Question1

First i calculate rotation's parameters with rotation function.

```
function [c, s] = rotation(A, p, q)
    if abs(A(p, q)) < 1e-6
        c = 1;
        s = 0;
    else
        tao = (A(q, q) - A(p, p)) / (2 * A(p, q));
        if tao >= 0
            t = 1 / (tao + sqrt(1 + tao ^ 2));
        else
            t = 1 / (tao - sqrt(1 + tao ^ 2));
        end
        c = 1 / sqrt(1 + t ^ 2);
        s = t * c;
    end
end
```

then write a function for create jacobi matrix

```
function jacobi = jMatrix(p, q, c, s, n)
    jacobi = eye(n);
    jacobi(p, p) = c;
    jacobi(q, q) = c;
    jacobi(p, q) = s;
    jacobi(q, p) = -s;
end
```

and write a function for calculate off diagonal.

```
function off = off_diagonal(A)
    off = 0;
    [m, n] = size(A);
    for i = 1:1:m
        for j = 1:1:n
            if i ~= j
                off = off + A(i, j) ^ 2;
            end
        end
    end
    off = sqrt(off);
end
```

now it's time for use Jacobi eigenvalue decomposition(cyclic-by-row) algorithm

```
function [jV, jD] = Jacobi_eig_cyclic(A)
    tolerance = 1e-6;
    [m, n] = size(A);
    jV = eye(n);
    jD = A;
    while off_diagonal(jD) > tolerance * fNorm(jD)
```

```
        for p = 1 : n - 1
            for q = p + 1 : n
                [c, s] = rotation(jD, p, q);
                jacobi = jMatrix(p, q, c, s, n);
                jD = jacobi.' * jD * jacobi;
                jV = jV * jacobi;
            end
        end
    end
end
```

we have an iteration with a good condition. First we must find rotation parameters with rotation function, then use a jacobi matrix from right and left to set off diagonal elements to zero. Iteration condition depends on off diagonal of our matrix. we can define a tolerance for compare off diagonal to our matrix norm.

Question2

```
function [U, S, V] = jacobi_svd_2sided(A)
    [m1, n1] = size(A);
    m = 0;
    n = 0;
    flag = 0;
    S = A;
    if m1 >= n1
        m = m1;
        n = n1;
        S = A;
        flag = 1;
    else
        m = n1;
        n = m1;
        S = A.';
    end
    U = eye(m);
    V = eye(n);
    tolerance = 1e-3;
    while off_diagonal(S) > tolerance
        for p = 1 : n
            for q = p + 1 : n
                [c, s] = symmetrization(S(p, p), S(p, q), S(q, p), S(q, q));
                J1 = jMatrix(p, q, c, s, m);
                S = J1 * S;
                U = U * J1.';
                [c, s] = rotation(S, p, q);
                J2 = jMatrix(p, q, c, s, n);
                J4 = jMatrix(p, q, c, s, m);
                S = J4.' * S * J2;
                U = U * J4;
                V = V * J2;
            end
            for q = n + 1 : m
                [c, s] = non_squared_zero(S(p, p), S(q, p));
                J3 = jMatrix(p, q, c, s, m);
                S = J3 * S;
                U = U * J3.';
            end
        end
    end
    for i = 1 : n
        if S(i, i) < 0
            S(i, i) = - S(i, i);
            U(:, i) = - U(:, i);
        end
    end
    if flag == 0
        S = S.';
        temp = U;
        U = V;
        V = temp;
    end
end
```

first i handle this code for tall vertical matrices. then with a conditional statement handle it for all matrices, first transpose our tall horizental matrix and then use our algorithm (for vertical matrices) normally. after that we have an iteration (we must use a good condition). In this iteration first we must symmetrize 2 by 2 matrices in rows and columns p and q indices. after this we can use symschur algorithm for set off diagonal elements to zero (set a_pq and a_qp to zero). after do this with first n row, we must set elements in rows n + 1 to m, to zero. for this we can multiple our matrix to a jacobi matrix from left (it is like a Givens problem) and set a_pq to zero. the last thing we must do is put a good condition for break our while iteration. This condition is check off diagonal of our matrix. Because off diagonal of Sigma matrix is small.

function below is for calculate rotation parameters symmetrization

```
function [c, s] = symmetrization(s_pp, s_pq, s_qp, s_qq)
    t = 0;
    if s_pq ~= s_qp
        t = (s_qp - s_pq) / (s_pp + s_qq);
    end
    c = 1 / sqrt(1 + t ^ 2);
    s = c * t;
end
```

and function below is for calculate rotation parameters for set a_pq to zero

```
function [c, s] = non_squared_zero(a_pp, a_pq)
    t = 0;
    if a_pp ~= 0
        t = a_pq / a_pp;
    end
    c = 1 / sqrt(1 + t ^ 2);
    s = c * t;
end
```

at the last lines of jacobi_svd_2sided, we must check our initial matrix was tall vertical or tall horizental and reconstruct U, S and V matrices. S goes to S transpose, U goes to V and V goes to U.

Question3

```
function [U, S, V] = one_sided(A)
    [m1, n1] = size(A);
    m = 0;
    n = 0;
    flag = 0;
    US = A;
    if m1 >= n1
        m = m1;
        n = n1;
        flag = 1;
    else
        m = n1;
        n = m1;
        US = A.';
    end
    U = zeros(m, m);
    S = zeros(m, n);
    V = eye(n);
    while ~check_orthogonality(US)
        for p = 1 : n - 1
            for q = p + 1 : n
                c = 0;
                s = 0;
                x = US(:, p);
                y = US(:, q);
                a = y.' * x;
                b = y.' * y - x.' * x;
                t = 0;
                if a == 0
                    c = 1;
                    s = 0;
                else
                    t = min((- b + sqrt(b ^ 2 + 4 * a ^ 2)) / a, (-b - sqrt(b ^ 2 + 4 * a ^ 2)
                end
                c = 1 / sqrt(1 + t ^ 2);
                s = c * t;
                J = jMatrix(p, q, c, s, n);
                US = US * J;
                V = V * J;
            end
        end
    end
    v = singular_value(US).';
    US = US(:, v);
    V = V(:, v);
    for i = 1 : n
        s_i = norm(US(:, i), 2);
        if s_i ~= 0
```

```
                S(i, i) = s_i;
                U(:, i) = US(:, i) / s_i;
            end
        end
        U = orthogonal(U, rank(U));
        if flag == 0
            S = S.';
            temp = U;
            U = V;
            V = temp;
        end
    end
```

first i handle this code for tall vertical matrices. then with a conditional statement handle it for all matrices, first transpose our tall horizental matrix and then use our algorithm (for vertical matrices) normally. after that we have an iteration (we must use a good condition). In this iteration first we choose two columns and then orthogonalize these columns using jacobi matrices. after this we must sort our singular values descending order using singular_value function. after that we must fill zero columns with orthogonal columns .the last thing we must do is put a good condition for break our while iteration. For write this condition, i wrote below function

```
function bool = check_orthogonality(A)
    bool = false;
    [m, n] = size(A);
    for i = 1 : n
        A(:, i) = A(:, i) / norm(A(:, i), 2);
    end
    tolerance = 1e-6;
    if norm((A.' * A - eye(n)), "fro") < tolerance
        bool = true;
    end
end
```

and below function is singular_value function that used for sort singular values

```
function v = singular_value(A)
    [m, n] = size(A);
    S = zeros(n, 1);
    v = zeros(n, 1);
    for i = 1 : n
        S(i) = norm(A(:, i), 2);
        v(i) = i;
    end
    for i = 1 : n - 1
        for j = i + 1 : n
            if S(i) < S(j)
                temp = S(i);
                S(i) = S(j);
                S(j) = temp;
                temp = v(i);
                v(i) = v(j);
                v(j) = temp;
            end
        end
    end
end
```

and last function used for orthogonalize U. for this i use a random vector and then use Gram-Schmidt algorithm for orthogonalize U's columns.

```
function Q = orthogonal(A, r)
    [m, n] = size(A);
    Q = A;
    h = 1;
    for i = r + 1 : n
        q = zeros(m, 1);
        q(h) = 1;
        flag = 0;
        while flag == 0
            for j = 1 : i - 1
                if abs(abs(q.' * Q(:, j)) - norm(q, 2) .* norm(Q(:, j), 2)) < 1e-10
                    flag = 1;
                    break;
                end
            end
            if flag == 1
                q(h) = 0;
                q(h + 1) = 1;
                h = h + 1;
                flag = 0;
            else
                u = q;
                for j = 1 : i - 1
                    u = u - (u.' * Q(:, j)) * Q(:, j);
                end
                Q(:, i) = u / norm(u, 2);
                q(h) = 0;
                q(h + 1) = 1;
                h = h + 1;
                break;
            end
        end
    end
end
```

at the last lines of one_sided, we must check our initial matrix was tall vertical or tall horizental and reconstruct U, S and V matrices. S goes to S transpose, U goes to V and V goes to U.