The 4-variable quadratic equation can be written as \( x^T A x = 1 \) with
\[
A = \begin{bmatrix}
2 & -1 & 1 & 0 \\
-1 & 1 & -1 & 0 \\
1 & -1 & 3 & 0 \\
0 & 0 & 0 & 3
\end{bmatrix}
\]

The ellipsoid is the set: \( \mathcal{E} = \{ x \in \mathbb{R}^4 : x^T A x = 1 \} \)

For a singular values interpretation:

\( \mathcal{E} \) is also the image of a transformation

\[
X = B Z
\]

where \( \|Z\| = 1 \)

If we can find \( B \), its singular values will be the lengths of the ellipsoid's semi-axes.
Finding B: we need \( \| z \| ^2 = z^T z = 1 \)

\[ Z = B X \overset{-1}{\theta}, \quad z^T z = x^T B B^{-1} x = 1 \]

So \( x^T B^{-1} B x = x^T A x \)

Then we can choose \( A = B^{-1} B \)

call \( K = B^{-1} \), \( A = K^T K \).

Since \( A \) is positive-definite, it has a Cholesky decomposition \( A = K^T K \) (Matlab's `chol(A)`)

Once \( K \) is found, we use \( B = K^{-1} \).

In Matlab: \( K = \begin{bmatrix}
\sqrt{2} & -\sqrt{2}/2 & \sqrt{2}/2 & 0 \\
0 & \sqrt{2}/2 & -\sqrt{2}/2 & 0 \\
0 & 0 & \sqrt{2} & 0 \\
0 & 0 & 0 & \sqrt{3}
\end{bmatrix} \)

Decomposing \( B \) into its SVD:

\[ B = U \Sigma V^T \]
\[ B = k^{-1} = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & \sqrt{2}/2 & 0 \\ 0 & 0 & 0 & \sqrt{3}/2 \end{bmatrix} \]

\[ \begin{bmatrix} 0.4241 \\ -0.7392 \\ 0 \\ 0.5207 \end{bmatrix} \begin{bmatrix} 0.8777 \\ 0.2332 \\ 0 \\ -0.3971 \end{bmatrix} = x \]

\[ \begin{bmatrix} 0.1721 & 0.6318 & 0 & 0.7558 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = x \]

\[ \begin{bmatrix} 1.7545 \\ 0.8274 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.4371 \end{bmatrix} = x \]

\[ V^T \]
The lengths of the principal (not "principle")
axes are 1.7545, 0.8274, 0.5774, 0.4871.
The directions of the principal axes are the
columns of \(U\).

---

1. Alternative solution without Cholesky

\[ \mathcal{E} = \{ x \in \mathbb{R}^4 : x^T A x = 1 \} \]

\(A = A^T > 0\), there is a full set of
independent eigenvectors and corresponding
positive eigenvalues. The eigenvectors can
be used in a change of basis to diagonalize \(A\).

Let \(x = Uz\), where \(U\) contains the eigenvectors
in the columns (note \(U\) will be orthogonal).

Then \(x^T A x = z^T U^T A U z = z^T (U^{-1} A U) z = 1\)
\[ U^T A U \] will be diagonal: 
\[ U = \begin{bmatrix}
0.4271 & -0.7392 & 0 & 0.8207 \\
0.8877 & 0.2332 & 0 & -0.3571 \\
0.1721 & 0.6318 & 0 & 0.7558 \\
0 & 0 & 1 & 0
\end{bmatrix} \]

\[ U^T A U = \begin{bmatrix}
0.3249 & 1.4608 & 3 \\
\end{bmatrix} \]

\[ \begin{bmatrix}
D \\
\end{bmatrix} \]

\[ D = \begin{bmatrix}
4.2143
\end{bmatrix} \]

\[ \text{Z- coordinates: } \quad Z^T D Z = 1 \]

\[ Z_1^2 D(1,1) + Z_2^2 D(2,2) + Z_3^2 D(3,3) + Z_4^2 D(4,4) = 1 \]

\[ \frac{Z_1^2}{\left( \frac{1}{\sqrt{D(1,1)}} \right)^2} + \frac{Z_2^2}{\left( \frac{1}{\sqrt{D(2,2)}} \right)^2} + \ldots = 1 \]

So the lengths of the semi-axes are \( \sqrt{\frac{1}{D(i,i)}} \)
The $U$ vectors that map to these $V$ vectors from the unit 4-sphere after applying the transformation are:

$$
U_1 = \begin{bmatrix}
0.7494 \\
0.6597 \\
0.0566 \\
0.0000
\end{bmatrix}, \\
U_2 = \begin{bmatrix}
-0.6116 \\
0.6569 \\
0.4409 \\
0.0000
\end{bmatrix}, \\
U_3 = \begin{bmatrix}
0.0000 \\
0.0000 \\
0.0000 \\
1.0000
\end{bmatrix}, \\
U_4 = \begin{bmatrix}
0.2536 \\
-0.3650 \\
0.8958 \\
0.0000
\end{bmatrix}
$$

Problem 2

Determine the Jacobian (velocity and angular velocity) for the cylindrical manipulator of HW2 in symbolic form, referring to the wrist center. Symbolic solution with MATLAB highly suggested.

Cylindrical base

The cylindrical manipulator from homework 2:

```matlab
% Define constants and symbols
h = 1;
d = 2;
syms q1 q2 q3;
q = [q1;q2;q3];

% Define the links and robot
Link = sym(zeros(4,4,3));
Link(:,:,1) = RotationMatrix(3, q1) * TranslationMatrix([0 0 h]); % Cylindrical base
Link(:,:,2) = TranslationMatrix([0 0 q2])*RotationMatrix(1, -pi/2) ;
Link(:,:,3) = TranslationMatrix([0 0 q3]);
Robot = mprod(Link,3);

% Determine the Jacobian matrices
Jv = jacobian(Robot(1:3,4),q);
z = squeeze(Link(1:3,3,:));
Jw = z .* repmat([1 0 0],3,1);
J = [Jv;Jw];
```

We calculate the velocity Jacobian using the definition: partial derivative of the position vector with respect to each joint angle (MATLAB has a function built in for this). We also calculate the angular velocity Jacobian matrix using the short cut defined in class (using the third column of the rotation matrix, called $z$ in our notes and here). We then combine these into a single Jacobian matrix.

$$
J = \begin{bmatrix}
-q_3 c_i & 0 & -s_i \\
-q_3 s_i & 0 & c_i \\
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
1 & 0 & 0
\end{bmatrix}
$$
**Whole robot**

There was some confusion exactly what portion of the robot we needed the Jacobian of; here I do the same thing but applied to the whole robot.

% Define constants and symbols
h = 1;
d = sym('d6');
syms q1 q2 q3 q4 q5 q6;
%d = [q1;q2;q3;q4;q5;q6];

% Define the links and robot
Link = sym(zeros(4,4,6));
Link(:,:,1) = RotationMatrix(3, q1) * TranslationMatrix([0 0 h]); % Cylindrical base
Link(:,:,2) = TranslationMatrix([0 0 q2]) * RotationMatrix(1, -pi/2); % Spherical wrist
Link(:,:,3) = TranslationMatrix([0 0 q3]);
Link(:,:,4) = RotationMatrix(3, q4) * RotationMatrix(1, -pi/2); % Cylindrical base
Link(:,:,5) = RotationMatrix(3, q5) * TranslationMatrix([0 0 d]); % Spherical wrist
Link(:,:,6) = RotationMatrix(3, q6);
Robot = mprod(Link,3);
CumLink = cummprod(Link,3);

% Determine the Jacobian matrices
Jv = jacobian(Robot(1:3,4),q); % This method agrees with Jv calculated below
z = remdim(CumLink(1:3,3,:),2); z = [0;0;1 z(:,1:end-1)];
o = remdim(CumLink(1:3,4,:),2); o = [zeros(3,1) o(:,1:end-1)];
Jv = z .* (1-repmat([1 0 0 1 1 1],3,1)) -z.* (1-repmat([1 0 0 1 1 1],3,1));
Jw = z .* repmat([1 0 0 1 1 1],3,1);
J = simplify([Jv;Jw]);

The velocity Jacobian was calculated using the method described in the notes in this case, which was tested and it returns the same thing as using the jacobian function. The Jacobian matrix is found:

\[
J = \begin{bmatrix}
-d_6(c_1c_5+c_4s_1s_5)q_3c_1 & 0 & -s_1 & -d_6c_1s_4s_5 & d_6(s_1s_5+c_1c_4c_5) & 0 \\
-d_6(c_5s_1-c_1c_4s_5)q_3s_1 & 0 & c_1 & -d_6s_1s_4s_5 & d_6(c_4c_5s_1-c_1s_5) & 0 \\
0 & 1 & 0 & -d_6c_4s_5 & -d_6c_5s_4 & 0 \\
0 & 0 & 0 & -s_1 & -c_1s_4 & c_1c_4s_5-c_5s_1 \\
0 & 0 & 0 & c_1 & -s_1s_4 & c_1c_5+c_4s_1s_5 \\
1 & 0 & 0 & 0 & -c_4 & -s_4s_5
\end{bmatrix}
\]

**Problem 3**

For the 3×3 linear velocity Jacobian above, find all singularities and provide a graphical interpretation.

What is the rank of the 3×3 angular velocity Jacobian? Provide an interpretation.

Here we consider the cylindrical base, which has a 3×3 linear and angular velocity Jacobian matrices.

Jv_det = simplify(det(Jv)); % Result: q3
Jv_rank = rank(Jv); % Result: 3
Jw_rank = rank(Jw); % Result: 1

Singularities occur when \( q_3 = 0 \), the only solution to the determinant of the velocity Jacobian matrix being zero. Graphical interpretation: This makes sense because this variable represents radial offset.
from the central axis of the robot, and we know a radius of zero is a special point. If we rotate \( q_1 \) when \( q_3 \) is zero, we know no velocity is produced at the point of interest.

The rank of the angular velocity Jacobian matrix is 1. Mathematically, this means we have only a single independent vector. Physically, this is because \( q_1 \) is the only variable that can produce a rotation since it is a revolute joint; the other two variables \( q_2 \) and \( q_3 \) apply to prismatic joints, which are incapable of producing rotation by themselves.

**Problem 4**

For the 2-link planar manipulator, prove that Yoshikawa's manipulability measure is independent of the first joint coordinate (consider the 2×2 linear velocity Jacobian only).

\[
\mu = \det(J) = L_1 L_2 |\sin q_2|\\
\]

We see that \( \mu \) is only a function of \( q_2 \); it is independent of \( q_1 \).

**Problem 5**

For the 2-link planar manipulator, plot the manipulability ellipses (planar velocity only) at a few values of \( q_2 \) for \( q_1 = 0 \). Take link lengths equal to 1. Determine the “best” value of \( q_2 \) visually.
% Generate q data
N = 12;
q1 = 0;
q2 = linspace(-pi, pi, N+1); q2=q2(1:end-1);

% Generate unit circle
r = linspace(-pi, pi, 20);
uc = [cos(r) ; sin(r)];

% Plot
figure();
set(gcf, 'color', [1 1 1]);
hold('on');
axis square;
xlim([-0.5 2.5]);
for k = 1 : N

% Find points of interest on the ellipse
   c = double(subs(Robot2, q, [q1;q2(k)])); c = c(1:3,4);
p0 = [0;0;0];
p1 = double(subs(Point1, q, [q1;q2(k)])); p1 = p1(1:3,4);
p2 = double(subs(Point2, q, [q1;q2(k)])); p2 = p2(1:3,4);
p = [p0 p1 p2];

% Create ellipse points
Jv2s = double(subs(Jv2,q,[q1;q2(k)]));
v = zeros(size(u));
for n = 1 : length(t)
   v(:,n) = Jv2s*u(:,n);
end

% Find the ellipse axes
[U,S,V] = svd(Jv2s);
Vmax = Jv2s*V(:,1);
Vmin = Jv2s*V(:,2);

% Plot robot, ellipse, and ellipse axes
plot(p(1,:), p(2,:), 'k.-');
plot(c(1) + scale*v(1,:), c(2) + scale*v(2,:), 'b');
plot(c(1) + scale*Vmax(1)*[-1 1], c(2) + scale*Vmax(2)*[-1 1], 'r-');
plot(c(1) + scale*Vmin(1)*[-1 1], c(2) + scale*Vmin(2)*[-1 1], 'r-');
end
Looking at this figure, eyeballing for the largest area of an ellipse, it looks like when the second link is in one of the two vertical positions ($q_2$ is $\pm \pi/2$), we have maximum manipulability. Smaller $q_2$ appear to be thinner, while larger appear to be shorter.

**Problem 6**

Assuming the statement in Problem 4 is true, find the value of $q_2$ maximizing $\mu$ analytically. What is $\mu$'s maximum value?

```matlab
% Substitute lengths
mu2 = subs(mu, L, [L1;L2]);

% Add the last point
q2b = [q2 pi];

% Substitute q value into mu formula
mu3 = double(subs(mu2, q(2), q2b));

% Plot
figure();
set(gcf, 'color', [1 1 1]);
plot(q2b, mu3);
xlabel('$q_2$');
ylabel('$\mu$');
```

Page 7 of 10
Just by looking at the plot, it looks like the maximum is at the suspected $\pm/2$. The value of $\mu$ is 1 at these points.

\[
\frac{\partial \mu}{\partial q_2} = \text{sgn}(\sin(q_2)) \cos(q_2)
\]

To symbolically determine the maximum, we take the derivative with respect to $q_2$, and set it equal to zero. This is satisfied only in the case when the sine of $q_2$ is zero, or the cosine of $q_2$ is zero; i.e. $\forall n \in \mathbb{Z}$, $q_2 = 2\pi n/4$. We can show that at only for $n=\pm1$ (when restricting ourselves from $-\pi$ to $\pi$) this is a maximum by taking the second derivative and checking it is negative, though this won't help with the discontinuous points; we can use intuition when looking at the plot to not formally/rigorously see that the maximums are correct.

Note that when considering physical capabilities, the arm cannot go underground; only one of these answers is physically possible.

**Functions**

Throughout this document, some functions are defined which may be unclear. They are defined here.

**mprod: matrix product**

\[
\text{function out = mprod(M,dim,direction)}
\]

% Default argument for direction is 'right'
if (nargin < 3)
    direction = 'right';
end
% Create indexing cells
S = cell(ndims(M),1);
for j = 1 : ndims(M)
    if (j == dim)
        S{j} = 0;
    else
        S{j} = ':';
    end
end

% Determine the output size
outsize = size(M);
outsize = [outsize(1:dim-1) outsize(dim+1:end)];
out = eye(outsize);

% Calculate output
for k = 1 : size(M,dim)
    S{dim} = k;
    Mk = subsref(M, struct('type', '()', 'subs', {S}));
    if (strcmp(direction,'right'))
        out = out*Mk;
    elseif (strcmp(direction,'left'))
        out = Mk*out;
    else
        error('Invalid direction.');</nend
end

cummprod: cumulative matrix product
function out = cummprod(M,dim,direction)

% Default argument for direction is 'right'
if (nargin < 3)
    direction = 'right';
end

% Create indexing cells
S = cell(ndims(M),1);
for j = 1 : ndims(M)
    if (j == dim)
        S{j} = 0;
    else
        S{j} = ':';
    end
end

% Determine the output size
out = zeros(size(M));

% Calculate output
sizes = size(M);
sizes = [sizes(1:dim-1) sizes(dim+1:end)];
last = eye(sizes);
for k = 1 : size(M,dim)
    S{dim} = k;
    Mk = subsref(M, struct('type', '()', 'subs', {S}));
    if (strcmp(direction,'right'))
        last = last*Mk;
    elseif (strcmp(direction,'left'))
        last = Mk*last;
    else
        error('Invalid direction.');</nend
out = subsasgn(out, struct('type', '()', 'subs', {S}), last);
end

Page 9 of 10
remdim: remove dimension

function out = remdim(M, dim, index)
    if (nargin < 2)
        dim = ndims(M);
    end

    if (nargin < 3)
        index = 1;
    end

    if (length(dim) == 1)
        outsize = size(M);
        outsize = [outsize(1:dim-1) outsize(dim+1:end)];
        S = cell(ndims(M), 1);
        for j = 1 : ndims(M)
            if (j == dim)
                S{j} = index;
            else
                S{j} = ':';
            end
        end
        Msub = subsref(M, struct('type', '()', 'subs', {S}));
    else
        out = reshaped(Msub, outsize);
    end

    else
        out = M;
        for k = 1 : length(dim)
            out = remdim(out, dim(k));
        end
    end