**VLSI DPS HW#1**

**電機四 4109064119 陳柏淳**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | % Problem 1. Last square optimization problem  A = [ 15 -13 20 -8;  -5 -15 -4 -4;  -17 16 -2 9;  10 -19 -14 -15;  -7 8 -7 15;  14 10 -8 -17;  -5 -3 16 -2;  13 -5 -10 -19];  b = [13; 10; -15; 9; 3; 18; 3; 20]; |

1. Last square optimization problem

a) Pseudo inverse

* Code

|  |  |
| --- | --- |
| 1  2 | % (a) Pseudo inverse  x\_pseudo = pinv(A) \* b; |

* Result

**x1 = 0.4638**

**x2 = -0.1005**

**x3 = -0.0716**

**x4 = -0.4137**

(b) QR decomposition

* Code

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | % (b) QR decomposition  [Q, R] = qr(A);  y = Q' \* b;  R1 = R(1:4, 1:4);  y1 = y(1:4);  x\_QR = inv(R1) \* y1; % R \* x\_b = Q' \* b |

* Result

**x1 = 0.4638**

**x2 = -0.1005**

**x3 = -0.0716**

**x4 = -0.4137**

(c) Compare if a) and b) yield the same result?

**Yes, both a) and b) yield the same result.**

2. Eigen decomposition

* Code

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  2425  26272829303132333435363738394041424344454647484950515253545556575859606162636465666768697071727374757677787980818283848586878889909192 | % Problem 2. Eigen decomposition  M = [ --2 16 -6 -16 3 15 -6 -19;  16 -17 10 -2 7 8 3 5;  --6 10 15 -1 -15 -18 9 -8;  --16 -2 -1 9 0 0 0 18;  3 7 -15 0 14 19 -12 11;  15 8 -18 0 19 10 -8 -17;  -6 3 9 0 -12 -8 15 20;  -19 5 -8 18 11 -17 20 20];    % Use iterative  convergence\_threshold = 1e-4;  [D\_iter, V\_iter, sweeps] = eig\_iterative(M, convergence\_threshold);  % Use eig()  [V, D] = eig(M);    disp('Eigenvalue matrix D Using Iterative:');  disp(D\_iter);  disp('Eigenvalue matrix V Using Iterative:');  disp(V\_iter);  disp('The numbers of sweeps:');  disp(sweeps);  disp('Eigenvalue matrix D Using eig():');  disp(D);  disp('Eigenvalue matrix V Using eig():');  disp(V);    function [D\_iter, V\_iter, sweeps] = eig\_iterative(M, Convergence\_Threshold)  M\_tiled = M;  convergence = false;  sweeps = 0;    % Find D  while ~convergence  [Q, R] = Given\_QR(M\_tiled);  M\_new = R \* Q;  convergence = (det(Q\*M\_new\*Q') - (det(diag(diag(Q\*M\_new\*Q'))))) / (det(M\_tiled)) < Convergence\_Threshold;  % convergence = (det((Q\*M\_new\*Q') - diag(diag(Q\*M\_new\*Q')))) / (det(M\_tiled)) < Convergence\_Threshold;  M\_tiled = M\_new;  sweeps = sweeps + 1;  end    % Find V  D\_iter = M\_tiled;  n = size(D\_iter, 1);  V\_iter = zeros(n);  % Compute eigenvectors for each approximate eigenvalue from D\_iter  for i = 1:n  lambda = D\_iter(i, i);  v = rand(n, 1);  for k = 1:10 % 10 iterations for refinement  v = (M - lambda \* eye(n)) \ v; % inverse iteration  v = v / norm(v); % Normalize  end  V\_iter(:, i) = v;  end  end    function [Q, R] = Given\_QR(M)  [n, m] = size(M);  Q = eye(m);  R = M;  for i = 1 : n-1  for j = i+1 : m  x = R(:, i);  q\_t = Givens(x, i, j);  Q = Q \* q\_t';  R = q\_t \* R;  end  end  end    function R = Givens(x, i, j)  r = sqrt(x(i)^2 + x(j)^2);  cost = x(i) / r;  sint = x(j) / r;    R = eye(length(x));  R(i, i) = cost;  R(i, j) = sint;  R(j, i) = -sint;  R(j, j) = cost;  end |

* Result

**Eigenvector matrix D Using Iterative:**

67.1862 0.0000 -0.0000 0.0000 0.0000 0.0000 -0.0000 -0.0000

0.0000 46.8120 -0.2845 -0.0000 0.0000 -0.0000 -0.0000 0.0000

-0.0000 -0.2845 -36.9128 -0.0000 0.0000 -0.0000 -0.0000 0.0000

-0.0000 -0.0000 -0.0000 -26.0043 0.0000 0.0000 0.0000 -0.0000

-0.0000 0.0000 0.0000 0.0000 16.4684 0.0000 -0.0000 0.0000

-0.0000 0.0000 0.0000 0.0000 0.0000 -11.0351 -0.0000 -0.0000

-0.0000 0.0000 0.0000 0.0000 0.0000 -0.0000 6.0581 0.0000

-0.0000 0.0000 0.0000 0.0000 0.0000 -0.0000 -0.0000 1.4275

**Eigenvector matrix V Using Iterative:**

-0.3839 -0.1511 -0.4687 0.3638 0.3729 0.5802 -0.0097 0.0447

-0.0728 -0.0006 0.7047 -0.1987 0.4103 0.3837 0.3740 0.0562

0.2682 -0.5003 -0.3288 -0.1075 0.1175 -0.2141 0.7059 -0.0092

0.2528 0.3313 -0.0014 0.3618 -0.3726 0.2220 0.3308 0.6308

-0.2731 0.5323 0.0063 0.4116 0.1844 -0.3783 0.3964 -0.3720

-0.4867 0.2184 -0.2156 -0.5016 0.1834 -0.2840 0.0533 0.5479

0.4086 -0.0333 0.0664 0.3131 0.6480 -0.3281 -0.3033 0.3315

0.4827 0.5337 -0.3531 -0.4083 0.2290 0.2993 0.0181 -0.2199

**Sweep times: 38**

**Eigenvector matrix D Using eig():**

-36.9137 0 0 0 0 0 0 0

0 -26.0043 0 0 0 0 0 0

0 0 -11.0351 0 0 0 0 0

0 0 0 1.4275 0 0 0 0

0 0 0 0 6.0581 0 0 0

0 0 0 0 0 16.4684 0 0

0 0 0 0 0 0 46.8129 0

0 0 0 0 0 0 0 67.1862

**Eigenvector matrix V Using eig():**

-0.4687 0.3638 -0.5802 -0.0447 0.0097 -0.3729 -0.1511 -0.3839

0.7047 -0.1987 -0.3837 -0.0562 -0.3740 -0.4103 -0.0006 -0.0728

-0.3288 -0.1075 0.2141 0.0092 -0.7059 -0.1175 -0.5003 0.2682

-0.0014 0.3618 -0.2220 -0.6308 -0.3308 0.3726 0.3313 0.2528

0.0063 0.4116 0.3783 0.3720 -0.3964 -0.1844 0.5323 -0.2731

-0.2156 -0.5016 0.2840 -0.5479 -0.0533 -0.1834 0.2184 -0.4867

0.0664 0.3131 0.3281 -0.3315 0.3033 -0.6480 -0.0333 0.4086

-0.3531 -0.4083 -0.2993 0.2199 -0.0181 -0.2290 0.5337 0.4827

* Verification

利用迭代方法計算eigen value主要取決於迭帶次數與收斂條件，由於題目給的收斂條件過於寬鬆，導致達到收斂條件後後的結果與eig()的結果相去甚遠。因此我將收斂公式從det(t – diag(t))) / det(M)修改為(det(t) – det(diag(t))) / det(M)，在sweeps 38次後就可以滿足收斂條件，達到與eig()相同的結果，而eig()函式在計算過程中會將eigen value由小至大排列，因此顯示矩陣內的元素位置會略有不同。

如果使用原始題目給的收斂條件去迭代的話，僅sweep 3次就會達到收斂條件了，此結果與eig()有很大的落差，使用原始收斂條件(det(t – diag(t))) / det(M)) (code:44行)得到的結果如下。

**Eigenvector matrix D Using Iterative (original converged condition):**

62.5493 20.4652 -1.1407 -2.8255 -0.6693 -0.7815 -0.2282 0.0021

20.4652 -32.0278 -6.1344 -1.5885 1.7020 1.5989 0.2942 -0.0024

-1.1407 -6.1344 46.5752 1.5508 -0.8664 -1.4046 -0.1330 0.0028

-2.8255 -1.5885 1.5508 -24.6506 4.3164 5.3311 0.5658 -0.0050

-0.6693 1.7020 -0.8664 4.3164 7.7065 11.8474 2.1704 -0.0290

-0.7815 1.5989 -1.4046 5.3311 11.8474 -3.8638 -0.1568 0.0256

-0.2282 0.2942 -0.1330 0.5658 2.1704 -0.1568 6.2839 0.0059

0.0021 -0.0024 0.0028 -0.0050 -0.0290 0.0256 0.0059 1.4274

**Eigenvector matrix V Using Iterative (original converged condition):**

0.3839 0.4302 -0.1511 0.3638 -0.0097 0.0377 -0.0097 0.0447

0.0728 -0.6814 -0.0006 -0.1987 0.3740 0.0533 0.3740 0.0562

-0.2682 0.3379 -0.5003 -0.1075 0.7059 -0.0034 0.7059 -0.0092

-0.2528 -0.0346 0.3313 0.3618 0.3308 0.6296 0.3308 0.6308

0.2731 -0.0472 0.5323 0.4116 0.3964 -0.3657 0.3964 -0.3720

0.4867 0.2645 0.2184 -0.5016 0.0533 0.5515 0.0533 0.5479

-0.4086 -0.0972 -0.0333 0.3131 -0.3033 0.3341 -0.3033 0.3315

-0.4827 0.3920 0.5337 -0.4083 0.0181 -0.2234 0.0181 -0.2199

**Sweep times: 3**