Attitude_Determination_Methods

July 24, 2020

1 Methods for Attitude Determination:

[1]:
```
import numpy as np
import math
from DCMmatrix import *
from Euler_Integrator import *
from scipy.linalg import *
from scipy import optimize
```

1.1 Concept Check 2 - TRIAD Method:

1. A spacecraft has two attitude sensors, sensing two unit vectors (directions), $\hat{\mathbf{v}}_i$ with $i = 1, 2$. We know the first sensor ($i = 1$) is more accurate than the second sensor. At an instant in time, the two vectors measured by the sensors have the body frame components:

$$\hat{\mathbf{v}}_1^B = \begin{pmatrix} 0.8273 \\ 0.5541 \\ -0.0920 \end{pmatrix} \quad \hat{\mathbf{v}}_2^B \begin{pmatrix} -0.8285 \\ 0.5522 \\ -0.0955 \end{pmatrix}$$

At the sma time, the four vectors are determined to have inertial frame components:

$$\hat{\mathbf{v}}_1^N = \begin{pmatrix} -0.1517 \\ -0.9669 \\ 0.2050 \end{pmatrix} \quad \hat{\mathbf{v}}_2^N \begin{pmatrix} -0.8393 \\ 0.4494 \\ -0.3044 \end{pmatrix}$$

Use the triad method to determine the estimated attitud $[\bar{B}N]$.

The T frame (Triad) is defined as follows:

$$t_1^B = \hat{\mathbf{v}}_1^B \quad t_1^N = \hat{\mathbf{v}}_1^N$$
$$t_2^B = \frac{t_1^B \times \hat{\mathbf{v}}_2^B}{|t_1^B \times \hat{\mathbf{v}}_2^B|} \quad t_1^N = \frac{t_1^N \times \hat{\mathbf{v}}_2^N}{|t_1^N \times \hat{\mathbf{v}}_2^N|}$$
$$t_3^B = t_1^B \times t_2^B \quad t_3^N = t_1^N \times t_2^N$$

where $\hat{\mathbf{v}}_i^B, \hat{\mathbf{v}}_i^N$ are first normalized.

[2]:
```
v_B1 = np.array([0.8273, 0.5541, -0.0920])
v_B1 = v_B1/(norm(v_B1,2))
v_B2 = np.array([-0.8285, 0.5522, -0.0955])
```

```
v_B2 = v_B2/(norm(v_B2,2))
v_N1 = np.array([-0.1517, -0.9669, 0.2050])
v_N1 = v_N1/(norm(v_N1,2))
v_N2 = np.array([-0.8393, 0.4494, -0.3044])
v_N2 = v_N2/(norm(v_N2,2))

## T frame in both references
t_B1 = v_B1
t_B2 = np.cross(t_B1, v_B2)
t_B2 = t_B2/norm(t_B2)
t_B3 = np.cross(t_B1, t_B2)
t_B3 = t_B3/norm(t_B3)

BT = np.hstack((t_B1.reshape(-1,1), t_B2.reshape(-1,1), t_B3.reshape(-1,1)))

t_N1 = v_N1
t_N2 = np.cross(t_N1, v_N2)
t_N2 = t_N2/norm(t_N2)
t_N3 = np.cross(t_N1, t_N2)
t_N3 = t_N3/norm(t_N3)

NT = np.hstack((t_N1.reshape(-1,1), t_N2.reshape(-1,1), t_N3.reshape(-1,1)))

BN = BT @ (NT.T)

print('The estimated attitude [BN] is : \n', BN)
```

```
The estimated attitude [BN] is :
 [[ 0.41555875 -0.85509088  0.31004921]
 [-0.83393237 -0.49427603 -0.24545471]
 [ 0.36313597 -0.15655922 -0.91848869]]
```

BN can also be computed by mean of the following expression: $BN = t_{N1}t_{B1}^T + (t_{N1} \times t_{N2})(t_B1 \times t_{B2})^T + t_{N2}t_{B2}^T$

Extracted from the the book: 'Fundamentals of Spacecraft Attitude Determination and Control' by Landis and Crassidis

```
[3]: A_triad = np.outer(t_B1, t_N1) + np.outer(t_B3, t_N3) + np.outer(t_B2, t_N2)
     print(A_triad)
```

```
[[ 0.41555875 -0.85509088  0.31004921]
 [-0.83393237 -0.49427603 -0.24545471]
 [ 0.36313597 -0.15655922 -0.91848869]]
```

2. Assume the estimated attitude is given by

$$\bar{B}N = \begin{bmatrix} 0.969846 & 0.17101 & 0.173648 \\ -0.200706 & 0.96461 & 0.17101 \\ -0.138258 & -0.200706 & 0.969846 \end{bmatrix}$$

and the true attitude is given by

$$BN = \begin{bmatrix} 0.963592 & 0.187303 & 0.190809 \\ -0.223042 & 0.956645 & 0.187303 \\ -0.147454 & -0.223042 & 0.963592 \end{bmatrix}$$

Express the estimation error in terms of a principal rotation angle in units of degrees.

[4]:
```
BT_est = np.array([[0.969846, 0.17101, 0.173648],[-0.200706, 0.96461, 0.
 ↪17101],[-0.138258, -0.200706, 0.969846
]])

BT_true = np.array([[0.963592, 0.187303, 0.190809],[-0.223042, 0.956645, 0.
 ↪187303],[-0.147454, -0.223042, 0.963592]])

Phi_est, _  = DCM_to_PR(BT_est)
Phi_true, _ = DCM_to_PR(BT_true)
error = abs(Phi_est - Phi_true)


print('The estimation error in terms of a principal rotation angle is :\n',␣
 ↪error, ' degrees')
```

```
The estimation error in terms of a principal rotation angle is :
 1.8284522666836587  degrees
```

2 Wahba's Problem.

We can improve on the TRIAD method in two ways: by allowing arbitrary weighting of the measurements and by allowing the use of more than two measurements. The latter is especially important for use with star trackers that can track many stars simultaneously. Wahba's problem is to find the orthogonal matrix A with determinant $+1$ that minimizes the loss function

$$L(A) = \frac{1}{2}\sum_{i=1}^{N} w_i||\mathbf{b}_i - A\mathbf{r}_i||^2$$

where $\{\mathbf{b}_i,\ \ i = 1, ..., N\}$ is a set of N unit vectors measured in a spacecraft's body frame, $\{\mathbf{r}_i\ \ i = 1, ..., N\}$ are the corresponding unit vectors in a reference frame, and $\{w_i\}$ are the corresponding non negative weights which for now we will set equal to 1. Since

$$||\mathbf{b}_i - A\mathbf{r}_i||^2 = ||\mathbf{b}_i||^2 + 2||A\mathbf{r}_i||^2 - 2\langle \mathbf{b}_i, A\mathbf{r}_i\rangle = 2 - 2tr(A\mathbf{r}_i\mathbf{b}_i^T)$$

then the loss function can be written in a very convenient form:

$$L(A) = \lambda_0 - tr(AB^T)$$

where $\lambda_0 = \sum_{i=1}^{N} w_i$ and the "attitude profile matrix" B is defined by the following sum of rank 1 matrices:

$$B = \sum_{i=1}^{N} w_i\mathbf{b}_i\mathbf{r}_i^T$$

Now it is clear the loss function is minimized when $tr(AB^T)$ is maximized.

Algorithms for solving Wahba's problem fall into two classes. The first solves for the attitude matrix directly, and the second solves for the quaternion representation of the attitude matrix. With error-free mathematics, all algorithms should lead to the same attitude, and testing shows this to be the case. Quaternion solutions have proven to be much more useful in practice, so we will consider them first.

2.1 Quaternion Solutions of Wahba's Problem

2.1.1 Davenport's q Method:

First we write the loss function in terms of the quaternions (Euler parameters) $\mathbf{q} = (\beta_0, \beta_1, \beta_2, \beta 3)$: (See Section 5.3 o the book)

$$L(A(\mathbf{q}) = \lambda_0 - \mathbf{q}^T K(B)\mathbf{q}$$

with

$$K(B) = \begin{bmatrix} tr(B) & Z^T \\ Z & B + B^T - tr(B) I_{3\times 3} \end{bmatrix} \quad Z = \sum_{i=1}^{n} w_i(\mathbf{b}_i \times \mathbf{r}_i) = \begin{bmatrix} B_{23} - B_{32} \\ B_{31} - B_{13} \\ B_{12} - B_{21} \end{bmatrix}$$

Note that the expression for the matrix $K(B)$ is different to the one from the book, this is because for us the first component f the quaternion is the scalar par β_0 as opossed to the last coordinate in the book.

Since we look for the matrix $\tilde{\mathbf{q}} = \min_{\mathbf{q}} L(A(\mathbf{q})) = \max_{\mathbf{q}} \mathbf{q}^T K(B)\mathbf{q}$ subject to $\mathbf{q}^T\mathbf{q} = 1$, by the Lagrange multipliers method, we optimize the function:

$$J(\mathbf{q}) = \mathbf{q}^T K(B)\mathbf{q} - \lambda \mathbf{q}^T\mathbf{q}$$

with λ the multiplier. The solution of this optimization problem is, by taking derivatives with respect to $\mathbf{q}$, are the quaternions $\mathbf{q}$ such that:

$$K(B)\mathbf{q} = \lambda\mathbf{q}$$

i.e , the eigenvector of $K(B)$, with eigenvalue λ. But, which eigenvector? Since we want to maximize $\mathbf{q}^T K(B)\mathbf{q} = \mathbf{q}^T\lambda\mathbf{q} = \lambda$, then que wished quaternion must be the one corresponding to the maximum eigenvalue, if a short rotation quaternion is needed, its first coordinate must be possitive and also it has to be of lenght 1.

Davenport's algorithm does not have a unique solution if the two largest eigenvalues of $K(B)$ are equal. This is not a failure of the q method; it means that the data are not sufficient to determine the attitude uniquely. Very robust algorithms exist to solve the symmetric eigenvalue problem, and Davenport's method remains the best method for solving Wahba's problem if one has access to one of these eigenvalue decomposition algorithms.

Example:

```
[5]: theta_true = (30., 20., -10.) #in degrees
     euler = (3,2,1)
     BN_true = DCMatrix(theta_true, euler)
```

```
[6]: BN_true
```

[6]:
```
array([[ 0.81379768,  0.46984631, -0.34202014],
       [-0.54383814,  0.82317294, -0.16317591],
       [ 0.20487413,  0.31879578,  0.92541658]])
```

[7]:
```
r1 = np.array([1,0,0])
r2 = np.array([0,0,1]) # measurements in the reference frame coordinates
b1 = BN_true @r1
b2 = BN_true@r2
```

[8]:
```
print(b1,b2)
```

```
[ 0.81379768 -0.54383814  0.20487413] [-0.34202014 -0.16317591  0.92541658]
```

[9]:
```
np.random.seed(0)
b1_noise = np.array([0.8190, -0.52820, 0.22420])
b2_noise = np.array([-0.31380,-0.15840, 0.93620])
#b1_noise = b1 + 0.01*np.random.rand(3)
#b2_noise = b2 + 0.01*np.random.rand(3)
```

[10]:
```
print(b1_noise, b2_noise)
```

```
[ 0.819  -0.5282  0.2242] [-0.3138 -0.1584  0.9362]
```

[11]:
```
## The weights:
w1, w2 = 1,1
```

[12]:
```
# The matrix K(B):

def KB(B, Z) :
    KB =  np.vstack((np.hstack((np.trace(B), Z)), np.hstack((Z.reshape(-1,1), B␣
 ↪+ B.T - np.trace(B)*np.eye(3)))))
    return KB
```

[13]:
```
B = w1*np.outer(b1_noise,r1) + w2*np.outer(b2_noise, r2)
Z = w1*np.cross(b1_noise, r1) + w2*np.cross(b2_noise, r2)
K = KB(B,Z)
```

[14]:
```
print(K)
```

```
[[ 1.7552 -0.1584  0.538   0.5282]
 [-0.1584 -0.1172 -0.5282 -0.0896]
 [ 0.538  -0.5282 -1.7552 -0.1584]
 [ 0.5282 -0.0896 -0.1584  0.1172]]
```

[15]:
```
lamb, v = eig(K)
```

[16]:
```
print(lamb, v)
```

```
[ 1.99969236+0.j -1.99969236+0.j -0.03656634+0.j  0.03656634+0.j] [[ 0.94806851
-0.14137121  0.26650527 -0.10077314]
 [-0.11720729  0.25969739  0.7703875   0.57037077]
 [ 0.14137121  0.94806851 -0.10077314 -0.26650527]
 [ 0.25969739  0.11720729 -0.57037077  0.7703875 ]]
```

Since the first eigenvalue is the biggest then we set the quaternion set to be the first vector in v the matrix of the eigenvectors of KB

[17]:
```
q = v[:,0]
```

[18]:
```
print('The short rotation quaternion of theses measurements is q = \n', q)
```

```
The short rotation quaternion of theses measurements is q =
 [ 0.94806851 -0.11720729  0.14137121  0.25969739]
```

Now the corresponding DCM matrix A is:

[19]:
```
A = EP_to_DCM(q)
print('The estimated BN is A = ', A)
print('\n the real DCM is \n BN_true = ', BN_true)
```

```
The estimated BN is A =  [[ 0.82514289  0.45928237 -0.32893604]
 [-0.52556131  0.83763943 -0.14881361]
 [ 0.20718233  0.29566855  0.93255327]]

 the real DCM is
 BN_true =  [[ 0.81379768  0.46984631 -0.34202014]
 [-0.54383814  0.82317294 -0.16317591]
 [ 0.20487413  0.31879578  0.92541658]]
```

[20]:
```
print('because of the noise in the measurements the matrix A.T@BN_true is not␣
↪exactly the identity but close. It is : \n', A.T@BN_true   )
```

```
because of the noise in the measurements the matrix A.T@BN_true is not exactly
the identity but close. It is :
 [[ 0.99976596  0.02111134 -0.00472658]
 [-0.02120251  0.99957213 -0.02014982]
 [ 0.00429917  0.02024532  0.9997858 ]]
```

[21]:
```
q_est = DCM_to_PR(A.T@BN_true)
print('So the error in the estimation in terms of the PR angel is Phi = ',␣
↪q_est[0], 'degrees')
```

```
So the error in the estimation in terms of the PR angle is Phi =
1.695973840742523 degrees
```

If we us the TRIAD method instead, we obtain:

[22]:
```
A_triad = TRIAD(b1_noise, b2_noise, r1, r2)
q_triad =  DCM_to_PR(A_triad.T @ BN_true)
print('The matrix BbarB, i.e the DCM B relative to its estimated bar B is also␣
 ↪almost the identity because of noisy measurements : \n',  A_triad.T @ BN_true)
print('The error in heestimation in terms of the PR angle is Phi_triad = ',␣
 ↪q_triad[0])
```

```
The matrix BbarB, i.e the DCM B relative to its estimated bar B is also almost
the identity because of noisy measurements :
 [[ 0.99967745  0.02147796  0.01355327]
 [-0.02120251  0.99957213 -0.02014982]
 [-0.01398025  0.01985596  0.9997051 ]]
The error in heestimation in terms of the PR angle is Phi_triad =
1.8525322520674505
```

2.1.2 Concept Check 3, 4 - Devenport's q-Method

6. A spacecraft has two attitude sensors, sensing two unit vectors $\hat{v}_i$, $i = 1, 2$. We know the first sensor is more accurate than the second sensor. At an instant in time, the two vectors measured by the sensors have the body frame components

$$\hat{v}_1^B = \begin{pmatrix} 0.8273 \\ 0.5541 \\ -0.0920 \end{pmatrix} \quad \hat{v}_2^B = \begin{pmatrix} -0.8285 \\ 0.5522 \\ -0.0955 \end{pmatrix}$$

At the same time, the two vectors are determined to have inertial frame components:

$$\hat{v}_1^N = \begin{pmatrix} -0.1517 \\ -0.9669 \\ 0.2050 \end{pmatrix} \quad \hat{v}_2^N = \begin{pmatrix} -0.8393 \\ 0.4494 \\ -0.3044 \end{pmatrix}$$

Use Davenport's q-Method to determine the estimated attitude $\bar{B}N$

[23]:
```
b1 = np.array([0.8273, 0.5541, -0.0920])
b2 = np.array([-0.8285, 0.5522, -0.0955])
r1 = np.array([-0.1517, -0.9669, 0.2050])
r2 = np.array([-0.8393, 0.4494, -0.3044])
b = (b1, b2)
r = (r1, r2)
w = (1,0.8)
barBN = davenport(b, r, w)
print('The davenport function prints the quaternion!')
```

```
[ 0.02640807 -0.84098146  0.50200028 -0.20012127]
The davenport function prints the quaternion!
```

[24]:
```
print(barBN)
```

```
[[ 0.41589439 -0.85491549  0.31008284]
 [-0.83377622 -0.49459666 -0.24533927]
 [ 0.36311028 -0.15650447 -0.91850818]]
```

Using QUEST method we get the same result but in a more efficient way!

```
[25]: q_quest, BarBN_quest = QUEST(b, r, w)
```

```
[26]: print('The BarBN matrix using the QUEST method is: \n', BarBN_quest)
```

```
The BarBN matrix using the QUEST method is:
 [[ 0.41589439 -0.85491549  0.31008284]
 [-0.83377622 -0.49459666 -0.24533927]
 [ 0.36311028 -0.15650447 -0.91850818]]
```

```
[27]: print(q_quest)
```

```
[ 0.02640807 -0.84098146  0.50200028 -0.20012127]
```

```
[ ]:
```

References

[1] Landis and Crassidis. Fundamentals of Spacecraft Attitude Determination and Control.