

# E 1

---

**a**

The code used to solve the problem is the following

```
%% Inputs
V = [80, 60, 40]'; % mm
de1 = deg2rad(+42);
de2 = deg2rad(-49);

DCM_a = eul2rotm([0, de1, de2], 'ZYX');
V_a = DCM_a * V;
VA = V_a - V;
```

Which provides the following value for **VA**, in mm.

```
VA =
    -33.2888
     9.5519
   -107.6801
```

**b**

The code used to solve the problem is the following

```
%% Inputs
V = [80, 60, 40]';

dx1 = deg2rad(-27);
dy1 = deg2rad(+65);

DCM_b = eul2rotm([0, dy1, dx1], 'ZYX');
V_b = DCM_b * V;
VB = V_b - V;
```

Which provides the following value for **VB**, in mm.

```
VB =
   -38.5768
```

11.6200  
-108.9543

**c**

The code used to solve the problem is the following

```
%% Inputs

dx1 = deg2rad(-27);
dy1 = deg2rad(+65);

DCM_b = eul2rotm([0, dy1, dx1], 'ZYX');

[Ang_c, PRV_c] = epvFromDCM2(DCM_b);
```

Which provides the following values for **Ang\_c**, in rad, and **PRV\_c**.

```
Ang_c =
    1.2185

PRV_c =
   -0.3441
    0.9130
    0.2192
```

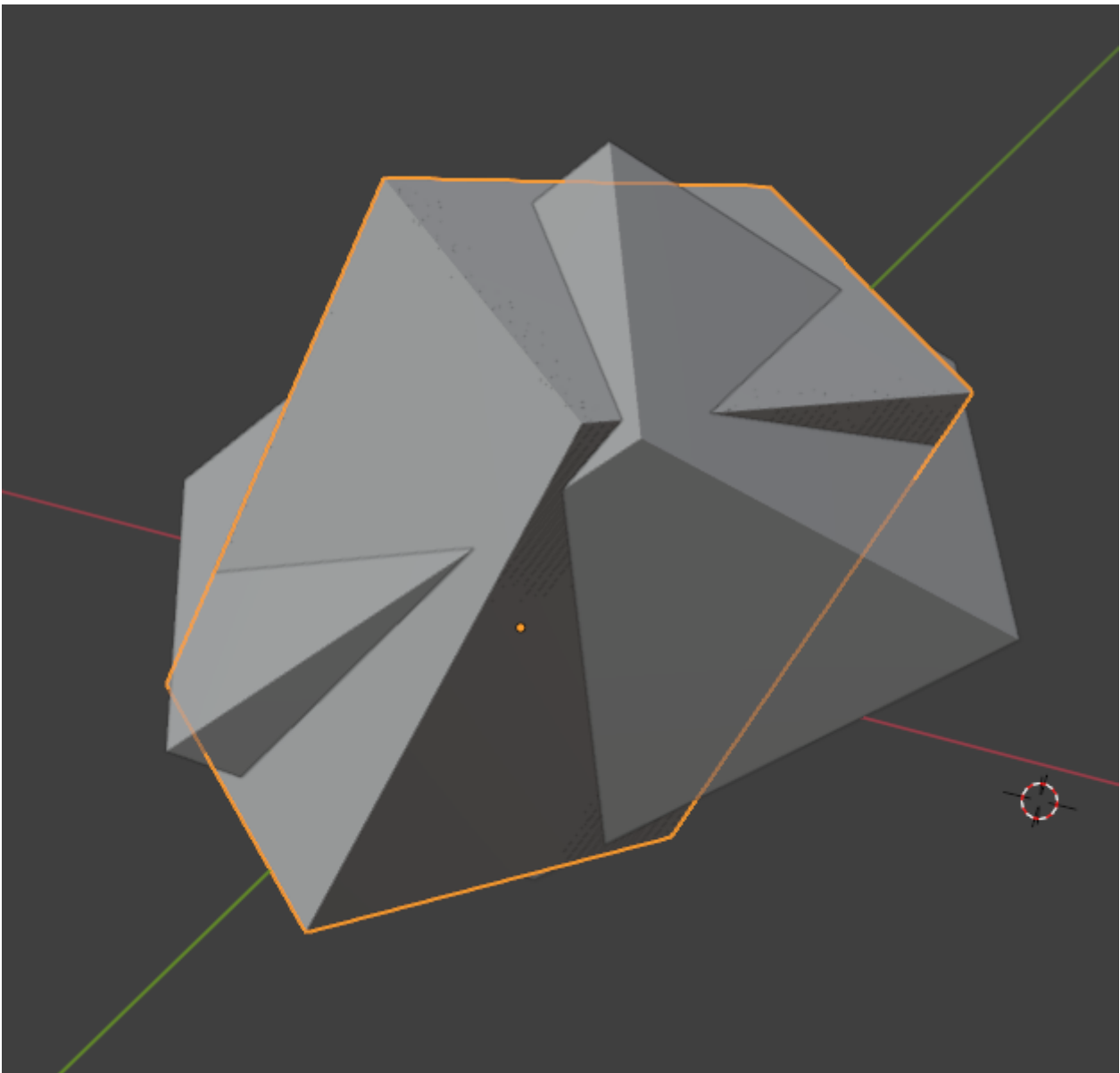
**d**

Using blender, you design the shape with the sizes on the drawing.

**For a**, you use the shortcut R, y, 42, <\CR>. This rotates around the global y axis. Then, R, x, x, -49, <\CR>. This rotates around the local x axis. Finally, enter on edit mode, select the desired vertex of the box and check the position. Remember to select global mode.

**For b**, similar procedure but without doing double y for using always global.

**For c**, we get the rotation vector from our code and on the rotation side bar we select rotation by axis angle. There, we use the x,y,z coordinates and angle of the matlab code. We observe the same result.



These procedures validated all the results.

## Note

`eul2rotm` is a function provided by matlab but `epvFromDCM2` not. This is its implementation:

```
function [angle, PRV] = epvFromDCM2(DCM)
%EPVFROMDCM2 Summary of this function goes here
% Detailed explanation goes here
[V, D] = eig(DCM); % V are eigenvectors, D is the diagonal matrix of
eigenvalues

% Find the eigenvector corresponding to the eigenvalue 1 (principal
axis)
[~, idx] = min(abs(diag(D) - 1)); % Find the index where eigenvalue is 1
PRV = V(:, idx); % Extract the corresponding eigenvector

% 2. Compute the rotation angle
trace_DCM = trace(DCM); % Trace of the DCM matrix
```

```
angle = acos((trace_DCM - 1) / 2); % Rotation angle in radians
end
```

## E 2

---

The code used to solve the problem is the following:

```
%% Inputs
MRP_endurance = [0.3353 0.1944 0.5528];
EP_Lander = [-0.5417 -0.2418 0.6654 0.4545];

%% Calculations
% First, we get the DCM for both rotations
Endurance_N = epToDCMVec(epFromRodrigues(MRP_endurance));
Lander_N = epToDCMVec(EP_Lander);
% Second, we compute the DCM from the Endurance to the Lander
Lander_Endurance = Lander_N * Endurance_N';
% Finally, we get the angle and principal axis
[angle, principal_axis] = epvFromDCM(Lander_Endurance);
```

Which finish with the following results

```
[2.722002257670760 0.557236265353963 0.806880451497850
0.215675406315517]
```

## E 3

---

The code used to solve this problem is the following:

```
%% Inputs
initial_angles = deg2rad([10 20 40]);
ang_vel = deg2rad([5 10 15]);
tot_ang_vel = norm(ang_vel);

EP_0 = epFromDCMSheppard(eul2rotm(initial_angles, 'ZYX'))'; % <<<<<<<<<
(a)

%% Calculation
tspan = [0 60];
[t, EP_n] = ode45(@(t, s) quaternion_derivative(s, ang_vel), tspan,
EP_0);
EP_n_norm = quaternion(EP_n(end, :))./ norm(EP_n(end, :)); % <<<<<<<<<<<<
(b)
```



```
EP_n_norm =  
    -0.93848 + 0.26112i - 0.092583j - 0.20614k
```

c

```
relative_error(end) =  
    2.4648e-04
```

d

```
exact_to_integrated_error =  
    7.4160e-04
```

## C 1

---

The code used to solve the problem is the following:

```
%% Inputs  
initialMRP = [0,0,0]';  
angVel = [-2, 1, 0.5]';  
tspan = [0 60];  
q0 = epFromRodrigues(initialMRP);  
  
der_mrp = @(mrp, ang) 0.25*((1-norm(mrp)^2)*ang+ 2*cross(mrp, ang)+  
    2*mrp*(dot(mrp, ang)));  
  
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);  
[t, q] = ode45(@(t, q) quaternion_derivative(q, angVel), tspan, q0,  
    options);  
  
q_mrp = rodriguesFromEp(q(end,:)/norm(q(end,:))); % ----- (a)  
  
error = []  
for i = 1:length(t)  
    error(i)=abs(norm(q(i,:))-1);  
end  
% plot(t, error);  
  
different_dt = [0.1, 0.01, 0.001];  
final_mrp = zeros([3, 3]);  
final_mrp_q = zeros([3, 4]);  
final_differences = zeros([3,1]);  
for i = 1:3
```

```

current_t = 0;
current_p = initialMRP;
while current_t < tspan(2)
    current_p = current_p + der_mrp(current_p,
angVel)*different_dt(i);
    if norm(current_p) > 1
        current_p = shadowRodriguesFromRodrigues(current_p);
    end
    current_t = current_t + different_dt(i);
end
final_mrp(i,:) = current_p; % ----- (b)
final_differences(i) = norm(q(end,:)-epFromRodrigues(current_p)); %
----- (c)
end

```

**a**

```

q_mrp =
    0.1662    -0.0831    -0.0416

```

**b**

```

% 10-1
0.2029    -0.1014    -0.0507
% 10-2
0.1615    -0.0808    -0.0404
% 10-3
0.1657    -0.0829    -0.0414

```

**c**

```

% 10-1
0.0803120556921397
% 10-2
0.0104756213489857
% 10-3
0.00111875657883155

```

## NOTE

The functions used are the following:

```

function dqdt = quaternion_derivative(q, omega)
    qq = quaternion(q(1), q(2), q(3), q(4));
    o = quaternion(0, omega(1), omega(2), omega(3));
    dq = 0.5*qq*o;
    [a0, a1, a2, a3] = dq.parts;
    dqdt = [a0, a1, a2, a3]';

end

function [sh] = shadowRodriguesFromRodrigues(s)
    s2 = s(1)^2 + s(2)^2 + s(3)^2;
    sh = -s./s2;

end

function [b] = epFromRodrigues(s)
    s2 = s(1)^2 + s(2)^2 + s(3)^2;
    b = [1-s2 2*s(1) 2*s(2) 2*s(3)]./(1+s2);

end

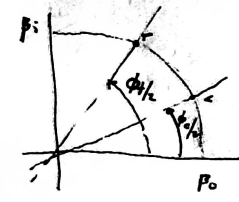
function [s] = rodriguesFromEp(b)
    s = [b(1+1) b(2+1) b(3+1)]./(1+b(0+1));

end

```

A 1

$P_1 A_1$



$\hat{e}_0 = \hat{e}_1$   
assumption

$t = \phi/2$   
 $c' = \phi_c/2$

$c = \cos$   
 $s = \sin$

$e_1^2 + e_2^2 + e_3^2 = 1$

$\begin{cases} \sin(A) - \sin(B) = 2 \cos\left(\frac{A+B}{2}\right) \sin\left(\frac{A-B}{2}\right) \\ \cos(A) - \cos(B) = -2 \sin\left(\frac{A+B}{2}\right) \sin\left(\frac{A-B}{2}\right) \end{cases}$

(a)  $\tilde{E}_{Pct} = \begin{pmatrix} ct - cc' \\ e_1(t - \phi_c') \\ e_2(t - \phi_c') \\ e_3(t - \phi_c') \end{pmatrix} \xrightarrow{\text{small angle}} \begin{pmatrix} -2s \frac{t+c'}{2} \\ e_1(2c \frac{t+c'}{2}) \\ e_2(2c \frac{t+c'}{2}) \\ e_3(2c \frac{t+c'}{2}) \end{pmatrix} \xrightarrow{\text{small angle}} \begin{pmatrix} -s \frac{t+c'}{2} & \frac{t+c'}{2} \\ e_1(c \frac{t+c'}{2}) & t-c' \\ e_2(c \frac{t+c'}{2}) & t-c' \\ e_3(c \frac{t+c'}{2}) & t-c' \end{pmatrix}$

$\|\tilde{E}_{Pct}\| \approx t - c' = \|E_{Pct}\|$  Same magnitude under small angle diff

$E_{Pct} = \begin{pmatrix} c(t-c') \\ e_1(t-c') \\ e_2(t-c') \\ e_3(t-c') \end{pmatrix} \xrightarrow{\text{small angle}} \begin{pmatrix} 0 \\ e_1(t-c') \\ e_2(t-c') \\ e_3(t-c') \end{pmatrix}$

components are very different, the exact approx has the angles on the  $P_1:3$  components while the approximation has it projected back  $P_1$  and  $P_0$

A 2



