The Power Method, QR Method, and Deflation

By Eli Slothower, CMU SCS class of 2025

The purpose of this document is to showcase the Power Method, the QR Method, and improvements upon these methods to calculate eigenvalues, eigenvectors, singular values, and singular vectors of a matrix. All theory and reasoning behind the following code can be found in the paper that accompanies this code.

In [193…

```
#Load in LinearAlgebra package
using LinearAlgebra
```

The Power Method

Below is our PowerMethod() function for a square, symmetric matrix A, which returns a tuple of, in this order, the dominant eigenvalue, the dominant eigenvector, and the dominant singular value. We use the below helper functions to extract these values from this tuple individually, as well as to compute the dominant left singular vector and the dominant right singular vector.

Note that if the inputted matrix does not meet the above preconditions, PowerMethod() will not run, which it would notify you of.

In [193…

```
function PowerMethod(A, x) #takes in a matrix A that meets the preconditions,
                           #and an initial approximation x
    m, n = size(A) #gets dimensions of A
    if (m == n) #checks that inputted matrix is square
        for i in 1:200
            x = ((A*x)/sqrt(dot((A*x), (A*x)))) #creates the next value
                                                 #of x (and divides by its
                                                 #magnitude for overflow
                                                 #purposes)
        end
        nextX = A*x #calculates one next x for eigenvalue computation
        lambda1 = dot(x, nextX)/dot(x, x) #calculates lambda_1
        eigenvector = x #finds corresponding eigenvector
        singularValue = sqrt(abs(lambda1)) #calculates dominant singular
                                           #value by taking square root
                                           #of lambda_1
        return(lambda1, eigenvector, singularValue) #returns tuple as
                                                    #outlined above
    else
        print("A does not meet the preconditions") #tells client if
                                                   #preconditions are
                                                   #not met
        return
    end

end
```

Out[193… PowerMethod (generic function with 1 method)

In [193…

```
function getEvaluePM(A, x)
    return PowerMethod(A, x)[1]
end
```

Out[193… getEvaluePM (generic function with 1 method)

In [193…

```
function getEvectorPM(A, x)
    return PowerMethod(A, x)[2]
end
```

Out[193… getEvectorPM (generic function with 1 method)

In [193…

```
function getSingularValuePM(A, x)
    return PowerMethod(A, x)[3]
end
```

Out[193… getSingularValuePM (generic function with 1 method)

In [193…
```
function getSingularVectorAtAPM(A, x)
    PMresult = PowerMethod(transpose(A)*A, x) #calculates PowerMethod()
                                               #on AtA
    eigenvector = PMresult[2]
    return eigenvector
end
```

Out[193… getSingularVectorAtAPM (generic function with 1 method)

In [193…
```
function getSingularVectorAAtPM(A, x)
    PMresult = PowerMethod(A*transpose(A), x) #calculates PowerMethod()
                                               #on AAt
    eigenvector = PMresult[2]
    return eigenvector
end
```

Out[193… getSingularVectorAAtPM (generic function with 1 method)

Example PM.1

In [193…
```
A = [2 0; 0 1]
x = [1;1]
result = getEvaluePM(A,x)
```

Out[193… 2.0

In [194…
```
A = [2 0; 0 1]
x = [1;1]
result = getEvectorPM(A,x)
```

Out[194…
```
2-element Vector{Float64}:
 1.0
 6.223015277861142e-61
```

In [194…
```
A = [2 0; 0 1]
x = [1;1]
result = getSingularValuePM(A,x)
```

Out[194… 1.4142135623730951

In [194…
```
A = [2 0; 0 1]
x = [1;1]
result = getSingularVectorAtAPM(A,x)
```

Out[194…
```
2-element Vector{Float64}:
 1.0
 3.8725919148493183e-121
```

In [194…
```
A = [2 0; 0 1]
x = [1;1]
result = getSingularVectorAAtPM(A,x)
```

Out[194…
```
2-element Vector{Float64}:
 1.0
 3.8725919148493183e-121
```

Example PM.2

In [194…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getEvaluePM(A,x)
```

Out[194…
```
17.034137096355504
```

In [194…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getEvectorPM(A,x)
```

Out[194…
```
3-element Vector{Float64}:
 0.5774416595564781
 0.5470106434803808
 0.6060862032152194
```

In [194…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getSingularValuePM(A,x)
```

Out[194…
```
4.127243280490684
```

In [194…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getSingularVectorAtAPM(A,x)
```

Out[194…
```
3-element Vector{Float64}:
 0.5774416595564782
 0.5470106434803808
 0.6060862032152194
```

In [194…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getSingularVectorAAtPM(A,x)
```

Out[194…
```
3-element Vector{Float64}:
 0.5774416595564782
 0.5470106434803808
 0.6060862032152194
```

Example PM.2.1

In [194…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [999;1200000;1456]
result = getEvaluePM(A,x)
```

Out[194…
```
17.034137096355504
```

In [195…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [999;1200000;1456]
result = getEvectorPM(A,x)
```

Out[195…
```
3-element Vector{Float64}:
 0.5774416595564781
 0.5470106434803808
 0.6060862032152194
```

In [195…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [999;1200000;1456]
result = getSingularValuePM(A,x)
```

Out[195…
```
4.127243280490684
```

In [195…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [999;1200000;1456]
result = getSingularVectorAtAPM(A,x)
```

Out[195…
```
3-element Vector{Float64}:
 0.5774416595564782
 0.5470106434803808
 0.6060862032152194
```

In [195…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [999;1200000;1456]
result = getSingularVectorAAtPM(A,x)
```

Out[195...
```
3-element Vector{Float64}:
 0.5774416595564782
 0.5470106434803808
 0.6060862032152194
```

Example PM.3

In [195...
```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getEvaluePM(A,x)
```

Out[195...
```
105077.92749234011
```

In [195...
```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getEvectorPM(A, x)
```

Out[195...
```
7-element Vector{Float64}:
 0.7045593165908005
 0.003167479342962275
 0.05261982026017225
 0.11212454652807066
 0.6698905053676605
 0.0017702795012152963
 0.19871827794675967
```

In [195...
```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getSingularValuePM(A, x)
```

Out[195...
```
324.1572573494848
```

In [195…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getSingularVectorAtAPM(A, x)
```

Out[195…

```
7-element Vector{Float64}:
 0.7061387957133922
 0.003167436349363254
 0.05253617071329216
 0.1119992424967313
 0.6683860178933628
 0.0017710402165370632
 0.19827022843472986
```

In [195…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getSingularVectorAAtPM(A, x)
```

Out[195…

```
7-element Vector{Float64}:
 0.7061387957133922
 0.003167436349363254
 0.05253617071329216
 0.1119992424967313
 0.6683860178933628
 0.0017710402165370632
 0.19827022843472986
```

The QR Method

Below is our QRMethod() function for a square matrix A where det(A) != 0, which returns a tuple of, in this order, a list of the eigenvalues, a matrix of the eigenvectors, and a list of the singular values. We use the below helper functions to extract these values from this tuple individually, as well as to compute the left singular vectors and the right singular vectors.

In [195...

```
function QRMethod(A)
    m, n = size(A) #gets dimensions of A
    if m == n && det(A) != 0 #checks that A is square and det(A) != 0
        Q, R = qr(A) #finds QR decomposition of A
        eigenvectors = Q
        for i in 1:50
            newA = R*Q #iteratively finds newA by swapping Q and R
            Q, R = qr(newA) #finds QR decomposition of newA, following
                            #the QR method
            eigenvectors = eigenvectors*Q #calculates matrix of A's
                                          #eigenvectors
        end
        eigenvalues = diag(R*Q,0) #takes the components on the diagonal
                                  #of RQ, which are the eigenvalues of A

        singularValues = []
        for i in eachindex(eigenvalues)
            append!(singularValues, sqrt(abs(eigenvalues[i])))
            #calculates vector of singular values based off of square
            #roots of eigenvalues of A
        end

        return (eigenvalues, eigenvectors*Q, singularValues)
    else
        print("A does not meet the preconditions") #tells client if
                                                   #preconditions are
                                                   #not met
        return
    end
end
```

Out[195... QRMethod (generic function with 1 method)

In [196...

```
function getEvaluesQR(A)
    return QRMethod(A)[1]
end
```

Out[196... getEvaluesQR (generic function with 1 method)

In [196...

```
function getEvectorsQR(A)
    return QRMethod(A)[2]
end
```

Out[196... getEvectorsQR (generic function with 1 method)

In [196…
```
function getSingularValuesQR(A)
    return QRMethod(A)[3]
end
```

Out[196…
```
getSingularValuesQR (generic function with 1 method)
```

In [196…
```
function getLeftSingularVectorsQR(A)
    QRAtA = QRMethod(transpose(A)*A) #calculates QRMethod() on AtA
    eigenvectorMatrix = QRAtA[2]
    return eigenvectorMatrix
end
```

Out[196…
```
getLeftSingularVectorsQR (generic function with 1 method)
```

In [196…
```
function getRightSingularVectorsQR(A)
    QRAAt = QRMethod(A*transpose(A)) #calculates QRMethod() on AAt
    eigenvectorMatrix = QRAAt[2]
    return eigenvectorMatrix
end
```

Out[196…
```
getRightSingularVectorsQR (generic function with 1 method)
```

Example QR.1

In [196…
```
A = [2 0; 0 1]
result = getEvaluesQR(A)
```

Out[196…
```
2-element Vector{Float64}:
 2.0
 1.0
```

In [196…
```
A = [2 0; 0 1]
result = getEvectorsQR(A)
```

Out[196…
```
2×2 Matrix{Float64}:
 1.0  0.0
 0.0  1.0
```

In [196…
```
A = [2 0; 0 1]
result = getSingularValuesQR(A)
```

Out[196…
```
2-element Vector{Any}:
 1.4142135623730951
 1.0
```

In [196…

```
A = [2 0; 0 1]
result = getLeftSingularVectorsQR(A)
```

Out[196…

```
2×2 Matrix{Float64}:
 1.0  0.0
 0.0  1.0
```

In [196…

```
A = [2 0; 0 1]
result = getRightSingularVectorsQR(A)
```

Out[196…

```
2×2 Matrix{Float64}:
 1.0  0.0
 0.0  1.0
```

Example QR.2

In [197…

```
A = [6 5 6; 5 4 7; 6 7 5]
result = getEvaluesQR(A)
```

Out[197…

```
3-element Vector{Float64}:
 17.034137096355504
 -2.561302422819227
  0.5271653264637243
```

In [197…

```
A = [6 5 6; 5 4 7; 6 7 5]
result = getEvectorsQR(A)
```

Out[197…

```
3×3 Matrix{Float64}:
 0.577442   0.10056   -0.810215
 0.547011   0.689054   0.475377
 0.606086  -0.717699   0.342882
```

In [197…

```
A = [6 5 6; 5 4 7; 6 7 5]
result = getSingularValuesQR(A)
```

Out[197…

```
3-element Vector{Any}:
 4.127243280490684
 1.6004069553770464
 0.7260615169968205
```

In [197…

```
A = [6 5 6; 5 4 7; 6 7 5]
result = getLeftSingularVectorsQR(A)
```

Out[197…

```
3×3 Matrix{Float64}:
 0.577442   0.10056   -0.810215
 0.547011   0.689054   0.475377
 0.606086  -0.717699   0.342882
```

In [197…

```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getRightSingularVectorsQR(A)
```

Out[197…

```
3×3 Matrix{Float64}:
 0.577442   0.10056   -0.810215
 0.547011   0.689054   0.475377
 0.606086  -0.717699   0.342882
```

Example QR.3

In [197…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
result = getEvaluesQR(A)
```

Out[197…

```
7-element Vector{Float64}:
  83154.66898108396
 -80972.65976933317
   5664.178317222833
   4090.3044401729617
  -1897.9260490013883
   1067.7078726978568
    131.7262071566637
```

In [197…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
result = getEvectorsQR(A)
```

Out[197…

```
7×7 Matrix{Float64}:
 -0.211808     -0.976443    …   0.0317327  -0.0170728   0.0180072
  0.00173457   -0.00265036      0.127985   -0.504825   -0.818175
  0.0600075    -0.0238357      -0.0265957   0.015296   -0.0650124
  0.107956     -0.0636803      -0.799551    0.435794   -0.388956
  0.929261     -0.19642         0.25778     0.158064   -0.0533551
  0.000674297  -0.00167234  …  -0.0184679   0.1339      0.198574
  0.276318     -0.0578394      -0.525197   -0.715399    0.36397
```

In [197…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
result = getSingularValuesQR(A)
```

Out[197…

```
7-element Vector{Any}:
 288.3655128150451
 284.5569534721181
  75.26073556126616
  63.95548795977529
  43.565193090371906
  32.67579949592445
  11.477203803917734
```

In [197…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
result = getLeftSingularVectorsQR(A)
```

Out[197…

```
7×7 Matrix{Float64}:
 0.780873    -0.623331      0.00760364  …   0.0317327  -0.0170728   0.0180072
 0.00314495   0.000377441  -0.130939        0.127985   -0.504825   -0.818175
 0.0480011    0.043185     -0.733339       -0.0265957   0.015296   -0.0650124
 0.104992     0.0684566     0.0421271      -0.799551    0.435794   -0.388956
 0.588594     0.745429      0.0483079       0.25778     0.158064   -0.0533551
 0.00179785  -0.000138343  -0.663907    …  -0.0184679   0.1339      0.198574
 0.174512     0.221907     -0.0113904      -0.525197   -0.715399    0.36397
```

In [197…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getRightSingularVectorsQR(A)
```

Out[197…

```
7×7 Matrix{Float64}:
 0.780873    -0.623331      0.00760364  …   0.0317327  -0.0170728   0.0180072
 0.00314495   0.000377441  -0.130939        0.127985   -0.504825   -0.818175
 0.0480011    0.043185     -0.733339       -0.0265957   0.015296   -0.0650124
 0.104992     0.0684566     0.0421271      -0.799551    0.435794   -0.388956
 0.588594     0.745429      0.0483079       0.25778     0.158064   -0.0533551
 0.00179785  -0.000138343  -0.663907    …  -0.0184679   0.1339      0.198574
 0.174512     0.221907     -0.0113904      -0.525197   -0.715399    0.36397
```

Power Method Using Deflation

Below is our PowerMethodDeflation() function for a square, symmetric matrix A, which returns a tuple of, in this order, a vector of all eigenvalues, a 2D vector (a 2D list) of all eigenvectors, and a list of the singular values. We use the below helper functions to extract these values from this tuple individually, as well as to compute the left singular vectors and the right singular vectors.

In [198…

```
function PowerMethodDeflation(A, x) #takes in same input
                                    #as PowerMethod()
    eigenvector = x #dummy vector to initialize variable
    allEvalues = [] #dummy vector to initialize variable
    allEvectors = [] #dummy vector to initialize variable
    m,n = size(A) #gets dimensions of A
    for i in 1:n
        PMResults = PowerMethod(A, x) #runs power method on A
        eigenvalue = PMResults[1] #gets dominant eigenvalue of A
        eigenvector = PMResults[2] #gets dominant eigenvalue of A
        append!(allEvalues, eigenvalue) #appends current dominant
                                        #eigenvalue to list of
                                        #all eigenvalues of A
        push!(allEvectors, eigenvector) #appends current dominant
                                          #eigenvector to list of all
                                          #eigenvectors of A
        A = (A - ((eigenvalue/((dot(eigenvector, eigenvector))^2))
            * eigenvector*transpose(eigenvector)))
        #calculates new A based off of deflation method by getting rid
        #of the current dominant eigenvalue and eigenvector, so when
        #the power method is called on this new A, the next dominant
        #eigenvalue and eigenvector will be found
    end

    singularValues = []
    for i in eachindex(allEvalues)
        append!(singularValues, sqrt(abs(allEvalues[i])))
        #calculates vector of singular values based off of square
        #roots of eigenvalues of A
    end

    return (allEvalues, allEvectors, singularValues)
    #returns tuple as outlined above

    #Note: an inputted matrix A that does not meet the preconditions will
    #halt the program through the call to PowerMethod(), which already
    #checks these preconditions. Because of this, we did not implement
    #checking these preconditions here again, because that would be
    #redundant
end
```

Out[198… PowerMethodDeflation (generic function with 2 methods)

In [198…

```
function getEvaluesPMD(A, x)
    return PowerMethodDeflation(A, x)[1]
end
```

Out[198… getEvaluesPMD (generic function with 2 methods)

In [198…
```
function getEvectorsPMD(A, x)
    return PowerMethodDeflation(A, x)[2]
end
```

Out[198… getEvectorsPMD (generic function with 2 methods)

In [198…
```
function getSingularValuesPMD(A, x)
    return PowerMethodDeflation(A, x)[3]
end
```

Out[198… getSingularValuesPMD (generic function with 2 methods)

In [198…
```
function getSingularVectorsAtAPMD(A, x)
    PMDresult = PowerMethodDeflation(transpose(A)*A, x)
    #calculates PowerMethod() on AtA

    eigenvector = PMDresult[2]
    return eigenvector
end
```

Out[198… getSingularVectorsAtAPMD (generic function with 2 methods)

In [198…
```
function getSingularVectorsAAtPMD(A, x)
    PMDresult = PowerMethodDeflation(A*transpose(A), x)
    #calculates PowerMethod() on AAt

    eigenvector = PMDresult[2]
    return eigenvector
end
```

Out[198… getSingularVectorsAAtPMD (generic function with 2 methods)

Example PMD.1

In [198…
```
A = [2 0; 0 1]
x = [1;1]
result = getEvaluesPMD(A,x)
```

Out[198…
```
2-element Vector{Any}:
 2.0
 1.0
```

In [198…
```
A = [2 0; 0 1]
x = [1;1]
result = getEvectorsPMD(A,x)
```

Out[198…
```
2-element Vector{Any}:
 [1.0, 6.223015277861142e-61]
 [-1.2446030555722283e-60, 1.0]
```

In [198…
```
A = [2 0; 0 1]
x = [1;1]
result = getSingularValuesPMD(A,x)
```

Out[198…
```
2-element Vector{Any}:
 1.4142135623730951
 1.0
```

In [198…
```
A = [2 0; 0 1]
x = [1;1]
result = getSingularVectorsAtAPMD(A,x)
```

Out[198…
```
2-element Vector{Any}:
 [1.0, 3.8725919148493183e-121]
 [-1.5490367659397273e-120, 1.0]
```

In [199…
```
A = [2 0; 0 1]
x = [1;1]
result = getSingularVectorsAAtPMD(A,x)
```

Out[199…
```
2-element Vector{Any}:
 [1.0, 3.8725919148493183e-121]
 [-1.5490367659397273e-120, 1.0]
```

Example PMD.2

In [199…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getEvaluesPMD(A,x)
```

Out[199…
```
3-element Vector{Any}:
 17.034137096355504
 -2.5613024228192276
  0.5271653264637228
```

In [199…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getEvectorsPMD(A,x)
```

Out[199…
```
3-element Vector{Any}:
 [0.5774416595564781, 0.5470106434803808, 0.6060862032152194]
 [0.10055966074712493, 0.6890544052397919, -0.7176989488985335]
 [-0.8102153321426852, 0.47537709509274567, 0.3428815145529151]
```

In [199…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getSingularValuesPMD(A,x)
```

Out[199…
```
3-element Vector{Any}:
 4.127243280490684
 1.6004069553770464
 0.7260615169968195
```

In [199…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getSingularVectorsAtAPMD(A,x)
```

Out[199…
```
3-element Vector{Any}:
 [0.5774416595564782, 0.5470106434803808, 0.6060862032152194]
 [0.10055966074712498, 0.6890544052397903, -0.717698948898535]
 [-0.8102153321425966, 0.4753770950928481, 0.3428815145529826]
```

In [199…
```
A = [6 5 6; 5 4 7; 6 7 5]
x = [1;1;1]
result = getSingularVectorsAAtPMD(A,x)
```

Out[199…
```
3-element Vector{Any}:
 [0.5774416595564782, 0.5470106434803808, 0.6060862032152194]
 [0.10055966074712498, 0.6890544052397903, -0.717698948898535]
 [-0.8102153321425966, 0.4753770950928481, 0.3428815145529826]
```

Example PMD.3

In [199…
```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getEvaluesPMD(A,x)
```

Out[199…

```
7-element Vector{Any}:
  105077.92749234011
 -102898.07681220338
    5664.178317222874
    4090.304440172935
   -1897.9260490013871
    1067.70787269786
     131.7262071566645
```

In [199…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getEvectorsPMD(A, x)
```

Out[199…

```
7-element Vector{Any}:
 [0.7045593165908005, 0.003167479342962275, 0.05261982026017225, 0.11212454652807066, 0.6698905053676605, 0.0017702795012152963, 0.19871827794675967]
 [-0.7052142342198308, 3.0234863635354026e-5, 0.03765983138129654, 0.05652844207719054, 0.6763768965920753, -0.0003346912991381668, 0.20142824284209992]
 [-0.007603642096408602, 0.13093863677172232, 0.7333394413234289, -0.04212712548499737, -0.048307921014386974, 0.6639069440513676, 0.011390414070041556]
 [0.004030465259957338, 0.2054437392950462, -0.6729608506380436, 0.04505471535825179, 0.03573612581646158, 0.7081823167680581, 0.008433682198244108]
 [-0.03173272150838021, -0.12798483516662526, 0.026595685049208624, 0.7995510868660561, -0.25778049897218774, 0.018467873486066943, 0.5251969129570998]
 [0.017072842650132758, 0.504824853693538, -0.015296005391544923, -0.4357944730953685, -0.15806402191177316, -0.13390024837948816, 0.7153992472995199]
 [-0.018007233034550697, 0.8181745897256592, 0.06501242500090056, 0.3889562511207561, 0.05335512556226336, -0.19857438300769817, -0.3639697028388841]
```

In [199…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getSingularValuesPMD(A, x)
```

Out[199…

```
7-element Vector{Any}:
 324.1572573494848
 320.7773009615914
  75.26073556126643
  63.95548795977508
  43.56519309037189
  32.6757994959245
  11.477203803917767
```

In [199…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getSingularVectorsAtAPMD(A, x)
```

Out[199…

```
7-element Vector{Any}:
 [0.7061387957133922, 0.0031674363493632545, 0.05253617071329216, 0.1119992424967313, 0.6683860178933628, 0.0017710402165370632, 0.19827022843472986]
 [-0.7068753014426122, 2.2792831075611973e-5, 0.037536497750205164, 0.05626544776689501, 0.6748083409699729, -0.00033885340616939214, 0.20096294820405255]
 [-0.007603642096409797, 0.13093863677172238, 0.7333394413234289, -0.04212712548499736, -0.04830792101438463, 0.6639069440513679, 0.011390414070042045]
 [0.004030465259959211, 0.20544373929504609, -0.6729608506380438, 0.04505471535825195, 0.035736125816458665, 0.708182316768058, 0.008433682198243576]
 [-0.03173272150838026, -0.12798483516662446, 0.026595685049221617, 0.7995510868660768, -0.25778049897201083, 0.018467873486067054, 0.5251969129571548]
 [0.01707284264996484, 0.5048248536935429, -0.015296005391517193, -0.4357944730953335, -0.15806402191136645, -0.1339002483794893, 0.7153992472996321]
 [-0.018007233028058577, 0.8181745897256248, 0.06501242500013753, 0.3889562511199688, 0.05335512555171472, -0.19857438300773328, -0.3639697028417869]
```

In [200…

```
A = [987 234 6587 12445 98661 89 29374;
     234 600 1 45 73 999 555;
     6587 1 5043 72 800 819 301;
     12445 45 72 20 4444 19 20;
     98661 73 800 4444 0 100 101;
     89 999 819 19 100 4572 0;
     29374 555 301 20 101 0 16;]
x = [1;1;1;1;1;1;1]
result = getSingularVectorsAAtPMD(A, x)
```

Out[200…

```
7-element Vector{Any}:
 [0.7061387957133922, 0.0031674363493632545, 0.05253617071329216, 0.1119992424967313, 0.6683860178933628, 0.0017710402165370632, 0.19827022843472986]
 [-0.7068753014426122, 2.2792831075611973e-5, 0.037536497750205164, 0.05626544776689501, 0.6748083409699729, -0.00033885340616939214, 0.20096294820405255]
 [-0.007603642096409797, 0.13093863677172238, 0.7333394413234289, -0.04212712548499736, -0.04830792101438463, 0.6639069440513679, 0.011390414070042045]
 [0.004030465259959211, 0.20544373929504609, -0.6729608506380438, 0.04505471535825195, 0.035736125816458665, 0.708182316768058, 0.008433682198243576]
 [-0.03173272150838026, -0.12798483516662446, 0.026595685049221617, 0.7995510868660768, -0.25778049897201083, 0.018467873486067054, 0.5251969129571548]
 [0.01707284264996484, 0.5048248536935429, -0.015296005391517193, -0.4357944730953335, -0.15806402191136645, -0.1339002483794893, 0.7153992472996321]
 [-0.018007233028058577, 0.8181745897256248, 0.06501242500013753, 0.3889562511199688, 0.05335512555171472, -0.19857438300773328, -0.3639697028417869]
```