Project Report

Project Idea:

Compare $U_{l'm'lm}(\vec{a}, A, p, s)$ numerically with

$$\exp\left[u_{l'm'lm}(X; p, s)\right]$$

where $g = e^X$, $X \in se(3)$, and

$$u_{l'm'lm}(X; p, s) = \sum x_i u_{l'm'lm}(E_i; p, s)$$

Solution:

Using equation (12.95) in [1],

$$U_{l',m';l,m}(a, A; p, s) = \sum_{j=-l}^{l} [l', m' \mid p, s \mid l, j](a) U_{jm}(A, l)$$

and several relating equations from [1], we can derive the matrix element U of IURs for $SE(3)$, and the explicit expression for u.

Notice that both U and u are 4-dimensional matrices, thus it is hard to numerically prove their exponential relation, i.e.,

$$U = \exp(u)$$

According to [2], a 4-indedx matrix like U and u above can be rewrite into a 2-index matrix, as long as the following conditions are satisfied:

$$\begin{cases} s \le l, l' \le L \\ |m| \le l, |m'| \le l' \end{cases}$$

and the indices of the 2-dimensional matrix and the indices of the 4-dimensional matrix have the following relation:

$$U^s_{l'm'lm}(p) = U^s_{ij}(p)$$

$$\begin{cases} i = l'(l'+1) + m' - s^2 + 1 \\ j = l(l+1) + m - s^2 + 1 \end{cases}$$

thus, by transforming U and u into two 2-dimensional matrices, we can numerically compare the value of U and $\exp(u)$ using MATALB.

To prove the credibility of the MATLAB scripts results, I divided the proof into 3 steps as following:

1. Show the *IUR_SE3.m* function generates correct terms for certain U by comparing the results with the explicit formed derived from [2] (page 335):

set $A = I(3)$, $a = \frac{\sqrt{3}}{3}\begin{bmatrix}1\\1\\1\end{bmatrix}$, $p = \frac{\pi}{2}$, $s = 0$,

(i) for $l' = 1, m' = 0, l = 0, m = 0$

then the result calculated from *IUR_SE3* function is

```
IUR_SE3(a, A, p, s, l1, l, m1, m)
```

```
ans = 0.0000 + 0.5732i
```

which is exactly the same as

```
1i*sqrt(3)*cos(el)*(-cos(r*p)+sin(r*p)/(r*p))/(r*p)
```

```
ans = 0.0000 + 0.5732i
```

(ii) for $l' = 2, m' = 1, l = 0, m = 0$
the result calculated from *IUR_SE3* function is

```
IUR_SE3(a, A, p, s, l1, l, m1, m)
```

```
ans = 0.1255 - 0.1255i
```

which is nearly idencical to

```
-1/r^3/p^3 * sqrt(15/8) * exp(-1i*az) ...
    * (3*r*p*cos(r*p) -3*sin(r*p) ...
    +r^2*p^2*sin(r*p)) * sin(2*el)
```

```
ans = 0.1254 - 0.1254i
```

thus, it is fair to say that the result from *IUR_SE3* function is quite credible.

2. Show that U is actually a unitary representation of $SE(3)$, i.e.
$$U(g_1 \circ g_2, p, s) = U(g_1, p, s)U(g_2, p, s)$$
Here, I set $p = \frac{\pi}{2}, s = 2, l' = 5, l = 5$

randomly generate A_1 and a_1, A_2 and a_2

```
a1 = rand(3,1)
```

```
a1 = 3×1
    0.0844
    0.3998
    0.2599
```

```
A1 = gen_SO3
```

```
A1 = 3×3
   -0.4503    0.8402    0.3023
   -0.3368    0.1537   -0.9289
   -0.8269   -0.5201    0.2138
```

```
a2 = rand(3,1)
```

```
a2 = 3×1
    0.1818
    0.2638
    0.1455
```

```
A2 = gen_SO3
```

```
A2 = 3×3
     0.8901    0.3730    0.2620
     0.2918   -0.9078    0.3012
     0.3502   -0.1917   -0.9169
```

calculate $(g_1 \circ g_2)(A_1A_2, A_1a_2 + a_1)$

```
aa = A1*a2+a1;
AA = A1*A2;

p = pi/2;
s = 2;
l1 = 5;
l = 5;
```

calculate $U_1(g_1, p, s)$, $U_2(g_2, p, s)$ respectively:

```
U1 = IUR_SE3(a1, A1, p, s, l1, l);
U2 = IUR_SE3(a2, A2, p, s, l1, l);
```

calculate $U_1(g_1, p, s)U_2(g_2, p, s)$

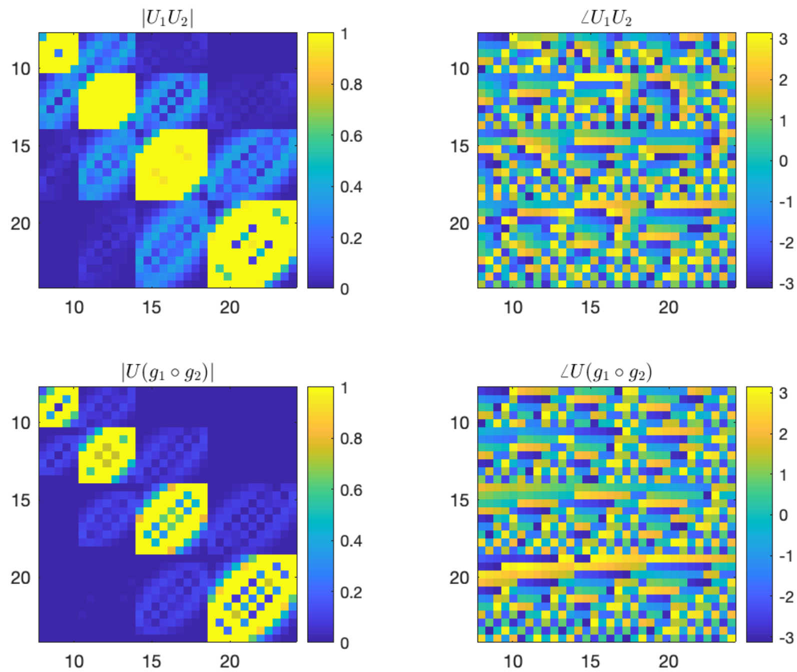
```
U_multi = U1*U2;
```

calculate $U(g_1 \circ g_2, p, s)$

```
UU = IUR_SE3(aa, AA, p, s, l1, l);
```

compare U_1U_2 and $U(g_1 \circ g_2)$ using *imagesc* function:

```
L = max(l1(end), l(end));
size = (L+1)^2-s^2;
h = figure; set(h, 'defaulttextinterpreter', 'latex');
mid_part = floor(size/4):1:ceil(size*3/4);
clims = [0 1];  % ignore terms with extreme values
subplot(2,2,1); imagesc(mid_part, mid_part, abs(U_multi), clims);
colorbar; title('$|U_1 U_2|$');
subplot(2,2,2); imagesc(mid_part, mid_part, angle(U_multi)); colorbar;
title('$\angle U_1 U_2$')
subplot(2,2,3); imagesc(mid_part, mid_part, abs(UU), clims); colorbar;
title('$|U(g_1 \circ g_2)|$');
subplot(2,2,4); imagesc(mid_part, mid_part, angle(UU)); colorbar;
title('$\angle U(g_1 \circ g_2)$');
```



As we can see from above, there is a similar pattern in the absolute value and angle value plot of $U(g_1, p, s)U(g_2, p, s)$ and $U(g_1 \circ g_2, p, s)$, thus, we can conclude that

$$U(\, g_1 \circ g_2, p, s) = U(g_1, p, s)U(g_2, p, s)$$

3. Prove $U = \exp{(u)}$ numerically using *imagesc* function

(i) For $p = \frac{\pi}{2}$, $s = 0$, set $l' = 5$, $l = 5$, thus $U \in M_{36\times36}$

randomly generate A and a:

```
a = rand(3,1)
```

```
a = 3×1
    0.5472
    0.1386
    0.1493
```

```
A = gen_SO3
```

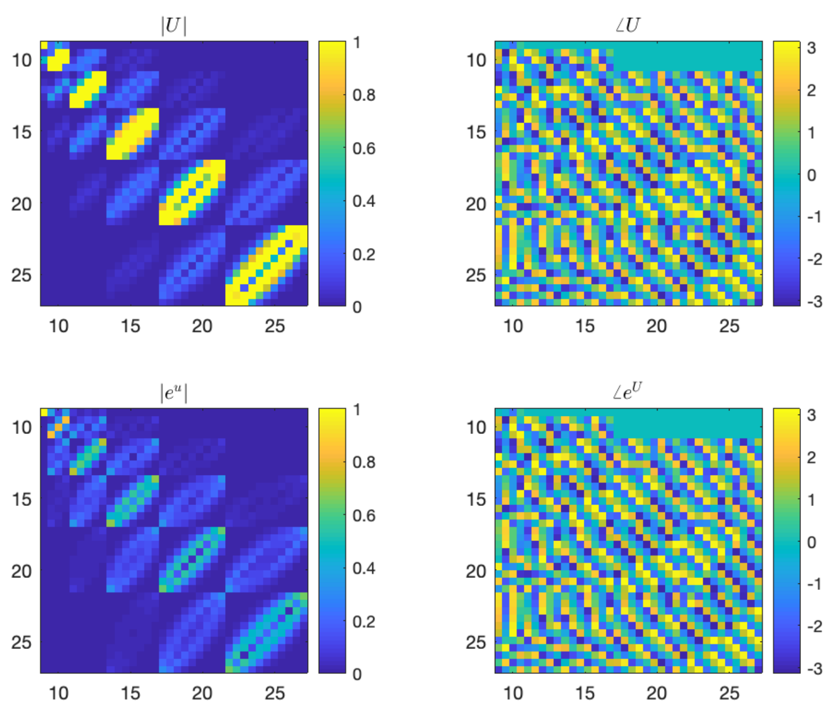
```
A = 3×3
   -0.9996   -0.0145   -0.0226
   -0.0236    0.8774    0.4792
    0.0129    0.4796   -0.8774
```

```
p = pi/2;
s = 0;
l1 = 5;
l = 5;
```

```
U = IUR_SE3(a, A, p, s, l1, l);

G = [A, a; zeros(1,3), 1];  % G \in SE(3)
g = logm(G);                % g \in se(3)

u = diff_iur_se3(g, p, s, l1, l);

U_exp = expm(u);
```

compare U and $exp(u)$ using *imagesc* function:

```
L = max(l1(end), l(end));
size = (L+1)^2-s^2;
h = figure; set(h, 'defaulttextinterpreter', 'latex');
mid_part = floor(size/4):1:ceil(size*3/4);
clims = [0 1];  % ignore terms with extreme values
subplot(2,2,1); imagesc(mid_part, mid_part, abs(U), clims); colorbar;
title('$|U|$');
subplot(2,2,2); imagesc(mid_part, mid_part, angle(U)); colorbar;
title('$\angle U$')
subplot(2,2,3); imagesc(mid_part, mid_part, abs(U_exp), clims); colorbar;
title('$|e^u|$');
subplot(2,2,4); imagesc(mid_part, mid_part, angle(U_exp)); colorbar;
title('$\angle e^U$');
```



Comparing plots of the absolute value and angle of U and $\exp(u)$, we can draw the conclusion that under this condition ($s = 0$),

$$U = \exp(u)$$

(ii) For $p = \frac{\pi}{2}$, $s = 3$, set $l' = 6$, $l = 6$, thus $U \in M_{40\times 40}$

randomly generate A and a:

```
a = rand(3,1)
```

```
a = 3×1
    0.9172
    0.2858
    0.7572
```

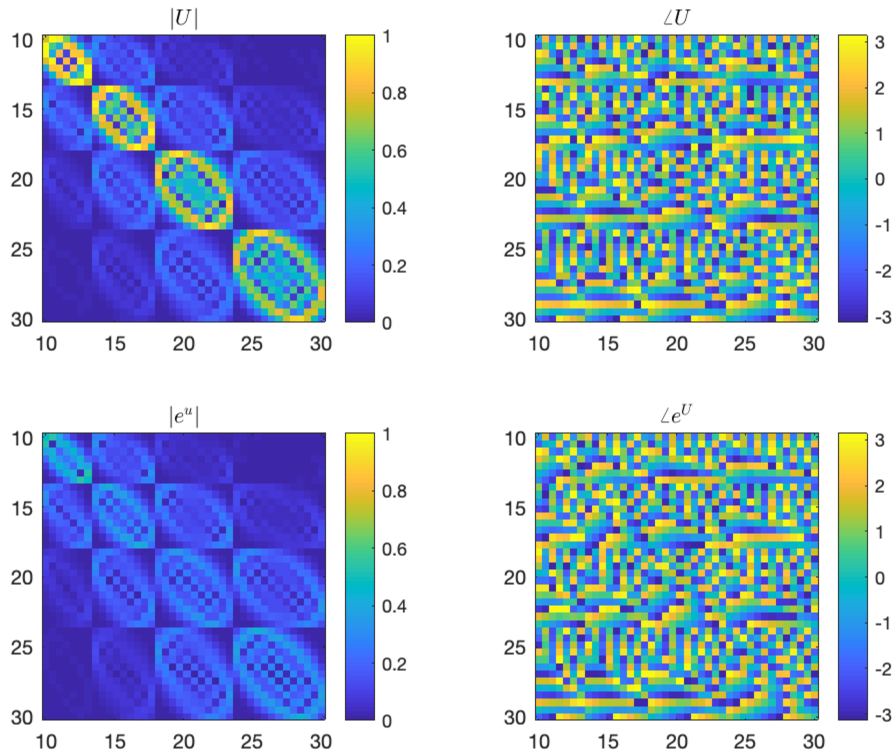
```
A = gen_SO3
```

```
A = 3×3
   -0.4211   -0.9068    0.0218
    0.3242   -0.1729   -0.9300
    0.8471   -0.3845    0.3668
```

```
p = pi/2;
s = 3;
l1 = 6;
l = 6;

U = IUR_SE3(a, A, p, s, l1, l);

G = [A, a; zeros(1,3), 1];  % G \in SE(3)
g = logm(G);                % g \in se(3)

u = diff_iur_se3(g, p, s, l1, l);

U_exp = expm(u);
```

compare U and $exp(u)$ using *imagesc* function:

```
L = max(l1(end), l(end));
size = (L+1)^2-s^2;
h = figure; set(h, 'defaulttextinterpreter', 'latex');
mid_part = floor(size/4):1:ceil(size*3/4);
clims = [0 1];  % ignore terms with extreme values
subplot(2,2,1); imagesc(mid_part, mid_part, abs(U), clims); colorbar;
title('$|U|$');
subplot(2,2,2); imagesc(mid_part, mid_part, angle(U)); colorbar;
title('$\angle U$')
subplot(2,2,3); imagesc(mid_part, mid_part, abs(U_exp), clims); colorbar;
title('$|e^u|$');
subplot(2,2,4); imagesc(mid_part, mid_part, angle(U_exp)); colorbar;
title('$\angle e^U$');
```



Comparing plots of the absolute value and angle of U and $\exp(u)$, we can draw the conclusion that under this condition ($s \neq 0$),

$$U = \exp(u)$$

Notice that there is a slightly difference between the magnitude range of the absolute value plot of U and $\exp(u)$, I think it is due to the fact that the matrices calculated above is a "finite approximation" of the infinite IUR matrices.

References:

1. Chirikjian, G.S., *Stochasstic Models, Information Theory, and Lie Groups, Volume 2,* Birkhauser Basel, 2012
2. Chirikjian, G.S., Kyatkin, A.B., *Engineering Applications of Noncommutative Harmonic Analysis,* CRC Press, Boca Raton, FL, 2001

MATLAB script (main functions):

```
function U = IUR_SE3(a, A, p, s, l1, l, m1, m)
%IUR_SE3    Irreducible Unitary Representations for SE(3)
%   U = IUR_SE3(a, A, p, s, l1, l)
%   U = IUR_SE3(a, A, p, s, l1, l, m1, m)
%
%   Input:
%    - a: translation, a 3*1 matrix
%    - A: rotation, a 3*3 matrix in SO(3)
%    - p: positive real number, first dual index
%    - s: integer, second dual index
%    - l1, m1, l, m: integers, sequenced indices of U, where
%           l1, l \ge abs(s),
%           m1 \in {-l1, l1},
%           m \in {-l, l}
%   Output:
%    - U: IUR for SE(3), l1*m1*l2*m2 matrix

%-- Auther: hshi17 11/17/18 --%

    if ~isequal(size(a), [3,1])
        disp('a has wrong dimentsion, set a = [0;0;0]');
        a = zeros(3,1);
    end

    if ~isequal(size(A), [3,3])
        disp('A has wrong dimentsion, set A = eye(3)');
        A = eye(3);
    end

    if l1(1) < abs(s)
        disp('value of l1 is incorrect');
        U = NaN; return;
    end
    if l(1) < abs(s)
        disp('value of l is incorrect');
        U = NaN; return;
    end

    if nargin ~= 6 && nargin ~= 8
        disp('input error');
        U = NaN; return;
    end

    L = max(l1(end), l(end));
    U_size = (L+1)^2 - s^2;     % size for 2-dim matrix
    U = zeros(U_size, U_size);

    for i = 1:U_size
        for j = 1:U_size
            lc = ceil(sqrt([i j]+s^2))-1;   % current l1 and l
```

```
            if (lc(1) <= l1(end)) && (lc(2) <= l(end))    % in boundary
                mc = [i j] - lc .* (lc+1) -1 + s^2;     % current m1 and m

                U_temp = zeros(2*lc(2)+1,1);
                for k = -lc(2):1:lc(2)
                    U_temp(k+lc(2)+1) = ...
    IUR_SE3_trans(a, lc(1), mc(1), p, s, lc(2), k) ...
    * IUR_SO3(A, lc(2),k, mc(2)).';
                end
                U(i,j) = sum(U_temp);
            end
        end
    end

    if nargin == 8  % l1, l, m1, m are index numbers
        if m1 > l1
            disp('value of m1 is incorrect');
            U = 0; return;
        end
        if m > l
            disp('value of m is incorrect');
            U = 0; return;
        end
        U = U(l1*(l1+1)+m1-s^2+1, l*(l+1)+m-s^2+1);
    end

end

function U = IUR_SE3_trans(a, l1, m1, p, s, l, m)

    [az, el, r] = cart2sph(a(1), a(2), a(3));

    U = zeros((l1+l)-abs(l1-l)+1,1);

    for k = abs(l1-l):1:(l1+l)
        U(k-abs(l1-l)+1) = ...
            (1i)^k * sqrt((2*l1+1)*(2*k+1)/(2*l+1)) ...
            * besselj_sph(k, p*r) ...
            * ClebshGordan(k, 0, l1, s, l, s) ...
            * ClebshGordan(k, m-m1, l1, m1, l, m) ...
            * sph_har(az, el, k, m-m1);
    end

    U = sqrt(4*pi) * sum(U);
end

function J = besselj_sph(nu, z)
   J = sqrt(pi/2./z) .* besselj(nu+0.5, z);
end

function C = ClebshGordan(j1, m1, j2, m2, j, m)
    C = (-1).^(m+j1-j2) .* sqrt(2*j+1) ...
        .* Wigner3j([j1, j2, j], [m1, m2, -m]);
end
```

```
function u = diff_iur_se3(X, p, s, l1, l, m1, m)
%DIFF_IUR_SE3   Differentiation of IUR for SE(3)
%   u = diff_iur_se3(X, p, s, l1, l);
%   u = diff_iur_se3(X, p, s, l1, l, m1, m);
%
%   Input:
%   - X: 4x4 matrix in se(3)
%   - p: positive real number, first dual index
%   - s: integre, second dual index
%    - l1, m1, l, m: integers, indices of U, where
%           l1, l \ge abs(s),
%           m1 \in {-l1, l1},
%           m \in {-l, l}
%   Output:
%    - u: differentiation of IUR for SE(3), l1*m1*l2*m2 matrix

%-- Auther: hshi17 12/20/18 --%

    if l1(1) < abs(s)
        disp('value of l1 is incorrect');
        u = NaN; return;
    end
    if l(1) < abs(s)
        disp('value of l is incorrect');
        u = NaN; return;
    end

    if nargin ~= 5 && nargin ~= 7
        disp('input error');
        u = NaN; return;
    end

    x = se32vec(X);

    L = max(l1(end), l(end));
    u_size = (L+1)^2 - s^2;
    u = zeros(u_size, u_size);

    if nargin == 7  % l1, l, m1, m are index numbers
        if m1 > l1
            disp('value of m1 is incorrect');
            u = 0; return;
        end
        if m > l
            disp('value of m is incorrect');
            u = 0; return;
        end
        u = diff_iur_se3_one(x, p, s, l1, l, m1, m);
    end

    for i = 1:u_size
        for j = 1:u_size
            lc = ceil(sqrt([i j]+s^2))-1;   % current l1 and l
```

```
            if (lc(1) <= l1(end) && lc(2) <= l(end))     % in boundary
                mc = [i j] - lc .* (lc+1) -1 + s^2;      % current m1 and m

                u(i,j) = ...
           diff_iur_se3_one(x, p, s, lc(1), lc(2), mc(1), mc(2));
            end
        end
    end

end

function u = diff_iur_se3_one(x, p, s, l1, l, m1, m)

    if l == 0
        u = 0;
    else

        c_mN = c_func(l, -m);
        c_mP = c_func(l, m);

        sigma_l     =   (l1==l);
        sigma_l1N   =   ((l1-1)==l);
        sigma_l1P   =   ((l1+1)==l);
    %     sigma_lN    =   (l1==(l-1));
    %     sigma_lP    =   (l1==(l+1));
        sigma_m     =   (m1==m);
        sigma_m1N   =   ((m1-1)==m);
        sigma_m1P   =   ((m1+1)==m);
        sigma_mN    =   (m1==(m-1));
        sigma_mP    =   (m1==(m+1));

        u1 = -1i/2 * c_mN .* sigma_l .* sigma_m1P ...
            - 1i/2 * c_mP .* sigma_l .* sigma_m1N;
        u2 = +1/2 * c_mN .* sigma_l .* sigma_m1P ...
            - 1/2 * c_mP .* sigma_l .* sigma_m1N;
        u3 = -1i * m .* sigma_l .* sigma_m;

        gamma_m1N = gamma_func(s, l1, -m1);
        gamma_m1P = gamma_func(s, l1, m1);
        gamma_mN = gamma_func(s, l, -m);
        gamma_mP = gamma_func(s, l, m);

        lambda_mN = lambda_func(s, l, -m);
        lambda_mP = lambda_func(s, l, m);

        u4 = -1i*p/2 * gamma_m1N .* sigma_mP .* sigma_l1N ...
            + 1i*p/2 * lambda_mP .* sigma_mP .* sigma_l ...
            + 1i*p/2 * gamma_mP .* sigma_mP .* sigma_l1P ...
            + 1i*p/2 * gamma_m1P .* sigma_mN .* sigma_l1N ...
            + 1i*p/2 * lambda_mN .* sigma_mN .* sigma_l ...
            - 1i*p/2 * gamma_mN .* sigma_mN .* sigma_l1P;

        u5 = -p/2 * gamma_m1N .* sigma_mP .* sigma_l1N ...
            + p/2 * lambda_mP .* sigma_mP .* sigma_l ...
```

```
            + p/2 * gamma_mP .* sigma_mP .* sigma_l1P ...
            - p/2 * gamma_m1P .* sigma_mN .* sigma_l1N ...
            - p/2 * lambda_mN .* sigma_mN .* sigma_l ...
            + p/2 * gamma_mN .* sigma_mN .* sigma_l1P;

        kappa1 = kappa_func(s, l1, m1);
        kappa = kappa_func(s, l, m);

        u6 = 1i*p * kappa1 .* sigma_m .* sigma_l1N ...
            + 1i*p * s*m/l/(l+1) .* sigma_m .* sigma_l ...
            + 1i*p * kappa .* sigma_m .* sigma_l1P;

        u = x(1) * u1 + x(2) * u2 + x(3) * u3 ...
            + x(4) * u4 + x(5) * u5 + x(6) * u6;
    end
end

function gamma = gamma_func(s, l, m)
%   gamma is defined as gamma_{l, m}^s
    if l ~= 0
        gamma = sqrt( ...
            (l^2-s^2) * (l-m) * (l-m-1) ...
                / l^2 / (2*l-1) / (2*l+1) ...
            );
    else
        gamma = 0;
    end
end

function lambda = lambda_func(s, l, m)
%   lambda is defined as lambda_{l, m}^s
    if l ~= 0
        lambda = s * sqrt((l-m) * (l+m+1)) ...
            / l / (l+1);
    else
        lambda = 0;
    end
end

function kappa = kappa_func(s, l, m)
%   kappa is defined as kappa_{l, m}^s

    if l ~= 0
        kappa = sqrt( ...
            (l^2-m^2) * (l^2-s^2) ...
                / l^2 / (2*l-1) / (2*l+1) ...
            );
    else
        kappa = 0;
    end
end
```