Solving simultaneous linear equations with quaternions

by Gonzalo Reynaga García
October 2016

This article will describe be procedure to solve simultaneous linear equations with quaternions, it will explain the 3 variables and 4 variables equations, converting or displaying each one as a vector. In the process, it will redefined some concepts like vectors for 3 and 4 variables. Also will show some numeric results of this vector expansion for 5 and 6 variables linear equations.

Introduction

The motivation to use quaternions to solve linear equations was due the associative property and try to convert one equation into a group of quaternions thinking in parallel programming to solve a big equation systems. At the end, the simplicity of the implementation to solve equations in a form of vectors described in this article, I think it will be very useful.

First of all, the notation to display quaternions used in this article will be as the following, for a quaternion:

$$a_0 + a_1 \boldsymbol{i} + a_2 \boldsymbol{j} + a_3 \boldsymbol{k}$$

where a_0 a_1 a_2 and a_3 are scalars

it will represented as

$$[a_0 \quad a_1 \quad a_2 \quad a_3]$$

or

$$\begin{bmatrix} a_0 ; \\ a_1 ; \\ a_2 ; \\ a_3 \end{bmatrix}$$

The quaternion between two quaternions is defined as:

$$[a_0 \quad a_1 \quad a_2 \quad a_3] \wedge [b_0 \quad b_1 \quad b_2 \quad b_3] = \qquad \ldots(1)$$

$$\begin{bmatrix} a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3 ; \\ a_1b_0 + a_0b_1 - a_3b_2 + a_2b_3 ; \\ a_2b_0 + a_3b_1 + a_0b_2 - a_1b_3 ; \\ a_3b_0 - a_2b_1 + a_1b_2 + a_0b_3 \end{bmatrix}$$

3 simultaneous linear equations

For a 3 equation system:

$$a_0 \boldsymbol{x_0} + a_1 \boldsymbol{x_1} + a_2 \boldsymbol{x_2} + a_3 = 0$$
$$b_0 \boldsymbol{x_0} + b_1 \boldsymbol{x_1} + b_2 \boldsymbol{x_2} + b_3 = 0$$
$$c_0 \boldsymbol{x_0} + c_1 \boldsymbol{x_1} + c_2 \boldsymbol{x_2} + c_3 = 0$$

If the we define 3 quaternions as

$$a = [0 \; a_0 \; a_1 \; a_2]$$
$$b = [0 \; b_0 \; b_1 \; b_2]$$
$$c = [0 \; c_0 \; c_1 \; c_2]$$

The quaternion product q_a^q_b^q_c gives us the determinant of the matrix in the scalar part.
So, with that in mind, can be found the determinant of the solution of the equation? yes.
It was found the solution(times determinant value) will be:

$$a_c\text{^}b\text{^}c - a\text{^}b_c\text{^}c + a\text{^}b\text{^}c_c$$

where:

$$a_c = [a_3 \; 0 \; 0 \; 0]$$
$$b_c = [b_3 \; 0 \; 0 \; 0]$$
$$c_c = [c_3 \; 0 \; 0 \; 0]$$

The solution will be (a_c^b^c − a^b_c^c + a^b^c_c) divided by the determinant, the scalar part of (q_a^q_b^q_c)

Then those quaternions was into a vector to create that result and, to maintaining the associative property, it was set those quaternions pair into the quaternion product …(1) in which some products were zero explained below.

if we define a vector as

$$[Q_0 \; Q_1 \; Q_2 \; Q_3]$$

Where Q_0 , Q_1 , Q_2 and Q_3 are quaternions
the product of this vectors

$$[Q_0 \; Q_1 \; Q_2 \; Q_3] \text{ ^ } [P_0 \; P_1 \; P_2 \; P_3] =$$
$$\begin{aligned}&[Q_0\text{^}P_0 - 0 - Q_2\text{^}P_2 - 0 ;\\ &[Q_1\text{^}P_0 + Q_0\text{^}P_1 - Q_3\text{^}P_2 + Q_2\text{^}P_3;\\ &[Q_2\text{^}P_0 + 0 + Q_0\text{^}P_2 - 0;\\ &[Q_3\text{^}P_0 - Q_2\text{^}P_1 + Q_1\text{^}P_2 + Q_0\text{^}P_3 \;]\end{aligned} \quad \text{…(2)}$$

The linear equations can set into a vectors

$$A = [[0 \; 0 \; 0 \; 0] \; [0 \; 0 \; 0 \; 0] \; a \; a_c]$$
$$B = [[0 \; 0 \; 0 \; 0] \; [0 \; 0 \; 0 \; 0] \; b \; b_c]$$
$$C = [[0 \; 0 \; 0 \; 0] \; [0 \; 0 \; 0 \; 0] \; b \; c_c]$$

and the A^B^C will give us the solution.

Inverse vector

For the case of 3 variables, in the point of view of geometry, one linear equation can be considered as a plane, 2 equations a line, and 3 equations a point (like applying the intersection for each plane). But what if apply an inverse, is like applying an union?

The inverse vector of A= `[0 0 0 0] [0 0 0 0] a` a_c`]` can be obtained as:

$$A\wedge A^{-1} = [[1\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]]$$

$$A^{-1} = [[0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ \ a^{-1}\ \ a^{-1}\wedge a_c\wedge a^{-1}] \qquad \ldots(3)$$

where a^{-1} is the inverse of a quaternion.
Note that a_c is $[a_4\ 0\ 0\ 0]$

Example:
A Math-c script of the implementation of the product is:
Function imported from file eqproduct.mc

```
function qm1(ia,ib)
      result = [0;0;0;0];
      [a b c d] = ia;
      [e f g h] = ib;

      num = a(0);
      if(isArray(num) == 0)
            result(0)=   a^e  -   0   - c^g  -   0;
            result(1)=  b^e  +  a^f  - d^g  +  c^h;
            result(2)=  c^e  +   0   + a^g  -   0;
            result(3)=  d^e  -  c^f  + b^g  +  a^h;
            return result;
      end
      result(0)=  qm1(a,e)  -   0        - qm1(c,g)  -   0;
      result(1)=  qm1(b,e)  +  qm1(a,f)  - qm1(d,g)  +  qm1(c,h);
      result(2)=  qm1(c,e)  +   0        + qm1(a,g)  -   0;
      result(3)=  qm1(d,e)  -  qm1(c,f)  + qm1(b,g)  +  qm1(a,h);
      return result;
end
///////////////////////////////////////////
#import eqproduct.mc

//Creating a random 3x3 matrix
M=randi(20,3)-10;//+1i*randi(20,3)-10i;
//Creating a random 3x1 vector
Z=randi(20,3,1)-10;//+1i*randi(20,3,1)-10i;
//The product of matrix and vector
MZ=M*Z;

//Zero quaternion
zq = [0;0;0;0];

//Vector a
a=[0;M(0,0);M(0,1);M(0,2)];
ac=[-MZ(0); 0; 0; 0];

//Vector B
b=[0;M(1,0);M(1,1);M(1,2)];
bc=[-MZ(1); 0; 0; 0];

//Vector C
c=[0;M(2,0);M(2,1);M(2,2)];
cc=[-MZ(2); 0; 0; 0];


A=[zq; zq; a; ac];
B=[zq; zq; b; bc];
C=[zq; zq; c; cc];

      w1=qm1(A,B);
      result=qm1(w1,C);

      print("Vector result\n");

      result /result(2)(0)

      print("expected result\n");

      Z
```

Script output:

```
Vector result
ans = [[0 ;
      0 ;
      0 ;
      0] ;
      [0 ;
      0 ;
      0 ;
      0] ;
      [1 ;
      -0.587209 ;
      0.139535 ;
      0.555233] ;
      [5.9186 ;
      7 ;
      -5 ;
      -2]]
expected result
ans = [7 ;
      -5 ;
      -2]
```

And also gives the correct result if the linear equation are complex numbers, in the code above just uncomment (//+1i*randi(20,3)-10i;) to generate a random matrix with imaginary numbers.

Analysing the result

Changing the multiplication order

If we change the order of the same vectors in the output of the script shown above,C^B^A give us:

```
>>> de=result(2)(0); //determinant of the system
>>> w1=qm1(C,B);
>>> r2=qm1(w1,A);
>>> r2/de
ans = [[0 ;
      0 ;
      0 ;
      0] ;
      [0 ;
      0 ;
      0 ;
      0] ;
      [-1 ;
      -0.587209 ;
      0.139535 ;
      0.555233] ;
      [5.9186 ;
      -7 ;
      5 ;
      2]]
```

change the sign of some elements so if we:

>>> r1=result;
>>> ((r1+r2)/2)/de
ans = [[0 ;

```
     0 ;
     0 ;
     0] ;
     [0 ;
     0 ;
     0 ;
     0] ;
     [0 ;
     -0.587209 ;
     0.139535 ;
     0.555233] ;
     [5.9186 ;
     0 ;
     0 ;
     0]]
>>> ((r1-r2)/2)/de
ans = [[0 ;
     0 ;
     0 ;
     0] ;
     [0 ;
     0 ;
     0 ;
     0] ;
     [1 ;
     0 ;
     0 ;
     0] ;
     [0 ;
     7 ;
     -5 ;
     -2]]
```

The solution of the equation is expected to be the same when changing the order of the equations, it seems it has two parts just like inner and outer product referred in [ref.1].

Symbolic output

A simulating symbolic output script was coded to see the process occurs in the multiplication of vectors.

The program and output for the case of 3 variable is shown below.
function imported from file eqproduct.mc

```
function qm1_simb(sign,ia,ib)
      result = ["";"";"";""];
      [a b c d] = ia;
      [e f g h] = ib;

      num = a;
      if(isString(num))
            result(0)=   str_simb("+",a,e)               + str_simb("-",c,g)  ;
            result(1)+=  str_simb("+",b,e)  +  str_simb("+",a,f)  +
str_simb("-",d,g)  +  str_simb("+",c,h);
            result(2)+=  str_simb("+",c,e)               + str_simb("+",a,g);
            result(3)+=  str_simb("+",d,e)  +  str_simb("-",c,f)  +
str_simb("+",b,g)  +  str_simb("+",a,h);
            return result;
      end
      result(0)=   qm1_simb("+",a,e)                  + qm1_simb("-",c,g)  ;
      result(1)+=  qm1_simb("+",b,e)  +  qm1_simb("+",a,f)  + qm1_simb("-",d,g)
+  qm1_simb("+",c,h);
      result(2)+=  qm1_simb("+",c,e)               + qm1_simb("+",a,g)  ;
      result(3)+=  qm1_simb("+",d,e)  +  qm1_simb("-",c,f)  + qm1_simb("+",b,g)
+  qm1_simb("+",a,h);
      return result;
end

function str_simb(sign,ia,ib)
      if((strlen(ia)==0)||(strlen(ib)==0))
            return "";
      end
      return sign+"("+ia+"^"+ib+")";
end

///////////////////////////////////////////
#import eqproduct.mc

      A=[""; ""; "A"; "Ac"];
      B=[""; ""; "B"; "Bc"];
      C=[""; ""; "C"; "Cc"];

      w1= qm1_simb("+",A,B);  //is"+" because we want +A^B
      result = qm1_simb("+",w1,C); //is "+" for +w1^B

      print("Vector result\n");

      result
```

The output of the script program above is:

```
Vector result
ans = [ ;
        ;
       +(-(A^B)^C) ;
       +(-(Ac^B)+(A^Bc)^C)+(-(A^B)^Cc)]
```

4 simultaneous linear equations

For a 4 equation system:

$$a_0\, \boldsymbol{x_0} + a_1\, \boldsymbol{x_1} + a_2\, \boldsymbol{x_2} + a_3\, \boldsymbol{x_2} + a_4 = 0$$
$$b_0\, \boldsymbol{x_0} + b_1\, \boldsymbol{x_1} + b_2\, \boldsymbol{x_2} + b_3\, \boldsymbol{x_2} + b_4 = 0$$
$$c_0\, \boldsymbol{x_0} + c_1\, \boldsymbol{x_1} + c_2\, \boldsymbol{x_2} + c_3\, \boldsymbol{x_2} + c_4 = 0$$
$$d_0\, \boldsymbol{x_0} + d_1\, \boldsymbol{x_1} + d_2\, \boldsymbol{x_2} + d_3\, \boldsymbol{x_2} + d_4 = 0$$

Just like the 3 variables case where we found the determinant with multiplications of quaternions a^b^c; for 4 variables case we found a vector structure where A^B^C^D is the determinant of the equation system.
the solution of a 3 variables A^B^C are the determinants necessary to complete the 4x4 determinant, just multiplying by D.
as the we defined A in 3 variables section

$$A = [[0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ a\ a_e]$$

so the solution of 4x4 system was found checking the relation of a 3x3 solution and was verified numerically.
$(A_c\text{^}B\text{^}C\text{^}D - A\text{^}B_c\text{^}C\text{^}D + A\text{^}B\text{^}C_c\text{^}D - A\text{^}B\text{^}C\text{^}D_c)$

where:

$$A_c = [[a_4\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]]$$

The determinant of (A^B^C^D) is found in index (1)(0);

Then was experimented with the expansion adding scalar in the same way was done in 3 variables. Probably there is a more efficient way to do it.
The redefinition of vector A and comparing with 3 variables vector

let
$Z_{1q} = [0\ 0\ 0\ 0]$
$Z_{2q} = [[0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]]$

to simplify

A in 3x3

$$A = [Z_{1q}\ Z_{1q}\ a\ a_e]$$

A in 4x4

$$A = [Z_{2q}\ Z_{2q}\ [Z_{1q}\ Z_{1q}\ a\ a_e]\ [a_c\ Z_{1q}\ Z_{1q}\ Z_{1q}]]$$

where

$a = [0\ a_0\ a_1\ a_2]$
$a_e = [a_3\ 0\ 0\ 0]$
$a_c = [a_4\ 0\ 0\ 0]$

Inverse vector

For the case of 4 variables,

The inverse vector of A= $[Z_{2q}\ Z_{2q}\ [Z_{1q}\ Z_{1q}\ a\ a_e]\ [a_c\ Z_{1q}\ Z_{1q}\ Z_{1q}]]$ can be obtained as:

$$A\wedge A^{-1} = [\ [[1\ 0\ 0\ 0]\ Z_{1q}\ Z_{1q}\ Z_{1q}]\ Z_{2q}\ Z_{2q}\ Z_{2q}]$$

$$A^{-1} = [[0\ 0\ 0\ 0]\ [0\ 0\ 0\ 0]\ \ A'^{-1}\ \ (A'^{-1})\wedge(A'_c)\wedge(A'^{-1})] \qquad \ldots(4)$$

where

$(A') = [Z_{1q}\ Z_{1q}\ a\ a_e]$
$(A'_c) = [a_c\ Z_{1q}\ Z_{1q}\ Z_{1q}]$
(A'^{-1}) is the inverse defined in …(3)

Script example for 4x4:

```
#import eqproduct.mc

//Creating a random 4x4 matrix
M=randi(20,4)-10;//+1i*randi(20,3)-10i;
//Creating a random 3x1 vector
Z=randi(20,4,1)-10;//+1i*randi(20,3,1)-10i;
//The product of matrix and vector
MZ=M*Z;

//Zero quaternion
z1q = [0;0;0;0];
z2q = [[0;0;0;0] [0;0;0;0] [0;0;0;0] [0;0;0;0]];

//Vector A
a=[0;M(0,0);M(0,1);M(0,2)];
ae=[M(0,3); 0; 0; 0];
ac=[-MZ(0); 0; 0; 0];

//Vector B
b=[0;M(1,0);M(1,1);M(1,2)];
be=[M(1,3); 0; 0; 0];
bc=[-MZ(1); 0; 0; 0];

//Vector C
c=[0;M(2,0);M(2,1);M(2,2)];
ce=[M(2,3); 0; 0; 0];
cc=[-MZ(2); 0; 0; 0];

//Vector D
d=[0;M(3,0);M(3,1);M(3,2)];
de=[M(3,3); 0; 0; 0];
dc=[-MZ(3); 0; 0; 0];

A=[z2q z2q [z1q; z1q; a; ae] [ac; z1q; z1q; z1q]];
B=[z2q z2q [z1q; z1q; b; be] [bc; z1q; z1q; z1q]];
C=[z2q z2q [z1q; z1q; c; ce] [cc; z1q; z1q; z1q]];
D=[z2q z2q [z1q; z1q; d; de] [dc; z1q; z1q; z1q]];

    w1=qm1(A,B);
    w1=qm1(w1,C);
    result=qm1(w1,D);
    print("Vector result\n");
    result /result(0)(1)(0)
    print("expected result\n");

    Z
```

Analysing the result

Just like the 3x3 case, in 4x4 it seems has 2 parts, inner and outer product when inverting the order of the vector product. Also gives the correct result if using complex numbers.

Symbolic output

Running the following program, displays the operation executed in the multiplication.

```
///////////////////////////////////////////
#import eqproduct.mc

      z1q="";
      z2q=[""; ""; ""; ""]

      A=[z2q z2q [z1q; z1q; "A"; "Ae"] ["Ac"; z1q; z1q; z1q]];
      B=[z2q z2q [z1q; z1q; "B"; "Be"] ["Bc"; z1q; z1q; z1q]];
      C=[z2q z2q [z1q; z1q; "C"; "Ce"] ["Cc"; z1q; z1q; z1q]];
      D=[z2q z2q [z1q; z1q; "D"; "De"] ["Dc"; z1q; z1q; z1q]];


      print("Vector A^B result\n");
      w1= qm1_simb("+",A,B)  //is "+" because we want +A^B



      w1 = qm1_simb("+",w1,C); //is "+" for +w1^C
      result = qm1_simb("+",w1,D); //is "+" for +w1^D

      result
```

Output

```
Vector A^B result
w1 = [[-(A^B) ;
      -(Ae^B)+(A^Be) ;
       ;
      ] ;
      [ ;
       ;
      +(Ac^B)+(A^Bc) ;
      +(Ac^Be)+(Ae^Bc)] ;
      [ ;
       ;
       ;
      ] ;
      [ ;
       ;
       ;
      ]]
Vector A^B^C^D result
ans = [[-(+(-(A^B)^C)^D) ;
      -(+(-(Ae^B)+(A^Be)^C)+(-(A^B)^Ce)^D)+(+(-(A^B)^C)^De) ;
       ;
      ] ;
      [ ;
       ;
      +(-(+(Ac^B)+(A^Bc)^C)+(-(A^B)^Cc)^D)+(+(-(A^B)^C)^Dc) ;
      +(-(+(Ac^Be)+(Ae^Bc)^C)+(+(Ac^B)+(A^Bc)^Ce)+(-(Ae^B)+(A^Be)^Cc)^D)+(-(+
(Ac^B)+(A^Bc)^C)+(-(A^B)^Cc)^De)+(+(-(Ae^B)+(A^Be)^C)+(-(A^B)^Ce)^Dc)] ;
      [ ;
       ;
       ;
      ] ;
      [ ;
       ;
       ;
      ]]
```

N simultaneous linear equations

5x5 equations
Then trying to repeat the expansion for 5x5 linear system, just was like in 4x4.
the vector used for 5x5

let
Z_{1q} = [0 0 0 0]
Z_{2q} = [Z_{1q} Z_{1q} Z_{1q} Z_{1q}]
Z_{3q} = [Z_{2q} Z_{2q} Z_{2q} Z_{2q}]

A= [Z_{3q}
Z_{3q}
[Z_{2q}; Z_{2q}; [Z_{1q}; Z_{1q}; a; a_{e1}]; [a_{e2}; Z_{1q}; Z_{1q}; Z_{1q}]]
[[a_c; Z_{1q}; Z_{1q}; Z_{1q}]; Z_{2q}; Z_{2q}; Z_{2q}]];

and executing A^B^C^D^E gives the negative of the solution, but if we use the vector

A= [Z_{3q}
Z_{3q}
[Z_{2q}; Z_{2q}; [Z_{1q}; Z_{1q}; a; a_{e1}]; [a_{e2}; Z_{1q}; Z_{1q}; Z_{1q}]]
[[$-a_c$; Z_{1q}; Z_{1q}; Z_{1q}]; Z_{2q}; Z_{2q}; Z_{2q}]];

gives the correct solution, note that the only difference is I change the sign a_c

6x6 equations
Then repeating the expansion for 6x6 linear system,
the vector used for 6x6

let
Z_{1q} = [0; 0; 0; 0]
Z_{2q} = [Z_{1q}; Z_{1q}; Z_{1q}; Z_{1q}]
Z_{3q} = [Z_{2q}; Z_{2q}; Z_{2q}; Z_{2q}]
Z_{4q} = [Z_{3q}; Z_{3q}; Z_{3q}; Z_{3q}]

A = [Z_{4q}

Z_{4q}

[Z_{3q}; Z_{3q};[Z_{2q}; Z_{2q}; [Z_{1q}; Z_{1q}; a; a_{e1}]; [a_{e2}; Z_{1q}; Z_{1q}; Z_{1q}]]; [[$-a_{e3}$; Z_{1q}; Z_{1q}; Z_{1q}]; Z_{2q}; Z_{2q}; Z_{2q}]]

[[[$-b_c$; Z_{1q}; Z_{1q}; Z_{1q}]; Z_{2q}; Z_{2q}; Z_{2q}]; Z_{3q}; Z_{3q}; Z_{3q}]];

and executing A^B^C^D^E^F gives the correct solution.

Conclusion

I suppose we can expand the number of variables in the same way to N, but I think this is not a efficient or correct way to do it, we can notice a “waste” of space in the vector.

But you can notice we use the same and simple script function “qm1” to multiply vectors of N variables. Also the inverse vector is easy to calculate and I think solving those equations in terms of vectors and quaternions could be useful.

Investigating in internet about similar results, I found that is very similar the 4 variables vector described in this article to the “Conformal Geometric algebra” described here [ref.2].

In particular the sphere equation in conformal algebra $s^*=x_1 \wedge x_2 \wedge x_3 \wedge x_4$.

In a 4x4 variables systems, only can represent until sphere primitives. To represent ellipses, parabolas, hyperbolas,(also spheres) it will necessary a 9 variables [ref.3] (in 3 dimensions) vector system.

Maybe the change of perspective shown here to solve equations be useful to simplify the algebra.

References

[1] https://en.wikipedia.org/wiki/Geometric_algebra
[2] http://www.gaalop.de/dhilden_data/CLUScripts/CandG_PrePrint.pdf
[3] https://en.wikipedia.org/wiki/Ellipse