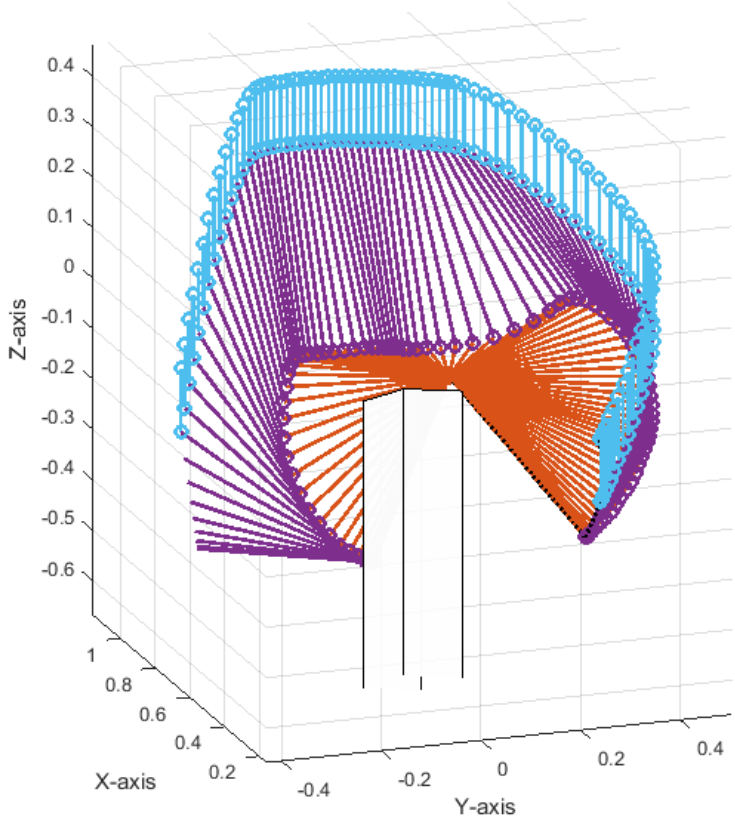
Final Project

MAE547 - Modeling and Control of Robots

Adil Ansari

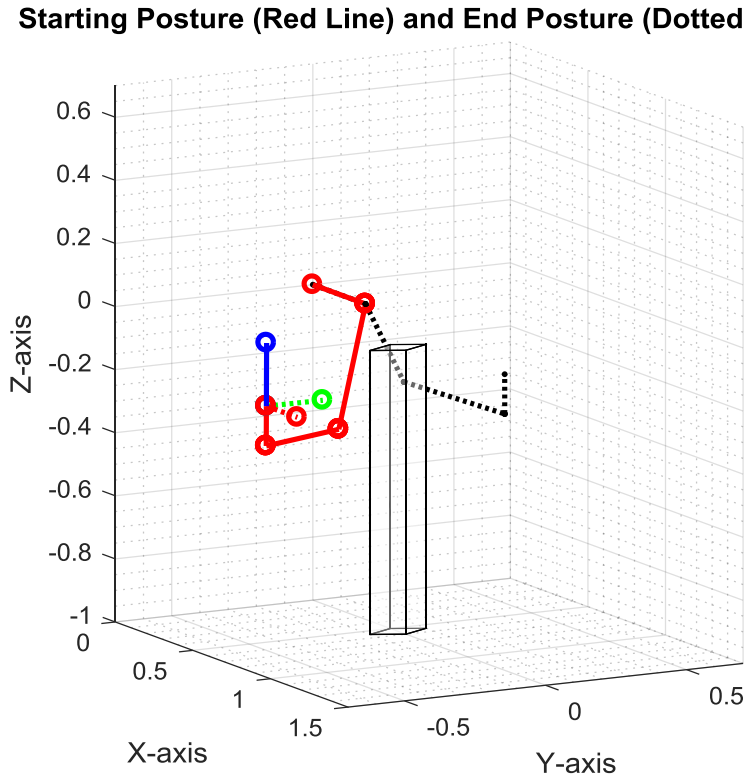
Saturday, November 25, 2017

Contents



Problem Description

The goal of this project was to move the robotic arm from posture A to posture B: This can be depicted by the colored plot depicting the posture A and the dashed line depicting posture B.



The angles and position can be describes as such.

$$q_a = [-109.19 \quad 80.81 \quad 64.66 \quad 70.93 \quad 0.25 \quad -88.23 \quad 63.10]^T \ (degrees)$$

$$q_b = [-53.41 \quad 112.63 \quad -13.84 \quad 92.16 \quad -41.09 \quad -97.44 \quad -23.31]^T \ (degrees)$$

The resultant end effector Cartesian position are as such:

$$x_a = [0.5841 \quad -0.4857 \quad -0.2321 \quad 2.0398 \quad 0.0063 \quad -2.0405\ ]^T$$

$$x_a = [0.5502 \quad -0.3774 \quad -0.2222 \quad -1.2766 \quad 0.0013 \quad 1.2766\ ]^T$$

Note that the first 3 elements describe the xyz position while the next three values are the zyz angles. These angles' rotation matrix is a 3x3 eye matrix. Also the desired orientation is 3x3 eye matrix throughout the trajectory. It should be noted the zyz angles have format $[\beta\ 0\ -\beta]^T$ since that represents a eye matrix orientation. In Euler ZYX it would be $[0\ 0\ 0]^T$. The primary method to solve the trajectory was done by controlling the end effector position. The forward kinemtics were not initated by an eye4x4 matrix

but rather the orientation matrix of 4x4 homogoneous matrix was replaced by Ry(90)*Rx(90) as the robot origin isn't pointed in the upward eye matrix direction.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_{\boldsymbol{a}}^{\dagger}\boldsymbol{e}$$

Where change in joint angles is described by the pseudo-inverse of the analytical jacobian multiplied by e.

$$\boldsymbol{e} = \boldsymbol{x}_{\boldsymbol{d}} - \boldsymbol{x}_{\boldsymbol{e}}$$

e is the error which is the difference between desired end effector position and current end effector position. These positions are described in the Cartesian coordinates and orientation in ZYZ angular format as previously mentioned. The analytical jacobian which is a function of joint angles and the DH parameters isn't square due to kinematic redundancy and hence pseudo inverse is invoked describes as such:

$$\boldsymbol{J}_{\boldsymbol{a}}^{\dagger} = \boldsymbol{J}_{\boldsymbol{a}}^{\boldsymbol{T}}(\boldsymbol{J}_{\boldsymbol{a}}\boldsymbol{J}_{\boldsymbol{a}}^{\boldsymbol{T}})^{-\mathbf{1}}$$

Analytical Jacobian can be computed by computing jacobian and multiplying it by inverse of matrix Ta:

$$\boldsymbol{T}_{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T}_{\boldsymbol{a}}(\boldsymbol{\varphi}) \end{bmatrix}$$

However since the robot is kinematically redundant i.e. 7 degrees of freedom in joint space and 6 degrees of freedom in Cartesian space, the end effector can end up at the desired end effector position without the desired posture matching desired pose B as there are infinitely many solutions. As a result methods were devised as such that end effector position at the end of trajectory had joint angles matching the desired joint angles. The additional contraints were computed by computing the vector $\dot{q}_0$ in the joint space.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_{\boldsymbol{a}}^{\dagger}(\boldsymbol{x}_{\boldsymbol{d}} - \boldsymbol{x}_{\boldsymbol{e}}) + \left(\boldsymbol{I}_{\boldsymbol{n}} - \boldsymbol{J}_{\boldsymbol{a}}^{\dagger}\boldsymbol{J}_{\boldsymbol{a}}\right)\dot{\boldsymbol{q}}_{\mathbf{0}}$$

Where $\dot{q}_0$ in the joint space is describes as the derivative of objective function w(q) with respect to joint angles:

$$\dot{\boldsymbol{q}}_{\mathbf{0}} = k(t)\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}}$$

The goal is to maximize joint w(q) as such that the constraints are satisfied. The strength of each impacts can be throttled by a variable $k_n(t)$ as a function of time or iteration.

Approach:

Points were generated in the Cartesian between starting and ending end effector position and orientation. These points were mapped on pseudo time using 1D interpolation. The method used for interpolation used was PCHIP (Piecewise Cubic

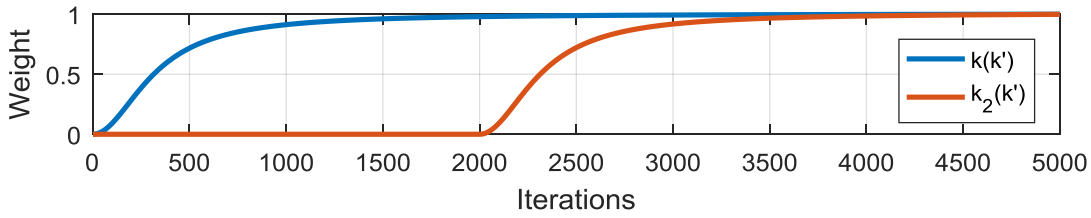
Hermite Interpolating Polynomial). The advantage of this is that it doesn't introduced spurious oscillations as would a simple interpolating polynomial or spline on individual chunks of point. The result of the interpolation is desired positon as a function of time ($\boldsymbol{x_d(t)}$). This results in the following system of differential equation excluding the additional constraints' aspect:

$$\dot{\boldsymbol{q}} = K\boldsymbol{J_a}(\boldsymbol{q})^{\dagger}\big(\boldsymbol{x_d}(t) - \boldsymbol{x_e}(\boldsymbol{q})\big)$$

K is the strength of the proportional controller.

Additional Constraints

In order to smoothen the acceleration in the beginning an additional impact factor was added k(t). This value varies between 0 and 1. An additional weight factor was used for activating constraints that become important later in the trajectory.



Iterations and time are proportional whereby one iteration's Δt is 5ms.

$$\dot{\boldsymbol{q}} = K\boldsymbol{J_a}(\boldsymbol{q})^{\dagger}\big(\boldsymbol{x_d}(t) - \boldsymbol{x_e}(\boldsymbol{q})\big)k(t) + \big(\boldsymbol{I_n} - \boldsymbol{J_a^{\dagger}}\boldsymbol{J_a}\big)\dot{\boldsymbol{q}}_{\boldsymbol{0}}$$

$$\dot{\boldsymbol{q}}_{\boldsymbol{0}} = k(t)\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}}$$

Limit Avoidance and joint space control

Maximizing the following function keeps the joint angle between ranges if k_2 is zero. Once k_2 becomes 1, the function is maximized by joint angles approaching desired angle pose:

$$\boldsymbol{w(q)} = -\frac{\boldsymbol{k(t)}}{\boldsymbol{2n}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}\left(\frac{\boldsymbol{q_i} - \boldsymbol{q_b}\boldsymbol{k_2(t)}}{\boldsymbol{q_{i_{Max}}} - \boldsymbol{q_{i_{Min}}}}\right)^{\boldsymbol{2}}$$

Obstacle Avoidance

$$\boldsymbol{w(q)} = \boldsymbol{k(t)}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}\boldsymbol{d(q)}$$

Where d(q) is the distance function as function of all joint positions between walls and obstacles. However computing analytical derivate with piecewise functions with complex interfaces can be computationally intensive and hence the interpolant end effector

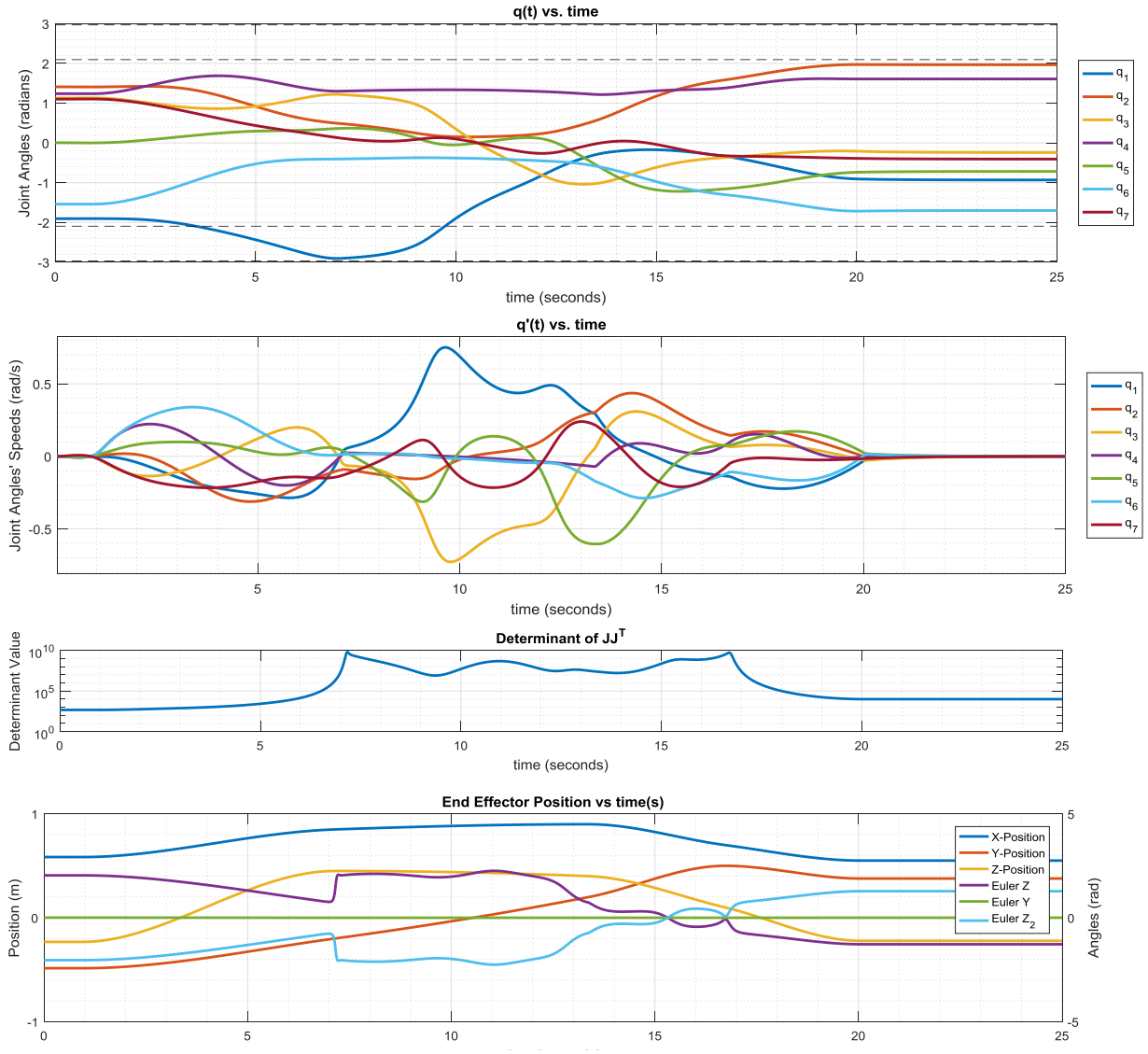
positions were manipulated manually to compute trajectory on the joint space that avoided the obstacle.

Singularity Avoidance

The singularity can be avoided by preventing the determinant from approaching zero by maximizing it.

$$\boldsymbol{w(q) = \sqrt{det(J(q)J(q)^T)}}$$

Results



Code:

Main matlab file

```
qa = [-109.19 80.81 64.66 70.93 0.25 -88.23 63.10]*pi/180;

qb = [-53.41 112.63 -13.84 92.16 -41.09 -97.44 -23.31]*pi/180;


xa = fwdkin(qa);
xi = [0.85;-.2;.45;0.7632;0.000;-0.7632];
xj = [0.9;.2;.4;-0.7632;0.000;0.7632];
xk = [0.7;.5;.1;-0.7632;0.000;0.7632];


[xb, ~, p] = fwdkin(qb);
rpb = reshape(p,[3 8]);

intPos = @(t) interp1([0 10 85 160 220 240 300]*5000/300,[xa xa xi xj xk xb
xb]',t,'pchip')';
%
K = .1;

k0 = 1;
N = 5000;
q = qa';

%%%%%%%%%%%%%
qt = zeros(7,N); qt(:,1) = q; %joint angles  storgae
xt = zeros(6,N); xt(:,1) = xa; %end effector positiion storage
sing = zeros(1,N);

figure(1)

for k = 1:N
    [xe, Ja, p, ~, Tf] = fwdkin(q); %forward Kinematics and Jacobian computer
    rp = reshape(p,[3 8]);
    if mod(k,40)==1
        rp = reshape(p,[3 8]);
        %subplot(1,2,1)
        col = 'rgb'; le = .2;
        for n = 1:3
            plot3(rp(1,end)+le*[0 Tf(1,n)],rp(2,end)+le*[0 Tf(2,n)],...
                rp(3,end)+le*[0 Tf(3,n)],[':o' col(n)],'linewidth',2);
            hold on
        end
        for i = 1:7
            plot3(rp(1,i:i+1),rp(2,i:i+1),rp(3,i:i+1),'ro-','linewidth',2)
            plot3(rpb(1,i:i+1),rpb(2,i:i+1),rpb(3,i:i+1),'k.:','linewidth',2)
        end

        xlabel('X-axis')
        ylabel('Y-axis')
        zlabel('Z-axis')
        grid on; grid minor;
```

```
                view(az,el)
%         view([93+k/4000*120 31.4159])
        title(num2str(k))
        vertices = ...
            [0.55, -0.1, -0.1;
            0.60, 0, -0.1;
            0.55, 0.1, -0.1;
            0.50, 0, -0.1;
            0.55, -0.1, -1;
            0.60, 0, -1;
            0.55, 0.1, -1;
            0.50, 0, -1];
        faces = [1 5 6 2;
            1 2 3 4;
            2 6 7 3;
            8 7 6 5;
            8 5 1 4];

patch('Vertices',vertices,'Faces',faces,'FaceColor','w','facealpha',.4)
        axis equal
        axis([0 1.5 -.7 .7 -1 .7]); hold off;
        drawnow;
        %writeVideo(vid,getframe(gcf));
        [az, el] = view;
    end
    tp = intPos(k);
    e = tp - xe; %error

    sing(k) = (det(Ja*Ja'));

    impactFactor = k^2/(k^2+N*20)*1.004;
    impactFactor2 = heaviside(k-3500)*(k-3500)^2/((k-3500)^2+N*20)*(1+4/90);
    Jap = pinv(Ja); %psuedo inverse
    qdot = Jap*e*K*impactFactor + (eye(7)-Jap*Ja)*...
        (k0*impactFactor*limitsAvoidance(q,qb*impactFactor)
+impactFactor*0*singularityAvoidance(q)); %Use Jacobian to compute dq/dt
    qdot_max = [98;98;100;130;140;180;180]*pi/180;
    if sum(qdot/.005>qdot_max) fprintf('Exceeded Speed Limit\n'); end
    q = q + qdot; %update q
    qt(:,k+1) = q; %Trajectory Accumulation
    xt(:,k+1) = xe; %End Effector Trajectory Accumulation
end
sing(end+1) = sing(end);
t = 0:0.005:25;
qtdot = diff(qt')/.005;
figure(2)
plot(t,qt,'linewidth',2);hold on
plot(t,[t*0+2*pi/3;t*0-2*pi/3;t*0+17*pi/18;t*0-17*pi/18]','k--');hold off
legend(num2str((1:7)','q_%i'),'location','eastoutside')
grid on;grid minor
title('q(t) vs. time')
xlabel('time (seconds)');
ylabel('Joint Angles (radians)');
figure(3);
tc = (0.005:0.005:25) - 0.005/2;
plot(tc,qtdot,'linewidth',2);hold on
```

```
plot(tc,[tc'*0+98*pi/180,tc'*0-98*pi/180],'k--');hold off;
legend(num2str((1:7)','q_%i'),'location','eastoutside');
grid on;grid minor
title('q''(t) vs. time')
xlabel('time (seconds)');
ylabel('Joint Angles'' Speeds (rad/s)');

figure(4)
semilogy(t,sing,'linewidth',2)
grid on;grid minor
title('Determinant of JJ^{T}')
xlabel('time (seconds)');
ylabel('Determinant Value');

figure(5)
[ax,h1,h2] = plotyy(t,xt(1:3,:),t,xt(4:6,:));
ylabel(ax(1), 'Position (m)'); set(h1,'LineWidth',2);
ylabel(ax(2), 'Angles (rad)'); set(h2,'LineWidth',2);
legend('X-Position','Y-Position','Z-Position','Euler Z','Euler Y','Euler
Z_2')
grid on;grid minor
xlabel('time (seconds)'); title('End Effector Position vs time(s)')

% dlmwrite('Ansari Adil.txt',qt','precision','%.4f','delimiter',' ');
```

Forward Kinematics and Jacobian

```
function [xe, Ja, p, z, Tf] = fwdkin(q)
d = [.340 0 .400 0 .4 0 .126]';
alpha = [-pi/2 pi/2 pi/2 -pi/2 -pi/2 pi/2 0]';
a = zeros(7,1);
T = zeros(4,4,7);

for i = 1:7
    T(:,:,i) = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) 0;
        sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) 0;
        0 sin(alpha(i)) cos(alpha(i)) d(i);
        0 0 0 1];
end

Tf = eye(4); p = zeros(3,1,8); z = zeros(3,1,7);
Tf(1:3,1:3) = myrotmat(pi/2,'y')*myrotmat(pi/2,'z'); %initial rotation
for i = 1:7
    p(:,:,i) = Tf(1:3,4);
    z(:,:,i) = Tf(1:3,3);
    Tf = Tf*T(:,:,i);
end
xe = zeros(6,1);
xe(1:3,:) = Tf(1:3,4);

phi = atan2(Tf(2,3),Tf(1,3));
theta = atan2(sqrt(Tf(1,3)^2 + Tf(2,3)^2),Tf(3,3));
psi = atan2(Tf(3,2),-Tf(3,1));

% phi = atan2(Tf(2,1),Tf(1,1));
% theta = atan2(-Tf(3,1),sqrt(Tf(3,2)^2 + Tf(3,3)^2));
```

```
% psi = atan2(Tf(3,2),Tf(3,3));

xe(4) = phi;
xe(5) = theta;
xe(6) = psi;

p(:,:,8) = Tf(1:3,4);
J = zeros(6,7);

for i = 1:7
    J(:,i) = [cross(z(:,:,i),Tf(1:3,4)-p(:,:,i));z(:,:,i)];
end

temp = [[ -(cos(phi)*cos(theta))/sin(theta), -
(cos(theta)*sin(phi))/sin(theta), 1]
    [-sin(phi), cos(phi), 0]
    [cos(phi)/sin(theta), sin(phi)/sin(theta), 0]];
Tai = blkdiag(eye(3),temp); %Ta inverse
Ja = Tai*J; %Compute Analytical Jacobian
```

Limits Avoidance and joint space angle manipulation

```
function q0 = limitsAvoidance(q,a)
q1=q(1);q2=q(2);q3=q(3);q4=q(4);q5=q(5);q6=q(6);q7=q(7);
a1=a(1);a2=a(2);a3=a(3);a4=a(4);a5=a(5);a6=a(6);a7=a(7);

q0 = [  (81*(2*a1 - 2*q1))/(4046*pi^2)
    (9*(2*a2 - 2*q2))/(224*pi^2)
  (81*(2*a3 - 2*q3))/(4046*pi^2)
    (9*(2*a4 - 2*q4))/(224*pi^2)
  (81*(2*a5 - 2*q5))/(4046*pi^2)
    (9*(2*a6 - 2*q6))/(224*pi^2)
 (162*(2*a7 - 2*q7))/(8575*pi^2)];
```