf{u}_j \right\|_2} \mathbf{sign}\left(\lambda_1\right)^m$$

$$\text{Let } K = \left\| c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^m \mathbf{u}_j \right\|_2$$

$$\mathbf{x}^{(m+1)} = \frac{\mathbf{A}\mathbf{x}^{(m)}}{\|\mathbf{A}\mathbf{x}^{(m)}\|_2}$$

$$\begin{aligned}
\mathbf{A}\mathbf{x}^{(m)} &= \mathbf{A}\frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^m \mathbf{u}_j}{K} \mathbf{sign}\left(\lambda_1\right)^m \\
&= \frac{c_1\lambda_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^m \lambda_j\mathbf{u}_j}{K} \mathbf{sign}\left(\lambda_1\right)^m \\
&= \frac{\lambda_1(c_1\ \mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j)}{K} \mathbf{sign}\left(\lambda_1\right)^m
\end{aligned}$$

$$\because \sqrt{\mathbf{sign}\left(\lambda_1\right)^2} = 1,\ K > 0$$

$$\therefore \sqrt{\left(\frac{\mathbf{sign}\left(\lambda_1\right)}{K}\right)^2} = \frac{1}{K}$$

$$\begin{aligned}
\|\mathbf{A}\mathbf{x}^{(m)}\|_2 &= \frac{1}{K}\|\lambda_1(\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j)\|_2 \\
&= \frac{|\lambda_1|}{K}\|\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j\|_2
\end{aligned}$$

$$\begin{aligned}
\therefore\ \mathbf{x}^{(m+1)} &= \frac{\frac{\lambda_1}{K}(c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j)}{\frac{|\lambda_1|}{K}\left\| c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j \right\|_2} \mathbf{sign}\left(\lambda_1\right)^m \\
&= \frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j}{\left\| c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^{m+1} \mathbf{u}_j \right\|_2} \mathbf{sign}\left(\lambda_1\right)^{m+1}
\end{aligned}$$

Thus by induction the equality holds $\forall\ k \geq 1$.

3. Argue that

$$\|\mathbf{x}^{(k)} - \mathbf{sign}\,(c_1)\mathbf{sign}\,(\lambda_1)^k \mathbf{u}_1\|_2 \to 0.$$

Hint: Use the fact from 2 above and the triangle inequality.

Answer:

$$\|\mathbf{x}^{(k)} - \mathbf{sign}\,(c_1)\mathbf{sign}\,(\lambda_1)^k \mathbf{u}_1\|_2 = \left\| \frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u}_j}{\left\| c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u}_j \right\|_2} \mathbf{sign}\,(\lambda_1)^k - \mathbf{sign}\,(c_1)\mathbf{sign}\,(\lambda_1)^k \mathbf{u}_1 \right\|_2$$

$$= \left\| \frac{c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u}_j}{\left\| c_1\mathbf{u}_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u}_j \right\|_2} - \mathbf{sign}\,(c_1)\mathbf{u}_1 \right\|_2$$

$$\because |\lambda_1| > |\lambda_j| \ \forall\ j = 2, \ldots, n$$

$$\therefore \left(\frac{\lambda_j}{\lambda_1}\right)^k \to 0 \text{ as k } \to \infty\ \forall\ j = 2, \ldots, n$$

$$\therefore \|\mathbf{x}^{(k)} - \mathbf{sign}\,(c_1)\mathbf{sign}\,(\lambda_1)^k \mathbf{u}_1\|_2 \to \left\| \frac{c_1\mathbf{u}_1}{\|c_1\mathbf{u}_1\|_2} - \mathbf{sign}\,(c_1)\mathbf{u}_1 \right\|_2$$

Since $\mathbf{U}$ is an orthonormal matrix, $\|\mathbf{u_1}\|_2 = 1$

$$\therefore \|\mathbf{x}^{(k)} - \mathbf{sign}\,(c_1)\mathbf{sign}\,(\lambda_1)^k \mathbf{u}_1\|_2 \to \left\| \frac{c_1\mathbf{u}_1}{|c_1|} - \mathbf{sign}\,(c_1)\mathbf{u}_1 \right\|_2 = \|\mathbf{sign}\,(c_1)\mathbf{u}_1 - \mathbf{sign}\,(c_1)\mathbf{u}_1\|_2 = 0$$

$$\therefore \|\mathbf{x}^{(k)} - \mathbf{sign}\,(c_1)\mathbf{sign}\,(\lambda_1)^k \mathbf{u}_1\|_2 \to 0 \text{ as } k \to \infty$$

Part 2. The Sweep Operator

You will next add an implementation of the sweep operator to your R package. For the following functions, save them all in a file called `homework2.R` and put this file in the R subdirectory of your package.

Please complete the following steps.

Step 1: Write a function `sweep_k` that applies the sweep operator to a symmetric matrix on the kth diagonal entry if possible. It should return an error message if

- the kth diagonal entry is not positive
- the input matrix is not symmetric

```
#' Sweep k
#'
#' \code{sweep_k} applies the sweep operator to a symmetric matrix
#' on the kth diagonal entry if it is possible.
#'
#' @param A The input symmetric matrix
#' @param k Diagonal index on which to sweep
#' @export
# sweep_k <- function(A, k) {
#
# }
```

Your function should return the matrix $\hat{\mathbf{A}} = \text{sweep}(\mathbf{A}, k)$ if it is possible to sweep.

Step 2: Write a function `isweep_k` that applies the inverse sweep operator to a symmetric matrix on the kth diagonal entry if possible. It should return an error message if

- the kth diagonal entry is not negative
- the input matrix is not symmetric

```
#' Inverse Sweep k
#'
#' \code{isweep_k} applies the inverse sweep operator to a symmetric matrix
#' on the kth diagonal entry if it is possible.
#'
#' @param A The input symmetric matrix
#' @param k Diagonal index on which to sweep
#' @export
# isweep_k <- function(A, k) {
#
# }
```

Your function should return the matrix $\hat{\mathbf{A}} = \text{sweep}^{-1}(\mathbf{A}, k)$ if it is possible to sweep.

Step 3: Write a function `sweep` that is a wrapper function for your `sweep_k` function. This function should apply the sweep operator on a specified set of diagonal entries. It should return an error message if

- the kth diagonal entry is not positive
- the input matrix is not symmetric

Also, it should by default, if a set of diagonal entries is not specified, apply the Sweep operator on all diagonal entries (if possible).

```
#' Sweep
#'
#' @param A The input symmetric matrix
#' @param k Diagonal index entry set on which to sweep
#' @export
# sweep <- function(A, k=NULL) {
#
# }
```

Step 4: Write a function `isweep` that is a wrapper function for your `isweep_k` function. This function should apply the sweep operator on a specified set of diagonal entries. It should return an error message if

- the kth diagonal entry is not negative
- the input matrix is not symmetric

Also, it should by default, if a set of diagonal entries is not specified, apply the Inverse Sweep operator on all diagonal entries (if possible).

```
#' Inverse Sweep
#'
#' @param A The input symmetric matrix
#' @param k Diagonal index entry set on which to sweep
#' @export
# isweep <- function(A, k=NULL) {
#
# }
```

Step 4: Write a unit test function `test-sweep` that

- checks that `sweep_k` and `isweep_k` do indeed undo the effects of each other.
- checks that `sweep` and `isweep` do indeed undo the effects of each other.
- checks the correctness of your sweep operator on a small random positive definite matrix $\mathbf{A}$. To create such a matrix, use something like the following code:

  ```
  n <- 10
  u <- matrix(rnorm(n), ncol=1)
  A <- tcrossprod(u)
  diag(A) <- diag(A) + 1
  ```

Recall that $\text{sweep}(\mathbf{A}, 1:n) = -\mathbf{A}^{-1}$. So, to check for correctness, use the following measure:

$$\|\mathbf{A} \times \text{sweep}(\mathbf{A}, 1:n) + \mathbf{I}\| \approx 0.$$

What constitutes approximately zero? That's a very good question. Please try experimenting with different tolerances to find a tolerance that correctly identifies when your implementation of the sweep operator is working. If you experiment enough, you will get a sense for when the code is correct up to "numerical noise."

Step 5: Use your `sweep` function to compute the regression coefficients in the following multiple linear regression problem. Let's keep things simple and not include the intercept in the model. Compare your answers to what is provided by the `lm` function in R.

```
set.seed(12345)
n <- 1000
p <- 10
X <- matrix(rnorm(n*p), n, p)
beta <- matrix(rnorm(p), p, 1)
y <- X%*%beta + matrix(rnorm(n), n, 1)
A = matrix(rep(0,(p+1)^2),nrow=(p+1))
A[1:p,1:p] = t(X)%*%X
A[1:p,p+1] = t(X)%*%y
A[p+1,1:p] = t(y)%*%X
A[p+1,p+1] = t(y)%*%y
sol = sweep(A,1:p)
fit= lm(y~0+X)
lm_coef = matrix(as.numeric(fit$coefficients),ncol=1)
sw_coef = as.matrix(sol[1:p,p+1],ncol=1)
norm(lm_coef-sw_coef)
```

```
## [1] 5.259682e-15
```

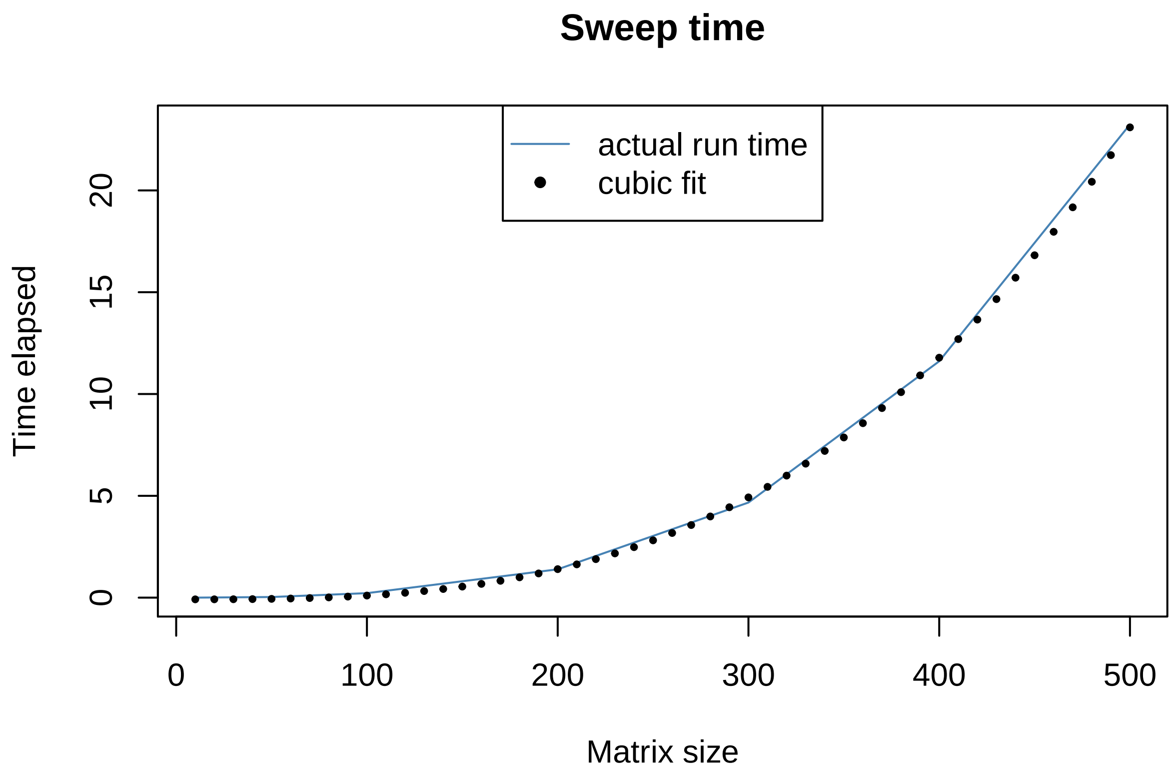
We can see that the sweep operator performs really well. The difference is on a scale of 10^{-15} for this example.

Step 6: Apply your `sweep` function on symmetric matrices of sizes $n = 100, 200, 300, 400$ and plot the run times versus n. To get the run time use `system.out`. You can use the third argument of the output of `system.out`. Does the run time scale as you'd expect?

```
x = c(10,50,seq(100,500,100))
time = function(size){
  n <- size
  u <- matrix(rnorm(n), ncol=1)
  A <- tcrossprod(u)
  diag(A) <- diag(A) + 1
  t = system.time(sweep(A))[3]
  return(t)
}
time_vec = as.numeric(sapply(x,time))
comp = lm(time_vec~I(x^3))
summary(comp)$r.squared
```

```
## [1] 0.9996837
```

```
plot(x,time_vec,type="l",xlab = "Matrix size", ylab = "Time elapsed", main = "Sweep time", col= "Steel 
x_new = data.frame(x=seq(10,500,10))
points(seq(10,500,10),predict(comp,newdata=x_new),pch=20,cex=0.6)
legend("top",legend=c("actual run time","cubic fit"),lty=c(1,NA),pch=c(NA, 20) ,col=c("steel blue","bla
```



From class we know that the computation complexity to sweep a whole matrix is of $O(n^3)$. By fitting a cubic linear regression we find R^2 to be almost 1. Together with the fitted line we see that the computational complexity is indeed of $O(n^3)$.