Homework 1

Problem 1

From the description of the function $f(A)$, we can know that we need to first find the two eiganvalues of the matrix A:

$$A = \begin{bmatrix} 0 & a \\ -a & 0 \end{bmatrix}$$

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} 0 & a \\ -a & 0 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) = 0$$

$$\det\begin{bmatrix} -\lambda & a \\ -a & \lambda \end{bmatrix} = 0$$

$$\lambda^2 + a^2 = 0$$

From this polynomial equation, we can find two values for eiganvalues: $\lambda_1 = -ai, \lambda_2 = ai$. By taking them into the below equation, we can find their corresponding eiganvectors:

$$(A - \lambda I) = 0$$

when $\lambda = -ai$, we have:

$$\begin{bmatrix} ai & a \\ -a & ai \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

and finally $Q_1 = \begin{bmatrix} x_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -i \end{bmatrix}$. Similarly, if we plug in the second eiganvalue, we can find the second eiganvector Q_2:

$$\begin{bmatrix} -ai & a \\ -a & -ai \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0, Q_2 = \begin{bmatrix} 1 \\ i \end{bmatrix}$$

With the given function $f(A)$, we can have:

$$f(A) = Qf(\Lambda)Q^{-1} = Q\ \text{diag}(f(\lambda_1), f(\lambda_2))Q^{-1}$$

where $Q = [Q_1\ \ Q_2] = \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix}$. To calculate its inverse, we have:

$$\det(Q) = (1 \cdot i) - (1 \cdot -i) = 2i$$

$$\text{Adj}(Q) = \begin{bmatrix} i & -1 \\ i & 1 \end{bmatrix}$$

$$Q^{-1} = \frac{1}{\det(Q)}\ \text{Adj}(Q) = \begin{bmatrix} \frac{1}{2} & \frac{i}{2} \\ \frac{1}{2} & -\frac{i}{2} \end{bmatrix}$$

Finally, we have:

$$
\begin{aligned}
f(A) &= \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix} \begin{bmatrix} e^{-ai} & 0 \\ 0 & e^{ai} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{i}{2} \\ \frac{1}{2} & -\frac{i}{2} \end{bmatrix} \\
&= \begin{bmatrix} e^{-ai} & e^{ai} \\ -ie^{-ai} & ie^{ai} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{i}{2} \\ \frac{1}{2} & -\frac{i}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}e^{-ai} + \frac{1}{2}e^{ai} & \frac{i}{2}e^{-ai} + \frac{i}{2}e^{ai} \\ -\frac{i}{2}e^{-ai} + \frac{i}{2}e^{-ai} & \frac{1}{2}e^{-ai} + \frac{1}{2}e^{-ai} \end{bmatrix}
\end{aligned}
$$

By the definition of $\sin(x)$ and $\cos(x)$, we can simplify it and get the final result:

$$
f(A) = \begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix}
$$

Problem 2

(a) Partial derivative of $f(x, y)$ with respect of r is:

$$
\begin{aligned}
\frac{\delta f}{\delta r} &= \frac{\delta f}{\delta x} \cdot \frac{\delta x}{\delta r} + \frac{\delta f}{\delta y} \cdot \frac{\delta y}{\delta r} \\
&= \frac{\delta f}{\delta x} \cdot \cos(\theta) + \frac{\delta f}{\delta y} \cdot \sin(\theta) \\
&= f_x \cdot \cos(\theta) + f_y \cdot \sin(\theta)
\end{aligned}
$$

(b) Partial derivative of $f(x, y)$ with respect of θ is:

$$
\begin{aligned}
\frac{\delta f}{\delta \theta} &= \frac{\delta f}{\delta x} \cdot \frac{\delta x}{\delta \theta} + \frac{\delta f}{\delta y} \cdot \frac{\delta y}{\delta \theta} \\
&= \frac{\delta f}{\delta x} \cdot -r\sin(\theta) + \frac{\delta f}{\delta y} \cdot r\cos(\theta) \\
&= f_x \cdot -r\sin(\theta) + f_y \cdot r\cos(\theta)
\end{aligned}
$$

Problem 3

(a)(1) With the definition of norm $\|\boldsymbol{q}\| = \sqrt{\boldsymbol{q}^T\boldsymbol{q}}$, we can simplify the function f as $f(\boldsymbol{q}) = \frac{1}{2}\log(\boldsymbol{q}^T\boldsymbol{q})$. By chain rule, we can have:

$$
\frac{df}{d\boldsymbol{q}} = \frac{\delta f}{\delta u} \cdot \frac{\delta u}{\delta \boldsymbol{q}}
$$

where $u = \boldsymbol{q}^T\boldsymbol{q}$. By calculating each part, we get:

$$
\begin{aligned}
\frac{\delta u}{\delta \boldsymbol{q}} &= 2\boldsymbol{q} \\
\frac{\delta f}{\delta u} &= \frac{1}{2} \cdot \frac{1}{u}\ln(e) = \frac{1}{2u} \\
\frac{df}{d\boldsymbol{q}} &= \frac{2\boldsymbol{q}}{2\,\|\boldsymbol{q}\|^2} = \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|^2}
\end{aligned}
$$

(a)(2) We can first change the format of the function g to the following:

$$g(\boldsymbol{q}) = \frac{\boldsymbol{q}_v}{\|\boldsymbol{q}_{\boldsymbol{v}}\|} = \boldsymbol{q}_{\boldsymbol{v}} \|\boldsymbol{q}_{\boldsymbol{v}}\|^{-1}$$

Now we implement the product rule to calculate its derivative:

$$\begin{aligned}\frac{dg(\boldsymbol{q})}{d\boldsymbol{q}} &= \boldsymbol{q}_v \frac{d\ \|\boldsymbol{q}_v\|^{-1}}{d\boldsymbol{q}} + \frac{d\boldsymbol{q}_v}{d\boldsymbol{q}}\ \|\boldsymbol{q}_v\|^{-1} \\ &= \boldsymbol{q}_v U + I\|\boldsymbol{q}_v\|^{-1}\end{aligned}$$

Now we calculate the value for U:

$$U = \frac{d\ \|\boldsymbol{q}_v\|^{-1}}{d\boldsymbol{q}}, \text{and we set } A = \|\boldsymbol{q}_v\|^{-1} = \left(\boldsymbol{q}_v^T \boldsymbol{q}_v\right)^{-\frac{1}{2}}$$

$$\frac{dA}{d\boldsymbol{q}_v} = -\|\boldsymbol{q}_v\|^{-3}\boldsymbol{q}_v$$

Thus we can have the final result:

$$\begin{aligned}\frac{dg(\boldsymbol{q})}{d\boldsymbol{q}} &= -\boldsymbol{q}_v\ \|\boldsymbol{q}_v\|^{-3}\boldsymbol{q}_v + I\|\boldsymbol{q}_v\|^{-1} \\ &= -\boldsymbol{q}_v^T\boldsymbol{q}_v\ \|\boldsymbol{q}_v\|^{-3} + I\|\boldsymbol{q}_v\|^{-1}\end{aligned}$$

(a)(3) For $\arccos(x)$, we have the following formula as its derivative:

$$\arccos'(x) = -\frac{1}{\sqrt{1-x^2}}$$

By taking our matrix into it, we can have the followings:

$$\frac{dh(\boldsymbol{q})}{d\boldsymbol{q}} = -\frac{1}{\sqrt{1-u^2}}, \text{ where } \ u = q_s\ \|\boldsymbol{q}\|^{-1}$$

By chain rule, we can know that $\frac{dh(\boldsymbol{q})}{d\boldsymbol{q}} = \frac{dh(\boldsymbol{q})}{du} \cdot \frac{du}{d\boldsymbol{q}}$. We have calculates the value for $\frac{du}{d\boldsymbol{q}}$ in the previous questions, which gives us:

$$\begin{aligned}\frac{dh(\boldsymbol{q})}{d\boldsymbol{q}} &= \frac{dh(\boldsymbol{q})}{du} \cdot \frac{du}{d\boldsymbol{q}} \\ &= -\frac{1}{\sqrt{1-u^2}} \cdot q_s - \|\boldsymbol{q}\|^{-3}\boldsymbol{q} \\ &= \frac{q_s\ \|\boldsymbol{q}\|^{-3}}{\sqrt{1-\left(q_s\ \|\boldsymbol{q}\|^{-1}\right)^2}} \cdot \boldsymbol{q}\end{aligned}$$

(b) From the below screenshot, we can see that the results matched:

[1]:
```
import torch
def logorithm(q):
    return torch.log(torch.norm(q))
def fraction(qv):
    return (qv/(torch.norm(qv)))
def arccos(q):
    qs = 1.0
    return torch.acos(qs/(torch.norm(q)))
```

[2]:
```
q = torch.tensor([[1.], [2.], [3.], [4.]])

jacobian_log_torch = torch.autograd.functional.jacobian(logorithm, q)
print("PyTorch Version: \n", jacobian_log_torch)

norm_q = torch.norm(q)
jacobian_log_manual = q / torch.pow(norm_q,2)
print("Manual Version: \n", jacobian_log_manual)
```

```
PyTorch Version:
 tensor([[0.0333],
        [0.0667],
        [0.1000],
        [0.1333]])
Manual Version:
 tensor([[0.0333],
        [0.0667],
        [0.1000],
        [0.1333]])
```

[3]:
```
qv = torch.tensor([[2.], [3.], [4.]])

jacobian_fraction_torch = torch.autograd.functional.jacobian(fraction, qv)
jacobian_fraction_torch = jacobian_fraction_torch.squeeze()
print("PyTorch Version: \n", jacobian_fraction_torch)

norm_qv = torch.norm(qv)
I = torch.eye(3)
first_term = -qv @ qv.T * torch.pow(norm_qv, -3)
second_term = I*torch.pow(norm_qv, -1)
jacobian_fraction_manual = first_term + second_term
print("Manual Version: \n", jacobian_fraction_manual)
```

```
PyTorch Version:
 tensor([[ 0.1601, -0.0384, -0.0512],
        [-0.0384,  0.1281, -0.0768],
        [-0.0512, -0.0768,  0.0832]])
Manual Version:
 tensor([[ 0.1601, -0.0384, -0.0512],
        [-0.0384,  0.1281, -0.0768],
        [-0.0512, -0.0768,  0.0832]])
```

[4]:
```
qs = 1.0

jacobian_fraction_torch = torch.autograd.functional.jacobian(arccos, q)
print("PyTorch Version: \n", jacobian_fraction_torch)

norm_q = torch.norm(q)
scaler_part = (qs*torch.pow(norm_q, -3))/(torch.sqrt(1-torch.pow(qs*torch.pow(norm_q, -1),2)))
jacobian_fraction_manual = scaler_part*q
print("Manual Version: \n", jacobian_fraction_manual)
```

```
PyTorch Version:
 tensor([[0.0062],
        [0.0124],
        [0.0186],
        [0.0248]])
Manual Version:
 tensor([[0.0062],
        [0.0124],
        [0.0186],
        [0.0248]])
```

Problem 4

In Gauss-Newton method, we need first find the function $e(x) \in \mathbb{R}$. In this problem, the best choice is $\sin(x) - \frac{1}{2}$. To find its Jacobian matrix, we take its derivative, which is $\cos(x)$. Thus, we can form the following equation to find the descent direction, δx_k:

$$\begin{aligned}
\delta x_k &= -\left(J^T J\right)^{-1}\left(J^T e(x_k)\right) \\
&= -\left(\cos(x_k)^T \cos(x_k)\right)^{-1}\left(\cos(x_k)^T\left(\sin(x_k) - \frac{1}{2}\right)\right) \\
&= -\frac{\cos(x_k)\left(\sin(x_k) - \frac{1}{2}\right)}{\cos(x_k)\cos(x_k)} \\
&= -\frac{\sin(x_k) - \frac{1}{2}}{\cos(x_k)}
\end{aligned}$$

With the fixed and given step size $\alpha = \frac{1}{2}$, we can calculate the following steps:

$$\begin{aligned}
x_0 &= 2 \\
x_1 &= 2 - \frac{1}{2} \times -\frac{\sin(2) - \frac{1}{2}}{\cos(2)} = 2.49177 \\
x_2 &= 2.49177 - \frac{1}{2} \times -\frac{\sin(2.49177) - \frac{1}{2}}{\cos(2.49177)} = 2.55774
\end{aligned}$$

Problem 5

This problem is solved largely relying on Python.

In order to convert the coordinate of x from frame {C} to frame {A}, we need to first convert it to frame {B}. Since the parametrization method used from frame {B} to frame {C} is axix-angle, we frist calculate the rotation matrix in frame {B}:

$$R = \exp\left(\hat{\theta}\right)$$

where $\hat{\theta} = \theta\boldsymbol{\eta} = \frac{\pi}{6} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}^T = \begin{bmatrix} \frac{\pi}{6\sqrt{2}} & \frac{\pi}{6\sqrt{2}} & 0 \end{bmatrix}^T$. Using Rodrigues formula, we can calculate the rotation matrix in frame {B}:

$$R = I + \left(\frac{\sin(\|\theta\|)}{\|\theta\|}\right)\hat{\theta} + \left(\frac{1-\cos(\|\theta\|)}{\|\theta\|^2}\right)\hat{\theta}^2$$

Using Python, we can get the calculated matrix:

$$R = \begin{bmatrix} 0.9330 & 0.0670 & 0.3536 \\ 0.0670 & 0.9330 & -0.3536 \\ -0.3536 & 0.3536 & 0.8660 \end{bmatrix}$$

We can also calculate the matrix T to get the coordinate of x in frame {B}:

$$\begin{aligned} {}_{\{B\}}T_{\{C\}} &= \begin{bmatrix} R & P \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} s_C \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 1.3536 \\ 1.6464 \\ 2.8660 \end{bmatrix} \end{aligned}$$

Now we get the coordinate of x in frame {B}, we can try to convert it to frame {A}. In this transition, we used Euler-Angle method, which needs to multiply rotation axis metrices:

$$R_{x(\varphi)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix} \qquad R_{y(\theta)} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \qquad R_{z(\psi)} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Using Python, we can calculate the final coordinateeasily using the following equation:

$$sxyz = rzyx$$

and get the final coordinate:

$$\begin{aligned} {}_{\{A\}}T_{\{B\}} &= R_{z(\psi)}R_{y(\theta)}R_{x(\varphi)} \begin{bmatrix} 1.3536 \\ 1.6464 \\ 2.8660 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 3.5077 \\ 3.4976 \\ 0.4804 \end{bmatrix} \end{aligned}$$

Appendix

I have pasted the Python codes used in solving above questions below for checking.

Used in Problem 3

```
import torch
def logorithm(q):
    return torch.log(torch.norm(q))
def fraction(qv):
    return (qv/(torch.norm(qv)))
def arccos(q):
    qs = 1.0
    return torch.acos(qs/(torch.norm(q)))
q = torch.tensor([[1.], [2.], [3.], [4.]])

jacobian_log_torch = torch.autograd.functional.jacobian(logorithm, q)
print("PyTorch Version: \n", jacobian_log_torch)

norm_q = torch.norm(q)
jacobian_log_manual = q / torch.pow(norm_q,2)
print("Manual Version: \n", jacobian_log_manual)
qv = torch.tensor([[2.], [3.], [4.]])

jacobian_fraction_torch = torch.autograd.functional.jacobian(fraction, qv)
jacobian_fraction_torch = jacobian_fraction_torch.squeeze()
print("PyTorch Version: \n", jacobian_fraction_torch)

norm_qv = torch.norm(qv)
I = torch.eye(3)
first_term = -qv @ qv.T * torch.pow(norm_qv, -3)
second_term = I*torch.pow(norm_qv, -1)
jacobian_fraction_manual = first_term + second_term
print("Manual Version: \n", jacobian_fraction_manual)
qs = 1.0

jacobian_fraction_torch = torch.autograd.functional.jacobian(arccos, q)
print("PyTorch Version: \n", jacobian_fraction_torch)

norm_q = torch.norm(q)
scaler_part = (qs*torch.pow(norm_q, -3))/(torch.sqrt(1-torch.pow(qs*torch.pow(norm_q,
-1),2)))
jacobian_fraction_manual = scaler_part*q
print("Manual Version: \n", jacobian_fraction_manual)
```

Used in Problem 4

```
import math

x_lst = [2]
for x in range(1,11):
    xk = x_lst[x-1]
    delta = -(math.sin(xk)-0.5)/(math.cos(xk))
    xkp1 = xk + 0.5 * delta
```

```
    x_lst.append(xkp1)

print(x_lst)
```

Used in Problem 5

```
import math
vector_theta = torch.tensor([[(math.pi)/(6*math.sqrt(2))],
                             [(math.pi)/(6*math.sqrt(2))],
                             [0]])
hat_theta = torch.tensor([[0,0,(math.pi)/(6*math.sqrt(2))],
                          [0,0,(math.pi)/(-6*math.sqrt(2))],
                          [-(math.pi)/(6*math.sqrt(2)),(math.pi)/(6*math.sqrt(2)),0]])
norm_theta = torch.norm(vector_theta)

I = torch.eye(3)
rotation_b = I + ((math.sin(norm_theta))/(norm_theta))*hat_theta+((1-math.cos(norm_theta))/
(norm_theta**2))*(hat_theta@hat_theta)

print(rotation_b)
print(norm_theta)
x_c = torch.tensor([[1.],[1.],[1.]])
t_c = torch.tensor([[0.],[1.],[2.]])

x_b = rotation_b @ x_c + t_c
print(x_b)
phi = math.pi/6
theta = math.pi/3
psi = math.pi/4

r_x = torch.tensor([
    [1.0, 0.0, 0.0],
    [0.0, math.cos(phi), -math.sin(phi)],
    [0.0, math.sin(phi), math.cos(phi)]
])

r_y = torch.tensor([
    [math.cos(theta), 0.0, math.sin(theta)],
    [0.0, 1.0, 0.0],
    [-math.sin(theta), 0.0, math.cos(theta)]
])

r_z = torch.tensor([
    [math.cos(psi), -math.sin(psi), 0.0],
    [math.sin(psi), math.cos(psi), 0.0],
    [0.0, 0.0, 1.0]
])

rotation_a = r_z @ r_y @ r_x
print(rotation_a)
phi = math.pi/6
theta = math.pi/3
```

```
psi = math.pi/4

r_x = torch.tensor([
    [1.0, 0.0, 0.0],
    [0.0, math.cos(phi), -math.sin(phi)],
    [0.0, math.sin(phi), math.cos(phi)]
])

r_y = torch.tensor([
    [math.cos(theta), 0.0, math.sin(theta)],
    [0.0, 1.0, 0.0],
    [-math.sin(theta), 0.0, math.cos(theta)]
])

r_z = torch.tensor([
    [math.cos(psi), -math.sin(psi), 0.0],
    [math.sin(psi), math.cos(psi), 0.0],
    [0.0, 0.0, 1.0]
])

rotation_a = r_z @ r_y @ r_x
print(rotation_a)

t_b = torch.tensor([[1],[1],[0]])
x_a = rotation_a @ x_b + t_b
print(x_a)
```