Quaternions

> UTBM - P2023 - MV51 - TP3 - Julien CONSTANT

Ordinairement, les systèmes informatiques ne fournissent pas les outils relatifs à la théorie des quaternions, alors qu'ils le font par exemple pour les nombres complexes. L'objectif de ce TP est alors d'établir un certain nombre de résultats relatifs à cette théorie et de développer les modules indispensables afin d'en disposer dans l'environnement de travail lorsque le besoin s'en fera sentir dans le cadre de travaux en image.

1. Structure de corps de l'ensemble des quaternions

1. **On considère tout quaternion $q = t + xi + yj + zk$ comme un vecteur $h = (t, x, y, z)$. Écrire une fonction `affiche_h(h)` qui, à partir du vecteur h défini ci-dessus, renvoie la chaîne de l'écriture de q sous forme conventionnelle.**

```
function [] = affiche_h(h)
    fprintf('q = %d + %di + %dj + %dk\n', h(1), h(2), h(3), h(4));
end
```

Exemple :

```
>> affiche_h([1 2 3 4])

q = 1 + 2i + 3j + 4k
```

2. **Écrire une fonction `somme_h(h1, h2)` qui, à partir des deux vecteurs h_1 et h_2 définis ci-dessus, renvoie le quaternion somme.**

```
function [h] = somme_h(h1, h2)
    h = h1 + h2;
end
```

Exemple :

```
>> somme_h([1 2 3 4], [5 6 7 8])

ans =
    6     8    10    12
```

3. **Écrire une fonction `produit_h(h1, h2)` qui, à partir des deux vecteurs h_1 et h_2 définis ci-dessus, renvoie le quaternion produit.**

```
function [h] = produit_h(h1, h2)
    v1 = h1(2:4);
    v2 = h2(2:4);
    v = h1(1) * v2 + h2(1) * v1 + cross(v1, v2);
    h(1) = h1(1) * h2(1) - dot(v1, v2);
    h(2:4) = v;
end
```

Exemple :

```
>> produit_h([1 2 3 4], [5 6 7 8])

ans =
   -60    12    30    24
```

4. **Écrire les fonctions `conjugue_h(h)`, `module_h(h)` et `inverse_h(h)` qui, à partir d'un quaternion convenable h renvoient le conjugué, le module et l'inverse s'ils existent...**

```
function [H] = conjugue_h(h)
    H = [h(1), - h(2:4)];
end

function [H] = norme_h(h)
    H = sqrt(h(1)^2 + h(2)^2 + h(3)^2 + h(4)^2);
end

function [H] = inverse_h(h)
    if ~isequal(h, [0, 0, 0, 0])
        H = conjugue_h(h) / norme_h(h)^2;
    else
        disp('Erreur : division par 0');
    end
end
```

Exemple :

```
>> conjugue_h([1 2 3 4])
ans =
    1    -2    -3    -4

>> norme_h([1 2 3 4])
ans =
    5.4772

>> inverse_h([1 2 3 4])
ans =
    0.0333   -0.0667   -0.1000   -0.1333

>> inverse_h([0 0 0 0])
Erreur : division par 0
```

2. Forme polaire

1. **Écrire une fonction `polaire_h(h)` qui, pour un quaternion convenable q représenté par h, renvoie sa forme polaire $[r, \theta]$ définie par :**

$$q = re^{I\theta}$$

```
function p = polaire_h(h)
    r = norme_h(h);
    theta = abs(acos((h(1) / r)));
    vect = h(2:4);
    i = vect / (r * sin(theta));
    p = [r, theta, i];
end
```

Exemple :

```
>> polaire_h([1 2 3 4])
ans =

    5.4772    1.3872    0.3714    0.5571    0.7428
```

2. **Écrire la fonction inverse `algebr_h(h)`.**

```
function [H] = algebr_h(r, theta, i)
    h = [0, 0, 0, 0];
    h(1) = r * cos(theta);
    h(2:4) = r * i * sin(theta);
    H = h;
end
```

Exemple :

```
>> polaire = polaire_h([1 2 3 4])
ans =
    5.4772    1.3872    0.3714    0.5571    0.7428

>> algebr_h(polaire(1), polaire(2), polaire(3:5))
ans =
    1.0000    2.0000    3.0000    4.0000
```

> On retrouve bien le quaternion de départ.

3. Exemple d'utilisation des outils développés : vérification de l'associativité du produit

Proposer une preuve informatique de l'associativité du produit de quaternions.

Les quaternions s'écrivant dans une base $1, i, j, k$ on peut prouver que cette base respecte l'associativité du produit en testant l'équation suivante :

$$(uv)w = u(vw)$$

avec u, v, w appartenant à la base : $(1, i, j, k) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

```
function [res] = proof()
    M = [
        [1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ];

    res = true;

    for u = 1:1:3
        for v = 1:1:3
            for w = 1:1:3
                % si produit_h(produit_h(u, v), w) ≠ produit_h(u, produit_h(v, w))
                if (produit_h(produit_h(M(u * 4 + 1 : u * 4 + 4), M(v * 4 : v * 4
+ 3)), M(w * 4 + 1 : w * 4 + 4)) ~= produit_h(M(u * 4 + 1 : u * 4 + 4),
produit_h(M(v * 4 + 1 : v * 4 + 4), M(w * 4 + 1 : w * 4 + 4))))
                    res = false;
                end
            end
        end
    end

end
```

Exemple :

```
>> proof()
ans =
  logical
   1
```

> La fonction `proof()` renvoie `true` si l'associativité est vérifiée, `false` sinon. Ici, elle renvoie `true` donc l'associativité a bien été vérifiée informatiquement.

4. Quaternions et rotations

1. **Étant donnés un vecteur non nul $\vec{n}$ et un réel θ d'un intervalle convenable, écrire une fonction `matrice_rotation(n, theta)` qui renvoie la matrice de rotation vectorielle $l_{\vec{n},\theta}$ obtenue en utilisant les résultats du cours.**

```
function [I] = matrice_rotation(n, theta)
    I = [
        n(1)^2 * (1 - cos(theta)) + cos(theta),
        n(1) * n(2) * (1 - cos(theta)) - n(2) * sin(theta),
        n(1) * n(3) * (1 - cos(theta)) + n(2) * sin(theta);

        n(1) * n(2) * (1 - cos(theta)) + n(1) * sin(theta),
        n(2)^2 * (1 - cos(theta)) + cos(theta),
        n(2) * n(3) * (1 - cos(theta)) - n(1) * sin(theta);

        n(1) * n(3) * (1 - cos(theta)) - n(2) * sin(theta),
        n(2) * n(3) * (1 - cos(theta)) + n(1) * sin(theta),
        n(3)^2 * (1 - cos(theta)) + cos(theta)
    ];
end
```

Exemple :

```
>> matrice_rotation([1 2 3], pi/6)
ans =
    1.0000   -0.7321    1.4019
    0.7679    1.4019    0.3038
   -0.5981    1.3038    2.0718
```

2. **Écrire une fonction `h_rot(h)` qui, à partir du quaternion unitaire q représenté par h, fournit la matrice de rotation vectorielle associée.**

```
function [I] = h_rot(q)
    s = q(1);
    x = q(2);
    y = q(3);
    z = q(4);

    r00 = 1 - 2*y^2 - 2*z^2;
    r01 = 2*x * y - 2*s * z;
    r02 = 2*x * z + 2*s * y;
    r10 = 2*x * y + 2*s * z;
    r11 = 1 - 2*x^2 - 2*z^2;
    r12 = 2*y * z - 2*s * x;
    r20 = 2*x * z - 2*s * y;
    r21 = 2*y * z + 2*s * x;
    r22 = 1 - 2*x^2 - 2*y^2;

    I = [r00 r01 r02; r10 r11 r12; r20 r21 r22];
end
```

Exemple :

```
>> h_rot([1, 2, 3, 4])
ans =
   -49     4    22
    20   -39    20
    10    28   -25
```

3. **Inversement, écrire une fonction `rot_h(L)` qui, à partir d'une matrice de rotation vectorielle L de $\mathcal{V}_3$, détermine le quaternion unitaire associé.**

```
function [q] = rot_h(L)
    r00 = L(1, 1);
    r01 = L(1, 2);
    r02 = L(1, 3);
    r10 = L(2, 1);
    r11 = L(2, 2);
    r12 = L(2, 3);
    r20 = L(3, 1);
    r21 = L(3, 2);
    r22 = L(3, 3);

    T = r00 + r11 + r22;
    Tmax = max([r00, r11, r22]);

    if (T > 0)
        s = sqrt(T + 1) / 2;
        x = (r21 - r12) / (4 * s);
        y = (r02 - r20) / (4 * s);
        z = (r10 - r01) / (4 * s);

    elseif (Tmax == r00)
        x = sqrt(r00 - r11 - r22 + 1) / 2;
        s = -(r12 - r21) / (4 * x);
        y = (r01 + r10) / (4 * x);
        z = (r02 + r20) / (4 * x);

    elseif (Tmax == r11)
        y = sqrt(r11 - r00 - r22 + 1) / 2;
        s = -(r20 - r02) / (4 * y);
        x = (r01 + r10) / (4 * y);
        z = (r12 + r21) / (4 * y);

    elseif (Tmax == r22)
        z = sqrt(r22 - r00 - r11 + 1) / 2;
        s = -(r01 - r10) / (4 * z);
        x = (r02 + r20) / (4 * z);
        y = (r12 + r21) / (4 * z);
    end

    q = [s x y z];
end
```

Exemple :

```
>> hrot = h_rot([1, 2, 3, 4])
ans =
   -49     4    22
    20   -39    20
    10    28   -25

>> rot_h(hrot)
ans =
    1     2     3     4
```

> On retrouve bien le quaternion de départ.