刘程华 2018011687 计91

1、Verify that the flop count is $\sim 3mn^2 - n^3$ for Givens QR on $p.42$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} \sqrt{x_i^2 + x_j^2} \\ 0 \end{bmatrix} \cos\theta = \frac{x_i}{\sqrt{x_i^2+x_j^2}}, \quad \sin\theta = \frac{-x_j}{\sqrt{x_i^2+x_j^2}}$$

It takes six flop(two squares, one root sign, one addition, two divisions) to transform an element into 0.

So, flop count $= \sum_{k=1}^{n} 6(m-k)(n-k) = 6\sum_{k=1}^{n}\left[mn - (m+n)k + k^2\right] \sim 3mn^2 - n^3$

2、Edit your R functions for algorithms of CGS, MGS, Householder transformation and Givens rotation. Use numerical examples to illustrate the behavior of these methods. For instance, take the 25×15 Vandermonde matrix $A = \left(p_i^{j-1}\right)$, where the $p_i's$ are equally spaced on $[0, 1]$. What is the condition number of A? Of $A'A$? What are the errors in terms of the $\|A - \tilde{Q}\tilde{R}\|$ and $\left\|I - \tilde{Q}^T\tilde{Q}\right\|$?

Code:

```
#CGS
classical_gram_schmidt<-function (A)
{
  stopifnot(is.numeric(A), is.matrix(A))
  m <- nrow(A)
  n <- ncol(A)
  if (m < n)
    stop(" rows of must be greater or equal colums! ")
  Q <- matrix(0, m, n)
  R <- matrix(0, n, n)
  for (k in 1:n) {
    Q[, k] <- A[, k]
    if (k > 1) {
      for (i in 1:(k - 1)) {
        R[i, k] <- t(Q[, i]) %*% Q[, k]
        Q[, k] <- Q[, k] - R[i, k] * Q[, i]
      }
    }
    R[k, k] <- (t(Q[, k]) %*% Q[, k ])^0.5
    Q[, k] <- Q[, k]/R[k, k]
  }
  return(list(Q = Q, R = R))
}
#MGS
```

```r
modified_gram_schmidt<-function (A)
{
  stopifnot(is.numeric(A), is.matrix(A))
  m <- nrow(A)
  n <- ncol(A)
  if (m < n)
    stop(" rows of must be greater or equal colums! ")
  Q <- matrix(0, m, n)
  R <- matrix(0, n, n)
  for (k in 1:n) {
    Q[, k] <- A[, k]
  }
  for (i in 1:n) {
    R[i, i] <- (t(Q[, i]) %*% Q[, i])^0.5
    Q[, i] <- Q[, i]/R[i, i]
    if(i<n)
      for(j in (i+1):n){
        R[i,j]<- (t(Q[, i]) %*% Q[, j])
        Q[, j]<-Q[, j]-R[i,j]*Q[,i]
      }
  }
  return(list(Q = Q, R = R))
}
#Householder
householder <- function (A)
{
    m <- nrow(A)
    n <- ncol(A)
    Q <- eye(m)
    for (k in 1:min(m - 1, n)) {
        ak <- A[k:m, k, drop = FALSE]
        s <- if (ak[1] >= 0)
            1
        else -1
        vk <- ak + s * Norm(ak) * c(1, rep(0, m - k))
        vk2 <- c(t(vk) %*% vk)
        Hk <- eye(m - k + 1) - 2/vk2 * (vk %*% t(vk))
        if (k == 1)
            Qk <- Hk
        else Qk <- blkdiag(eye(k - 1), Hk)
        A <- Qk %*% A
        Q <- Q %*% Qk
    }
    return(list(Q = Q, R = A))
}

#Givens
_givens <- function(xa, xb) {
    if (xb != 0) {
```

```
        G <- matrix(c(xa, -xb, xb, xa), 2, 2) / norm(c(xa, xb))
        x <- as.matrix(c(r, 0))
    } else {
        G <- eye(2)
        x <- as.matrix(c(xa, 0))
    }
    return(list(G = G, x = x))
}
givens <- function (A)
{
    stopifnot(is.numeric(A), is.matrix(A))
    n <- nrow(A)
    m <- ncol(A)
    if (n != m)
        stop("Matrix 'A' must be a square matrix.")
    Q <- eye(n)
    for (k in 1:(n - 1)) {
        l <- which.max(abs(A[(k + 1):n, k])) + k
        if (A[k, k] == 0 && A[l, k] == 0)
            stop("Matrix 'A' does not have full rank.")
        j <- which(A[(k + 1):n, k] != 0) + k
        j <- unique(c(l, j[j != 1]))
        for (h in j) {
            gv <- _givens(A[k, k], A[h, k])
            G <- gv$G
            x <- gv$x
            Q[c(k, h), ] <- G %*% Q[c(k, h), ]
            A[k, k] <- x[1]
            A[h, k] <- 0
            A[c(k, h), (k + 1):m] <- G %*% A[c(k, h), (k + 1):m]
        }
    }
    return(list(Q = t(Q), R = triu(A)))
}
```

Numerical verification:

```
> A <- matrix(as.integer(runif(n = 16,min=1,max=100)),nrow=4)
> A
     [,1] [,2] [,3] [,4]
[1,]   51   26   64   50
[2,]   18   99   31   68
[3,]   76   80   62   49
[4,]   20   55   33   25
> classical_gram_schmidt(A)
$Q
          [,1]        [,2]       [,3]       [,4]
[1,] 0.5345959 -0.34800600  0.71283230  0.29149472
[2,] 0.1886809  0.86477187  0.09432913  0.45570949
```

```
[3,] 0.7966527 -0.06500513 -0.59513022 -0.08329963
[4,] 0.2096454  0.35614010  0.35887624 -0.83691150

$R
         [,1]      [,2]     [,3]     [,4]
[1,] 95.39916 107.84162 96.37401 83.83721
[2,]  0.00000  90.95155 12.25785 47.12244
[3,]  0.00000   0.00000 23.49031 21.86652
[4,]  0.00000   0.00000  0.00000 20.55851

> modified_gram_schmidt(A)
$Q
          [,1]        [,2]        [,3]        [,4]
[1,] 0.5345959 -0.34800600  0.71283230  0.29149472
[2,] 0.1886809  0.86477187  0.09432913  0.45570949
[3,] 0.7966527 -0.06500513 -0.59513022 -0.08329963
[4,] 0.2096454  0.35614010  0.35887624 -0.83691150

$R
         [,1]      [,2]     [,3]     [,4]
[1,] 95.39916 107.84162 96.37401 83.83721
[2,]  0.00000  90.95155 12.25785 47.12244
[3,]  0.00000   0.00000 23.49031 21.86652
[4,]  0.00000   0.00000  0.00000 20.55851

> householder(A)
$Q
           [,1]        [,2]        [,3]        [,4]
[1,] -0.5345959  0.34800600  0.71283230 -0.29149472
[2,] -0.1886809 -0.86477187  0.09432913 -0.45570949
[3,] -0.7966527  0.06500513 -0.59513022  0.08329963
[4,] -0.2096454 -0.35614010  0.35887624  0.83691150

$R
              [,1]          [,2]          [,3]      [,4]
[1,] -9.539916e+01 -1.078416e+02 -9.637401e+01 -83.83721
[2,]  6.618053e-16 -9.095155e+01 -1.225785e+01 -47.12244
[3,]  2.998161e-15 -1.419169e-15  2.349031e+01  21.86652
[4,] -5.895097e-16 -6.976408e-15  1.776357e-15 -20.55851

> givens(A)
$Q
          [,1]        [,2]        [,3]        [,4]
[1,] 0.5345959 -0.34800600  0.71283230  0.29149472
[2,] 0.1886809  0.86477187  0.09432913  0.45570949
[3,] 0.7966527 -0.06500513 -0.59513022 -0.08329963
[4,] 0.2096454  0.35614010  0.35887624 -0.83691150

$R
```

```
         [,1]      [,2]     [,3]     [,4]
[1,] 95.39916 107.84162 96.37401 83.83721
[2,]  0.00000  90.95155 12.25785 47.12244
[3,]  0.00000   0.00000 23.49031 21.86652
[4,]  0.00000   0.00000  0.00000 20.55851
```

25×15 **Vandermonde：**

Since the Givens decomposition is to embed the rotation matrix into the unit matrix and multiply it to get Q, the matrix cannot be multiplied when the number of rows and columns are not equal.So Givens only deals with square matrices.

Therefore, the Givens method is not considered below.

```
> v0=as.vector(rand(25,1))
> vand=vandermonde.matrix(v0, 15)
> kappa(vand)
[1] 2.997201e+12
> kappa(t(vand)%*%vand)
[1] 2.627191e+17
>
> norm(vand-classical_gram_schmidt(vand)$Q%*%classical_gram_schmidt(vand)$R,'2')
[1] 3.5695e-16
> norm(vand-modified_gram_schmidt(vand)$Q%*%modified_gram_schmidt(vand)$R,'2')
[1] 3.5695e-16
> norm(vand-householder(vand)$Q%*%householder(vand)$R,'2')
[1] 2.440901e-15
> norm(diag(15)-
t(classical_gram_schmidt(vand)$Q)%*%classical_gram_schmidt(vand)$Q,'2')
[1] 2.198188e-05
> norm(diag(15)-
t(modified_gram_schmidt(vand)$Q)%*%modified_gram_schmidt(vand)$Q,'2')
[1] 2.198188e-05
> norm(diag(25)-t(householder(vand)$Q)%*%householder(vand)$Q,'2')
[1] 1.830638e-15
```

It can be seen that the householder error is the smallest and the effect is the best from the above results.

3、Write R functions for Gaussian elimination with and without partial pivoting. Solve a set of linear equations with your functions to show how pivoting is better.

Code：

```
lu_normal <- function(A){
  n = ncol(A)
  m = nrow(A)
  U = A
  L = diag(m)
```

```
  for (k in 1:(m-1)) {
    for (j in (k+1):m) {
      L[j,k] = U[j,k]/U[k,k]
      U[j,k:m] = U[j,k:m] - L[j,k]*U[k,k:m]
    }
  }
  return(list(L = L, U = U))
}
lu_partial_pivoting = function(A){
  n = ncol(A)
  m = nrow(A)
  U = A
  L = diag(m)
  P = diag(m)
  for (k in 1:(m-1)) {
    tra = U[k:m, k]
    maxlabel = which.max(abs(tra))+k-1
    maxtmp = U[maxlabel, k:m]
    U[maxlabel, k:m] = U[k, k:m]
    U[k, k:m] = maxtmp
    if (k > 1) {
      tra = L[maxlabel, 1:(k-1)]
      L[maxlabel, 1:(k-1)] = L[k, 1:(k-1)]
      L[k, 1:(k-1)] = tra
    }
    tra = P[maxlabel,]
    P[maxlabel,] = P[k,]
    P[k,] = tra
    for (j in (k+1):m) {
      L[j,k] = U[j,k]/U[k,k]
      U[j,k:m] = U[j,k:m] - L[j,k]*U[k,k:m]
    }
  }
  return(list(L = L, U = U, P = P))
}
```

Test:

The problem of finding a system of linear equations is easily solved by the chasing method after LU decomposition, multiplied by L^{-1} on both sides.

```
> set.seed(1234)
> A = rand(4,4)
> A
          [,1]        [,2]      [,3]      [,4]
[1,] 0.1137034 0.860915384 0.6660838 0.2827336
[2,] 0.6222994 0.640310605 0.5142511 0.9234335
[3,] 0.6092747 0.009495756 0.6935913 0.2923158
[4,] 0.6233794 0.232550506 0.5449748 0.8372956
```

```
> B = A
> B[1,1] = 0
> lu__partial_pivoting(B)
$L
          [,1]       [,2]       [,3] [,4]
[1,] 1.0000000  0.0000000  0.0000000    0
[2,] 0.0000000  1.0000000  0.0000000    0
[3,] 0.9982674  0.4741035  1.0000000    0
[4,] 0.9773738 -0.2529784 -0.9533523    1

$U
          [,1]      [,2]       [,3]        [,4]
[1,] 0.6233794 0.2325505  0.5449748  0.83729563
[2,] 0.0000000 0.8609154  0.6660838  0.28273358
[3,] 0.0000000 0.0000000 -0.3455721 -0.04645647
[4,] 0.0000000 0.0000000  0.0000000 -0.49879885

$P
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    1
[2,]    1    0    0    0
[3,]    0    1    0    0
[4,]    0    0    1    0

> lu_normal(B)
$L
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]  Inf    1    0    0
[3,]  Inf  NaN    1    0
[4,]  Inf  NaN  NaN    1

$U
     [,1]      [,2]      [,3]      [,4]
[1,]    0 0.8609154 0.6660838 0.2827336
[2,]  NaN      -Inf      -Inf      -Inf
[3,]  NaN       NaN       NaN       NaN
[4,]  NaN       NaN       NaN       NaN
```

It can be seen from the above that the lu__partial_pivoting is better.We can also verify this with the method in question 2.

4、Compare solving equations by QR factorization and by Gaussian elimination first theoretically then use a numerical example.

QR：

$$Ax = b \Leftrightarrow QRx = b$$
$$Rx = Q^{-1}b = Q^{T}b$$

It is worth mentioning that here A is a square matrix, so the time complexity of using the householder algorithm is$\Theta(\frac{4}{3}m^3)$

Matrix multiplication cost$\Theta(m^3)$

Use the chase method to solve for x cost$O(m^2)$。Consider only three items, the total is$\Theta(\frac{7}{3}m^3)$

Gauss：

$$Ax = b \Leftrightarrow LUx = b$$
$$Ux = L^{-1}b$$

Decompose$\sum_{k=1}^{m-1}\sum_{j=k+1}^{m} 2(m-k+1) = \sum_{k=1}^{m-1} 2(m-k+1)(m-k) \sim \frac{2}{3}m^3$

Matrix multiplication cost$\Theta(\frac{1}{2}m^3)$

Upper triangular matrix inversion cost$\Theta(\frac{1}{2}m^2)$

The algorithm time complexity is$\Theta(\frac{7}{6}m^3)$

Use the chase method to solve for x cost$O(m^2)$。Consider only three items, the total is$\Theta(\frac{7}{6}m^3)$

It can be seen from the above that the former time is about twice the latter.

Code:

```
solve_Gauss = function(A,b){
  LU = lu_partial_pivoting(A)
  P = LU$P
  U = LU$U
  L = LU$L
  b = P%*%b
  m = nrow(A)
  y = matrix(0,m,1)
  y[1] = b[1]/L[1,1]
  for (i in 2:m) {
    B = b[i]
    for (j in 1:(i-1)) {
      B = B - y[j]*L[i,j]
    }
    y[i] = B/L[i,i]
  }
  x = matrix(0,m,1)
  x[m] = y[m]/U[m,m]
  for (i in (m-1):1){
    Y = y[i]
    for (j in m:(i+1)){
      Y = Y- U[i,j] * x[j]
    }
    x[i] = Y / U[i,i]
```

```
  }
  return(x = x)
}

solve_QR = function(A,b){
  m = nrow(A)
  House = Householder(A)
  Q = House$Q
  R = House$R
  y = t(Q) %*% b
  x = matrix(0,m,1)
  x[m] = y[m]/R[m,m]
  for (i in (m-1):1){
    Y = y[i]
    for (j in m:(i+1)){
      Y = Y - R[i,j] * x[j]
    }
    x[i] = Y / R[i,i]
  }
  return(x = x)
}
```

Test:

```
> t=rand(40,40)
> b = rand(40,1)
> system.time(solve_Gauss(t,b))
 用户   系统   流逝
0.006 0.001 0.006
> system.time(solve_QR(t,b))
 用户   系统   流逝
0.009 0.000 0.010
```

The result is as expected!