Data 605 Homework 2

Jean Jimenez

2023-09-07

Problem Set 2

The second problem for this week's homework was to create a function that will conduct LU factorization on a square matrix.

Function

I created a function that does the LU Factorization of any 2x2, 3x3, or 4x4 matrix.

My function is called lu_decomp:

```
lu_decomp = function(A) {

  #Getting the dimensions of matrix

  n = dim(A)[1]

  #checking if matrix is square
  if (n != dim(A)[2]) {
    stop("The matrix must be square.")
  }

  #establishing upper and lower triangle matrix (to be filled)
  L = diag(n)
  U = matrix(0, nrow = n, ncol = n)

  #hard coding each LU decomp individually for 2x2 3x3 and 4x4


  switch(as.character(n),
         #2x2
         '2' = {
           L[2, 1] = A[2, 1] / A[1, 1]
           U[1,] = A[1,]
           U[2, 2] = A[2, 2] - L[2, 1] * A[1, 2]
         },
         #3x3
         '3' = {
           L[2, 1] = A[2, 1] / A[1, 1]
           L[3, 1] = A[3, 1] / A[1, 1]
```

```r
          L[3, 2] = (A[3, 2] - L[3, 1] * A[1, 2]) / A[2, 2]

          U[1, ] = A[1, ]
          U[2, 2:3] = A[2, 2:3] - L[2, 1] * A[1, 2:3]
          U[3, 3] = A[3, 3] - L[3, 1] * A[1, 3] - L[3, 2] * A[2, 3]
          U[2, 1] = A[2, 1]
          U[3, 1:2] = A[3, 1:2] - L[3, 1] * A[1, 1:2] - L[3, 2] * A[2, 1:2]
        },
        #4x4
        '4' = {
          L[2, 1] = A[2, 1] / A[1, 1]
          L[3, 1] = A[3, 1] / A[1, 1]
          L[4, 1] = A[4, 1] / A[1, 1]

          L[3, 2] = (A[3, 2] - L[3, 1] * A[1, 2]) / A[2, 2]
          L[4, 2] = (A[4, 2] - L[4, 1] * A[1, 2]) / A[2, 2]

          L[4, 3] = (A[4, 3] - L[4, 1] * A[1, 3] - L[4, 2] * A[2, 3]) / A[3, 3]

          U[1,] = A[1,]
          U[2, 2:4] = A[2, 2:4] - L[2, 1] * A[1, 2:4]
          U[3, 3:4] = A[3, 3:4] - L[3, 1] * A[1, 3:4] - L[3, 2] * A[2, 3:4]
          U[4, 4] = A[4, 4] - L[4, 1] * A[1, 4] - L[4, 2] * A[2, 4] - L[4, 3] * A[3, 4]
          U[2, 1] = A[2, 1]
          U[3, 1:2] = A[3, 1:2] - L[3, 1] * A[1, 1:2] - L[3, 2] * A[2, 1:2]
          U[4, 1:3] = A[4, 1:3] - L[4, 1] * A[1, 1:3] - L[4, 2] * A[2, 1:3] - L[4, 3] * A[3, 1:3]
        },
        stop("This function only works for 2x2, 3x3, and 4x4 matrices.")
  )

  # calculate LXU and checking results to see if its close enough to A

  LU = L %*% U

  check_result = all.equal(A, LU, tolerance = 3)

  if (!isTRUE(check_result)) {
    stop(paste("Decomposition failed: ", check_result))
  }

  print("Matrix L:")
  print(L)

  print("Matrix U:")
  print(U)

  return(list(L = L, U = U))

}
```

How it Works

My function lu_decomp can accept any 2x2, 3x3, or 4x4 matrix.

First, the function will get the dimension of the matrix entered . The function will also check if the matrix is a square (if it isn't, it stops/errors). Afterwards, I defined the L matrix (starting with the Identity matrix ones on diagonal.) and Matrix U filled with zeros (future upper triangle matrix). Then I hard coded and filled in column by column L and U based on the position of each value in A. I mapped it by hand on paper . After I had L and U filled up, I added a check to see if my function actually works. What it does is that it checks if LxU==A (or if it is close enough). Note that I added a few degrees of freedom. My code isn't perfect and since I randomly made matrices to run through it I have to add some degrees of freedom to account for the not so perfect (and possibly non integer) examples of LU decomposition.

My function then prints out L and U for the user to see and also returns L and U matrix in a list.

Testing it out

I tested out my function on a few 2x2, 3x3, and 4x4 cases to make sure that it worked. The following are the test cases that I used:

```
# 2x2 Test Cases
A1 = matrix(c(2, 1, 1, 2), nrow = 2)
result1 = lu_decomp(A1)
```

2x2

```
## [1] "Matrix L:"
##      [,1] [,2]
## [1,]  1.0    0
## [2,]  0.5    1
## [1] "Matrix U:"
##      [,1] [,2]
## [1,]    2  1.0
## [2,]    0  1.5
```

```
A2 = matrix(c(3, 4, 2, 1), nrow = 2)
result2 = lu_decomp(A2)
```

```
## [1] "Matrix L:"
##          [,1] [,2]
## [1,] 1.000000    0
## [2,] 1.333333    1
## [1] "Matrix U:"
##      [,1]      [,2]
## [1,]    3  2.000000
## [2,]    0 -1.666667
```

```
A3 = matrix(c(5, 2, 3, 4), nrow = 2)
result3 = lu_decomp(A3)
```

```
## [1] "Matrix L:"
##      [,1] [,2]
## [1,]  1.0    0
```

```
## [2,]  0.4    1
## [1] "Matrix U:"
##      [,1] [,2]
## [1,]    5  3.0
## [2,]    0  2.8
```

```r
A4 = matrix(c(1, 3, 2, 6), nrow = 2)
result4 = lu_decomp(A4)
```

```
## [1] "Matrix L:"
##      [,1] [,2]
## [1,]    1    0
## [2,]    3    1
## [1] "Matrix U:"
##      [,1] [,2]
## [1,]    1    2
## [2,]    0    0
```

```r
# 3x3 Test Cases
A5 = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 10), nrow = 3)
result5 = lu_decomp(A5)
```

3x3

```
## [1] "Matrix L:"
##      [,1] [,2] [,3]
## [1,]    1  0.0    0
## [2,]    2  1.0    0
## [3,]    3 -1.2    1
## [1] "Matrix U:"
##      [,1] [,2] [,3]
## [1,]  1.0    4  7.0
## [2,]  2.0   -3 -6.0
## [3,]  2.4    0 -1.4
```

```r
A6 = matrix(c(2, 1, 4, 3, 2, 1, 1, 2, 3), nrow = 3)
result6 = lu_decomp(A6)
```

```
## [1] "Matrix L:"
##      [,1] [,2] [,3]
## [1,]  1.0  0.0    0
## [2,]  0.5  1.0    0
## [3,]  2.0 -2.5    1
## [1] "Matrix U:"
##      [,1] [,2] [,3]
## [1,]  2.0  3.0  1.0
## [2,]  1.0  0.5  1.5
## [3,]  2.5  0.0  6.0
```

```r
A7 = matrix(c(9, 6, 3, 4, 7, 1, 2, 1, 8), nrow = 3)
result7 = lu_decomp(A7)
```

```
## [1] "Matrix L:"
##           [,1]        [,2] [,3]
## [1,] 1.0000000  0.00000000    0
## [2,] 0.6666667  1.00000000    0
## [3,] 0.3333333 -0.04761905    1
## [1] "Matrix U:"
##           [,1]     [,2]       [,3]
## [1,] 9.0000000 4.000000  2.0000000
## [2,] 6.0000000 4.333333 -0.3333333
## [3,] 0.2857143 0.000000  7.3809524
```

```r
A8 = matrix(c(2, 4, 1, 7, 5, 3, 1, 1, 1), nrow = 3)
result8 = lu_decomp(A8)
```

```
## [1] "Matrix L:"
##      [,1] [,2] [,3]
## [1,]  1.0  0.0    0
## [2,]  2.0  1.0    0
## [3,]  0.5 -0.1    1
## [1] "Matrix U:"
##      [,1] [,2] [,3]
## [1,]  2.0    7  1.0
## [2,]  4.0   -9 -1.0
## [3,]  0.4    0  0.6
```

```r


# 4x4 Test Cases
A9 = matrix(c(1, 1, 1, 1, 4, 4, 7, 2, 3, 4, 4, 1, 1, 2, 3, 4), nrow = 4)
result9 = lu_decomp(A9)
```

4x4

```
## [1] "Matrix L:"
##      [,1]  [,2] [,3] [,4]
## [1,]    1  0.00    0    0
## [2,]    1  1.00    0    0
## [3,]    1  0.75    1    0
## [4,]    1 -0.50    0    1
## [1] "Matrix U:"
##       [,1] [,2] [,3] [,4]
## [1,]  1.00    4    3  1.0
## [2,]  1.00    0    1  1.0
## [3,] -0.75    0   -2  0.5
## [4,]  0.50    0    0  4.0
```

```r
A10 = matrix(c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4), nrow = 4)
result10 = lu_decomp(A10)
```

```
## [1] "Matrix L:"
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    1    1    0    0
## [3,]    1    0    1    0
## [4,]    1    0    0    1
## [1] "Matrix U:"
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    1    0    0    0
## [3,]    0    0    0    0
## [4,]    0    0    0    0
```

```r
A11 = matrix(c(1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7), nrow = 4)
result11 = lu_decomp(A11)
```

```
## [1] "Matrix L:"
##      [,1]       [,2] [,3] [,4]
## [1,]    1  0.0000000  0.0    0
## [2,]    2  1.0000000  0.0    0
## [3,]    3 -0.6666667  1.0    0
## [4,]    4 -1.0000000 -0.4    1
## [1] "Matrix U:"
##          [,1] [,2]      [,3]      [,4]
## [1,] 1.000000  2.0  3.000000  4.000000
## [2,] 2.000000 -1.0 -2.000000 -3.000000
## [3,] 1.333333  0.0 -1.333333 -2.666667
## [4,] 3.200000  1.6  0.000000 -1.600000
```

```r
A12=matrix(c(11, 12, 13, 14, 12, 13, 14, 15, 13, 14, 15, 16, 14, 15, 16, 17), nrow = 4)
result12 = lu_decomp(A12)
```

```
## [1] "Matrix L:"
##          [,1]        [,2]        [,3] [,4]
## [1,] 1.000000  0.00000000  0.00000000    0
## [2,] 1.090909  1.00000000  0.00000000    0
## [3,] 1.181818 -0.01398601  1.00000000    0
## [4,] 1.272727 -0.02097902 -0.01678322    1
## [1] "Matrix U:"
##            [,1]        [,2]       [,3]       [,4]
## [1,] 11.0000000 12.00000000 13.0000000 14.0000000
## [2,] 12.0000000 -0.09090909 -0.1818182 -0.2727273
## [3,]  0.1678322  0.00000000 -0.1678322 -0.3356643
## [4,]  0.4699301  0.23496503  0.0000000 -0.2349650
```