Appendices

Table of contents

Appendix A: Matlab function implementing the optimization algorithm NSGA-II

1) Main function in Matlab: trajectory_planning_calculate.m

```
%% defining constraints

sx = 10;
q_desireds = load ('q_desireds.txt');
Q = [q_desireds(1:3,:)/1000;q_desireds(4:end,:)]';
QC = [ 0 1000/1000; 0 1000/1000; 0 500/1000; -1*pi 1*pi; -1*pi    1*pi];
VC = [ [3000 3000 3000]*sx/60/1000/1 3000/80*2*pi/60 3000/80*2*pi/60];
n_VC = 1;
AC = [ [15200 15200 15200]*sx/2/pi/1000 24600/80 24600/80]';
JC = [25 25 25 308 308]*1e2;
FC = [[2.39;2.39;2.39]*2*pi/(sx/1000);[0.637*80;0.637*80]/1];
n_hs = 2;
objectiveNr = 4;
kl = [1e0 1e3 1e1 1e0];
pop = 100; gen = 200;

%% optimizing
[gens_hs_obj4_total,gens_u_solution,gens_Q_solution,gens_V_solution,gens_A_solution,gens_J_
solution] = nsga_2(pop,gen,objectiveNr,kw,kl,Q,QC,VC,AC,JC,FC,n_VC,n_hs);

%% save the best found solution

save gens_hs_obj4_total.txt gens_hs_obj4_total -ASCII
save gens_u_solution.mat gens_u_solution
save gens_Q_solution.mat gens_Q_solution
save gens_V_solution.mat gens_V_solution
save gens_A_solution.mat gens_A_solution
save gens_J_solution.mat gens_J_solution


%% plot

ylabels = {'Total travelling time' 'Total energy' 'Integral of jerks squared' 'Integral of accelerations
squared' 'Cost function'};
for i = 4:-1:0
      fig_Nr = 5-i;
      figure (fig_Nr);
      plot(gens_hs_obj4_total(end-i,:));
      grid on;
      xlabel('Generation');
      ylabel(ylabels{fig_Nr});
end
```

2) Sub Matlab function program of NSGA-II: nsga_2.m

```
function
[gens_hs_obj4_total,gens_u_solution,gens_Q_solution,gens_V_solution,gens_A_solution,gens_J_
solution] = nsga_2(pop,gen,objectiveNr,kw,kl,Q,QC,VC,AC,JC,FC,n_VC,n_hs)
[pointNr,jointNr] = size(Q);
m = pointNr+2;
gens_hs_obj4_total = zeros(pointNr+1+4+1,gen);
gens_u_solution = zeros(m,jointNr,gen);
gens_Q_solution = zeros(m,jointNr,gen);
gens_V_solution = zeros(m,jointNr,gen);
gens_A_solution = zeros(m,jointNr,gen);
gens_J_solution = zeros(m-1,jointNr,gen);

if nargin < 2
    error('NSGA-II: Please enter the population size and number of generations as input
arguments.');
end
% Both the input arguments need to of integer data type
if isnumeric(pop) == 0 || isnumeric(gen) == 0
    error('Both input arguments pop and gen should be integer datatype');
end
% Minimum population size has to be 20 individuals
if pop < 20
    error('Minimum population for running this function is 20');
end
if gen < 5
    error('Minimum number of generations is 5');
end
pop = round(pop);
gen = round(gen);

V = pointNr+1;
M = objectiveNr;
min_range = 1e-3*ones(V,1);
max_range = 10*ones(V,1);
chromosome = initialize_variables(pop, M, V,Q,QC,VC,AC,JC,FC,n_VC,n_hs);
  chromosome = non_domination_sort_mod(chromosome, M, V);
for i = 1 : gen
    pool = round(pop/2);
    tour = 2;
    parent_chromosome = tournament_selection(chromosome, pool, tour);
    mu = 10;
    mum = 100;
    offspring_chromosome = ...
```

```
        genetic_operator(parent_chromosome, ...
        M, V, mu, mum, min_range, max_range,Q);
    [main_pop,temp] = size(chromosome);
    [offspring_pop,temp] = size(offspring_chromosome);
    clear temp
    intermediate_chromosome(1:main_pop,:) = chromosome;
    intermediate_chromosome(main_pop + 1 : main_pop + offspring_pop,1 : M+V) = ...
        offspring_chromosome;

    intermediate_chromosome = ...
        non_domination_sort_mod(intermediate_chromosome, M, V);
    chromosome = replace_chromosome(intermediate_chromosome, M, V, pop);
    w1 = kw(1);w2 =kw(2);w3 = kw(3);w4 = kw(4);
    l1 = kl(1);l2 =kl(2);l3 = kl(3);l4 = kl(4);      total = w1*chromosome(:,V + 1)/l1 + w2*chromosome(:,V + 2)/l2 + w3*chromosome(:,V + 3)/l3 + w4*chromosome(:,V + 4)/l4;

    [total_min,n_min] = min(total);
    hs_gen_min = chromosome(n_min,1:V)';
    gen_obj4_min = chromosome(n_min,V+1:V+M)';

    gens_hs_obj4_total(:,i) = [hs_gen_min;gen_obj4_min;total_min];

    u_gen_min = zeros(m,jointNr);
    Q_new = zeros(m,jointNr);
    acc = zeros(m,jointNr);
    v = zeros(m,jointNr);
    dacc = zeros(m-1,jointNr);

    for j_joint = 1:jointNr
        [acc(:,j_joint), Q_new(:,j_joint)] = cal_acc_q (Q(:,j_joint), hs_gen_min);
    end
    for j_p = 1:m
        if j_p == 1 || j_p == m
            v(j_p,:) = zeros(1,jointNr);
        else
            v(j_p,:) = -acc(j_p,:)/2*hs_gen_min(j_p) + (Q_new(j_p+1,:)-Q_new(j_p,:))/hs_gen_min(j_p) + ( acc(j_p,:)-acc(j_p+1,:) )/6*hs_gen_min(j_p);
        end
        u_gen_min(j_p,:) = Dynamics(Q_new(j_p,:),v(j_p,:),acc(j_p,:));
    end
    for j_p = 1:m-1
        dacc(j_p,:) = ( acc(j_p+1,j_joint)-acc(j_p,j_joint) )/hs_gen_min(j_p);
    end
    gens_u_solution(:,:,i) = u_gen_min;
```

```
        gens_A_solution(:,:,i) = acc;
        gens_Q_solution(:,:,i) = Q_new;
        gens_V_solution(:,:,i) = v;
        gens_J_solution(:,:,i) = dacc;
end
```

3) Sub Matlab function of generating time-interval lengths h_i, i=1, 2, …, 12: hs_generate.m

```
function hs = hs_generate(Q,VC,n_VC,n_hs)

[pointNr,jointNr] = size(Q);
m = pointNr+2;
Q_new = zeros(m,jointNr);

hs = zeros(m-1,1);

for large_times = 1:10000
        hs_min = zeros(pointNr-1,1);
        hs_tmp = zeros(pointNr-1,1);
        for j_p = 1:pointNr-1
                for j_joint = 1:jointNr
                        if j_joint == 1
                                hs_min(j_p) = abs( Q(j_p+1,j_joint) - Q(j_p,j_joint) )/VC(j_joint) * n_VC;
                        else
                                hs_min(j_p) = max(hs_min(j_p), abs( Q(j_p+1,j_joint) - Q(j_p,j_joint) )/VC(j_joint) * n_VC );
                        end
                end
                hs_tmp(j_p) = hs_min(j_p) * n_hs *( 1+rand(1) );
        end
        for try_times = 1:10000
                hs(1) = hs_tmp(1)*(0.5+1*rand(1));
                hs(2) = hs_tmp(1)*(0.5+1*rand(1));
                hs(m-1) = hs_tmp(pointNr-1)*(0.5+1*rand(1));
                hs(m-2) = hs_tmp(pointNr-1)*(0.5+1*rand(1));

                hs(3:m-3) = hs_tmp(2:m-4);

                for j_joint = 1:jointNr
                        [~, Q_new(:,j_joint)] = cal_acc_q(Q(:,j_joint), hs);
                end

                hs_min = [0; 0; hs_min(3:m-3); 0; 0];
```

```
            success_Nr = 0;
            for j_p = [1 2 m-2 m-1]
                for j_joint = 1:jointNr
                    if j_joint == 1
                        hs_min(j_p) = abs( Q_new(j_p+1,j_joint) -
Q_new(j_p,j_joint) )/VC(j_joint) * n_VC;
                    else
                        hs_min(j_p) = max(hs_min(j_p), abs( Q_new(j_p+1,j_joint) -
Q_new(j_p,j_joint) )/VC(j_joint) * n_VC );
                    end
                end
                if hs(j_p) <= hs_min(j_p)
                    success_Nr = 1;
                    break
                end
            end
            if success_Nr == 0
                return
            end
        end
end
```

4) Sub Matlab function of constraints: constraint.m

```
function flag = constraint(hs,Q,QC,VC,AC,JC,FC)

flag = 1;
[pointNr,jointNr] = size(Q);
m = pointNr+2;
ts = zeros(1,m);
for j = 2:m
    ts(j) = ts(j-1) + hs(j-1);
end
dtime = 0.005;
Q_new = zeros(m,jointNr);
acc = zeros(m,jointNr);
for j_joint = 1:jointNr
    [acc(:,j_joint), Q_new(:,j_joint)] = cal_acc_q (Q(:,j_joint), hs);
end

for j_joint = 1:jointNr
    for j_p = 1:m
        if abs( acc(j_p,j_joint) ) > AC(j_joint)
            flag = 0;
```

```
            return
        end
    end

    for j_p = 1:m-1
        if hs(j_p) <= abs( Q_new(j_p+1,j_joint) - Q_new(j_p,j_joint) )/VC(j_joint)
            flag = 0;
            return
        end

        dacc = abs( acc(j_p+1,j_joint)-acc(j_p,j_joint) )/hs(j_p);
        if dacc > JC(j_joint)
            flag = 0;
            return
        end

        v_1 = -acc(j_p,j_joint)/2*hs(j_p) + (Q_new(j_p+1,j_joint)-Q_new(j_p,j_joint))/hs(j_p)
+ ( acc(j_p,j_joint)-acc(j_p+1,j_joint) )/6*hs(j_p);
        if abs(v_1) > VC(j_joint)
            flag = 0;
            return
        end
        v_2 = acc(j_p+1,j_joint)/2*hs(j_p) + (Q_new(j_p+1,j_joint)-Q_new(j_p,j_joint))/hs(j_p)
+ ( acc(j_p,j_joint)-acc(j_p+1,j_joint) )/6*hs(j_p);
        if abs(v_2) > VC(j_joint)
            flag = 0;
            return
        end
        if acc(j_p,j_joint)-acc(j_p+1,j_joint) ~= 0
            tij = acc(j_p,j_joint)/( acc(j_p,j_joint)-acc(j_p+1,j_joint) );
            if tij*(tij - 1) < 0
                v_3                                                                    =
-hs(j_p)/2/(acc(j_p+1,j_joint)-acc(j_p,j_joint))*acc(j_p,j_joint)*acc(j_p+1,j_joint) ...
                    + (Q_new(j_p+1,j_joint)-Q_new(j_p,j_joint))/hs(j_p) ...
                    - hs(j_p)/6*(acc(j_p+1,j_joint)-acc(j_p,j_joint));
                if abs(v_3) > VC(j_joint)
                    flag = 0;
                    return
                end
            end
        end

        tt = ts(j_p) : dtime : ts(j_p+1);
        qq = acc(j_p,j_joint)/(6*hs(j_p))*(ts(j_p+1)-tt).^3 ...
```

```
                + acc(j_p+1,j_joint)/(6*hs(j_p))*(tt-ts(j_p)).^3 ...
                + (Q_new(j_p+1,j_joint)/hs(j_p) - hs(j_p)*acc(j_p+1,j_joint)/6)*(tt-ts(j_p)) ...
                + (Q_new(j_p,j_joint)/hs(j_p) - hs(j_p)*acc(j_p,j_joint)/6)*(ts(j_p+1)-tt);
            if min(qq) < QC(j_joint,1) || max(qq) > QC(j_joint,2)
                flag = 0;
                return
            end
        end

end

for j_p = 1:m
        if j_p == 1 || j_p == m
            v = zeros(1,jointNr);
        else
            v = -acc(j_p,:)/2*hs(j_p) + (Q_new(j_p+1,:)-Q_new(j_p,:))/hs(j_p) +
( acc(j_p,:)-acc(j_p+1,:) )/6*hs(j_p);
        end
        u = Dynamics(Q_new(j_p,:),v,acc(j_p,:));
        for j_joint = 1:jointNr
            if abs( u(j_joint) ) > FC(j_joint)
                flag = 0;
                return
            end
        end
end
end
```

5) Sub Matlab function of calculation for acceleration and the extra joint displacement:

cal_acc_q.m

```
function [acc, q] = cal_acc_q (q, hs)

n = length(hs);
q = [q(1); 0; q(2:end-1); 0; q(end)];
K = 2*eye(n-1);
K(1,2) = hs(2)/(hs(2)+hs(1));
K(n-1,n-2) = 1 - hs(n)/(hs(n)+hs(n-1));
B = zeros(n-1,1);
for j = 2:n-2
        K(j,j+1) = hs(j+1)/(hs(j+1)+hs(j));
        K(j,j-1) = 1 - K(j,j+1);
end
```

```
for j = 3:n-3
    B(j) = 6/( hs(j+1)+hs(j) ) * ( (q(j+2)-q(j+1))/hs(j+1) - (q(j+1)-q(j))/hs(j) );
end
a_start = 0; a_end = 0;
v_start = 0; v_end = 0;
B(1) = 6/( hs(2)+hs(1) ) * ( q(3)/hs(2) + q(1)/hs(1) ) - (1 - hs(2)/(hs(2)+hs(1)) )*a_start;
B(2) = 6/( hs(3)+hs(2) ) * ( (q(4)-q(3))/hs(3) - q(3)/hs(2) );
B(n-2) = 6/( hs(n-1)+hs(n-2) ) * ( -q(n-1)/hs(n-1) - (q(n-1)-q(n-2))/hs(n-2) );
B(n-1) = 6/( hs(n)+hs(n-1) ) * ( q(n+1)/hs(n) + q(n-1)/hs(n-1) ) - hs(n)/(hs(n)+hs(n-1))*a_end;

K = [K zeros(n-1,2); -(hs(1)^2)/6 zeros(1,n-2) 1 0; zeros(1,n-2) -(hs(n)^2)/6 0 1];
K(1,n) = 6/hs(2)/hs(1);
K(2,n) = -6/( hs(3)+hs(2) )/hs(2);
K(n-2,n+1) = -6/( hs(n-1)+hs(n-2) )/hs(n-1);
K(n-1,n+1) = 6/hs(n)/hs(n-1);
B(n) = q(1) + hs(1)*v_start + (hs(1)^2)/3*a_start;
B(n+1) = q(n+1) - hs(n)*v_end + (hs(n)^2)/3*a_end;

tmp = K\B;
acc = [a_start; tmp(1:n-1); a_end];
q(2) = tmp(n);
q(n) = tmp(n+1);
end
```

6) Sub Matlab function of calculation for Dynamics: Dynamics.m

```
function u_DLCC = DLCC_cal_Dynamics(phi,dphi,ddphi,mp)

mp = 10;
sx=10;
sz=10;
L01=400;
L1=420;
L2=50;
L3=450;
L4=180;
L5=180;
L6=80;
e=0;
Lt=225;
L_O2O3 = L2+L4+sqrt(2)*L5;
Lp = Lt - (sqrt(2)*L6+L4+e);

n0 = 1;
```

```
phi = phi(:);
phi(1:3) = phi(1:3)*2*pi/sx*1000;
phi1 = phi(1);phi2 = phi(2);phi3 = phi(3);phi4 = phi(4);phi5 = phi(5);
dphi(1:3) = dphi(1:3)*2*pi/sx*1000;
dphi1 = dphi(1);dphi2 = dphi(2);dphi3 = dphi(3);dphi4 = dphi(4);dphi5 = dphi(5);
ddphi(1:3) = ddphi(1:3)*2*pi/sx*1000;
ddphi1 = ddphi(1);ddphi2 = ddphi(2);ddphi3 = ddphi(3);ddphi4 = ddphi(4);ddphi5 = ddphi(5);

%% Outward Recursion
%Link 1
k1=2*pi*L1/sx;
a=atan((phi2-phi1)/k1);
da=k1.*(dphi2-dphi1)./(k1^2+(phi2-phi1).^2);
k2=k1^2+(phi2-phi1).^2;
dda=k1*((ddphi2-ddphi1).*k2-2*(phi2-phi1).*(dphi2-dphi1).^2)./k2.^2;
d1=zeros(3,n0);d1(3,:)=da;
dd1=zeros(3,n0);dd1(3,:)=dda;
p10=zeros(3,n0);R01=zeros(3,3*n0);R10=zeros(3,3*n0);
rc1=[0,0,-44.182]';m1=83.025;
I1=[5095170.782 -61.008 1019.065
    -61.008 3291706.841 -15.858
    1019.065 -15.858 2367692.766];
ddp1=zeros(3,n0);
nc1=zeros(3,n0);
for n=1:n0
    p10(:,n)=[sx*(phi1(n)+phi2(n))/4/pi,0,0]';
    R01(:,3*(n-1)+1:3*n)=[cos(a(n)) sin(a(n)) 0;-sin(a(n)) cos(a(n)) 0;0 0 1];
    R10(:,3*(n-1)+1:3*n)=R01(:,3*(n-1)+1:3*n)';
    ddp1(:,n)=R01(:,3*(n-1)+1:3*n)*[sx/4/pi*(ddphi2(n)-ddphi1(n));0;-9806.65];
    ddpc1(:,n)=ddp1(:,n)+[-rc1(2)*dda(n)-rc1(1)*da(n)^2;rc1(1)*dda(n)-rc1(2)*da(n)^2;0];
    nc1(:,n)=I1*dd1(:,n)+cross(d1(:,n),I1*d1(:,n));
end
fc1=m1*ddpc1;
%Link 2
d2=d1;
dd2=dd1;
p21=zeros(3,n0);R12=zeros(3,3*n0);R21=zeros(3,3*n0);
rc2=[-304.563,-1.306,-278.945]';
m2=21.333;
I2=[1475990.573 -3970.014 -748196.633
    -3970.014 2366543.574 -8327.223
    -748196.633 -8327.223 911968.216];
ddp2=zeros(3,n0);ddpc2=zeros(3,n0);nc2=zeros(3,n0);
for n=1:n0
```

```
    p21(:,n)=[L3,0,sz*phi3(n)/2/pi+L01]';
    R12(:,3*(n-1)+1:3*n)=[1 0 0
        0 1 0
        0 0 1];
    R21(:,3*(n-1)+1:3*n)=R12(:,3*(n-1)+1:3*n)';
    C=[-L3*da(n)^2,L3*dda(n),sz*ddphi3(n)/2/pi]';
    ddp2(:,n)=ddp1(:,n)+C;
    nc2(:,n)=I2*dd2(:,n)+cross(d2(:,n),I2*d2(:,n));
end
fc2=m2*ddpc2;
%Link 3
d3=zeros(3,n0);d3(2,:)=(da+dphi4)/sqrt(2);d3(3,:)=(da+dphi4)/sqrt(2);
dd3=zeros(3,n0);dd3(2,:)=(dda+ddphi4)/sqrt(2);dd3(3,:)=(dda+ddphi4)/sqrt(2);
cs_d3=d3*180/pi;
cs_dd3=dd3*180/pi;
ddp3=zeros(3,n0);ddpc3=zeros(3,n0);nc3=zeros(3,n0);
rc3=[0,-63.686,-468.138]';
p32=zeros(3,n0);R23=zeros(3,3*n0);R32=zeros(3,3*n0);
m3=8.022;
I3=[142265.977 -0.114 0.002
    -0.114 45016.139 46002.096
    0.002 46002.096 109393.601];
for n=1:n0
    p32(:,n)=[0,0,L_O2O3]';
    R23(:,3*(n-1)+1:3*n)=[sin(phi4(n)) -cos(phi4(n)) 0
        cos(phi4(n))/sqrt(2) sin(phi4(n))/sqrt(2) 1/sqrt(2)
        -cos(phi4(n))/sqrt(2) -sin(phi4(n))/sqrt(2) 1/sqrt(2)];
    R32(:,3*(n-1)+1:3*n)=R23(:,3*(n-1)+1:3*n)';
    ddp3(:,n)=R23(:,3*(n-1)+1:3*n)*ddp2(:,n);
    ddpc3(:,n)=ddp3(:,n)+cross(dd3(:,n),rc3)+cross(d3(:,n),cross(d3(:,n),rc3));
    nc3(:,n)=I3*dd3(:,n)+cross(d3(:,n),I3*d3(:,n));
end
fc3=m3*ddpc3;
%Link 4
ddp4=zeros(3,n0);ddpc4=zeros(3,n0);nc4=zeros(3,n0);
rc4=[0.055,-52.909,-261.241]';
p43=zeros(3,n0);
R34=zeros(3,3*n0);R43=zeros(3,3*n0);
m4=9.120;
I4=[119573.680 27.342 -0.849
    27.342 36160.108 -18452.465
    -0.849 -18452.465 94076.578];
for n=1:n0
    p43(:,n)=[0,0,sqrt(2)*e]';
```

```
    R34(:,3*(n-1)+1:3*n)=[-cos(phi5(n)) -sin(phi5(n)) 0
        sin(phi5(n))/sqrt(2) -cos(phi5(n))/sqrt(2) 1/sqrt(2)
        -sin(phi5(n))/sqrt(2) cos(phi5(n))/sqrt(2) 1/sqrt(2)];
    R43(:,3*(n-1)+1:3*n)=R34(:,3*(n-1)+1:3*n)';
    d4(:,n)=R34(:,3*(n-1)+1:3*n)*d3(:,n)+dphi5(n)*[0,1/sqrt(2),1/sqrt(2)]';

dd4(:,n)=R34(:,3*(n-1)+1:3*n)*dd3(:,n)+cross((R34(:,3*(n-1)+1:3*n)*d3(:,n)),(dphi5(n)*[0,1/sqr
t(2),1/sqrt(2)]'))+ddphi5(n)*[0,1/sqrt(2),1/sqrt(2)]';

ddp4(:,n)=R34(:,3*(n-1)+1:3*n)*(ddp3(:,n)+cross(dd3(:,n),p43(:,n))+cross(d3(:,n),cross(d3(:,n),p
43(:,n))));
    ddpc4(:,n)=ddp4(:,n)+cross(dd4(:,n),rc4)+cross(d4(:,n),cross(d4(:,n),rc4));
    nc4(:,n)=I4*dd4(:,n)+cross(d4(:,n),I4*d4(:,n));%unit: kg*mm*mm/s^2
end
cs_d4=d4*180/pi;
cs_dd4=dd4*180/pi;
fc4=m4*ddpc4;

%% Inward Recursion
% external forces and moments
R54=[1 0 0;0 1 0;0 0 1];
p54=[0,0,Lp]';

f5=[0;0;mp*9.8]*ones(1,n0);f5=-1000*f5;
n5=[0;0;0]*ones(1,n0);n5=-1000000*n5;

f4=zeros(3,n0);n4=f4;
f3=f4;n3=f4;
f2=f4;n2=f4;
f1=f4;n1=f4;
for n=1:n0
    f4(:,n)=fc4(:,n)+R54*f5(:,n);
    n4(:,n)=R54*n5(:,n)+nc4(:,n)+cross(rc4,fc4(:,n))+cross(p54,R54*f5(:,n));
    f3(:,n)=R43(:,3*(n-1)+1:3*n)*f4(:,n)+fc3(:,n);

n3(:,n)=R43(:,3*(n-1)+1:3*n)*n4(:,n)+nc3(:,n)+cross(rc3,fc3(:,n))+cross(p43(:,n),R43(:,3*(n-1)+
1:3*n)*f4(:,n));
    f2(:,n)=R32(:,3*(n-1)+1:3*n)*f3(:,n)+fc2(:,n);

n2(:,n)=R32(:,3*(n-1)+1:3*n)*n3(:,n)+nc2(:,n)+cross(rc2,fc2(:,n))+cross(p32(:,n),R32(:,3*(n-1)+
1:3*n)*f3(:,n));
    f1(:,n)=R21(:,3*(n-1)+1:3*n)*f2(:,n)+fc1(:,n);

n1(:,n)=R21(:,3*(n-1)+1:3*n)*n2(:,n)+nc1(:,n)+cross(rc1,fc1(:,n))+cross(p21(:,n),R21(:,3*(n-1)+
```

```
1:3*n)*f2(:,n));
        M5(n)=n4(:,n)'*[0,1/sqrt(2),1/sqrt(2)]';
        M4(n)=n3(:,n)'*[0,1/sqrt(2),1/sqrt(2)]';
        F3(n)=f2(:,n)'*[0,0,1]';
        M3(n)=F3(n)*sz/2/pi;
        C1=(R10(:,3*(n-1)+1:3*n)*n1(:,n))'*[0,0,1]'*2*cos(a(n)).^2/L1;
        C2=(R10(:,3*(n-1)+1:3*n)*f1(:,n))'*[1,0,0]';
        F1(n)=(C2-C1)/2;
        F2(n)=(C2+C1)/2;
        M1(n)=F1(n)*sx/2/pi;
        M2(n)=F2(n)*sx/2/pi;
end
%% Unit conversion
f4=f4/1000;%N
n4=n4/1000;%Nmm
f3=f3/1000;
n3=n3/1000;
f2=f2/1000;
n2=n2/1000;
f1=f1/1000;
n1=n1/1000;
%% Unit conversion
F1=F1/1000;%N
F2=F2/1000;
F3=F3/1000;
M1=M1/1000;%Nmm
M2=M2/1000;
M3=M3/1000;
M4=M4/1000;
M5=M5/1000;
u_DLCC = [F1 F2 F3 [M4 M5]/1000];    % Output:joint 1~3—N, joint 4,5--Nm
```

Appendix B: Matlab function showing how to call/use the function implementing the optimization algorithm NSGA-II

1) Main function in Matlab: trajectory_planning_calculate.m

```
clear;clc;close all;
% pop - Population size
% gen - Total number of generations
pop = 100;
gen = 100;
% result(gen,V+M+1) - [set of decision variables(x1 x2...), objective function values(f1 f2...), cost function(Fc)]
result = nsga_2(pop,gen);
x1 = result(:,1);
x2 = result(:,2);
f1 = result(:,3);
f2 = result(:,4);
Fc = result(:,5);

plot(Fc)
ylabels = {'Decision variable 1' 'Decision variable 2' 'Objective function 1' 'Objective functions 2' 'Cost function'};
for i = 1:5
    figure (i);
    plot(result(:,i),'Linewidth',1.5,'Color','b');
    grid on;
    xlabel('Generation');
    ylabel(ylabels{i});
    figure_FontSize=12;
    set(get(gca,'XLabel'),'FontSize',figure_FontSize);
    set(get(gca,'YLabel'),'FontSize',figure_FontSize);
    set(gcf,'Position',[100 100 600 350]);
    set(gca,'fontsize',14);
end
```

2) Sub Matlab function program of NSGA-II: nsga_2.m

```
function f = nsga_2(pop,gen)

%% function nsga_2(pop,gen)
% is a multi-objective optimization function where the input arguments are
% pop - Population size
% gen - Total number of generations
% f(gen,V+M+1) - [set of decision variables(x1 x2...), objective function values(f1 f2...), cost
```

```
function(Fc)]
%
% This functions is based on evolutionary algorithm for finding the optimal
% solution for multiple objective i.e. pareto front for the objectives.
% Initially enter only the population size and the stoping criteria or
% the total number of generations after which the algorithm will
% automatically stopped.
%
% You will be asked to enter the number of objective functions, the number
% of decision variables and the range space for the decision variables.
% Also you will have to define your own objective funciton by editing the
% evaluate_objective() function. A sample objective function is described
% in evaluate_objective.m. Kindly make sure that the objective function
% which you define match the number of objectives that you have entered as
% well as the number of decision variables that you have entered. The
% decision variable space is continuous for this function, but the
% objective space may or may not be continuous.
%
% Original algorithm NSGA-II was developed by researchers in Kanpur Genetic
% Algorithm Labarotary and kindly visit their website for more information
% http://www.iitk.ac.in/kangal/


%    Copyright (c) 2009, Aravind Seshadri
%    All rights reserved.
%
%    Redistribution and use in source and binary forms, with or without
%    modification, are permitted provided that the following conditions are
%    met:
%
%        * Redistributions of source code must retain the above copyright
%          notice, this list of conditions and the following disclaimer.
%        * Redistributions in binary form must reproduce the above copyright
%          notice, this list of conditions and the following disclaimer in
%          the documentation and/or other materials provided with the distribution
%
%    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
%    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
%    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
%     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
```

```
%    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
%    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
%     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
%    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
%    POSSIBILITY OF SUCH DAMAGE.

%% Simple error checking
% Number of Arguments
% Check for the number of arguments. The two input arguments are necessary
% to run this function.
if nargin < 2
    error('NSGA-II: Please enter the population size and number of generations as input arguments.');
end
% Both the input arguments need to of integer data type
if isnumeric(pop) == 0 || isnumeric(gen) == 0
    error('Both input arguments pop and gen should be integer datatype');
end
% Minimum population size has to be 20 individuals
if pop < 20
    error('Minimum population for running this function is 20');
end
if gen < 5
    error('Minimum number of generations is 5');
end
% Make sure pop and gen are integers
pop = round(pop);
gen = round(gen);
%% Objective Function
% The objective function description contains information about the
% objective function. M is the dimension of the objective space, V is the
% dimension of decision variable space, min_range and max_range are the
% range for the variables in the decision variable space. User has to
% define the objective functions using the decision variables. Make sure to
% edit the function 'evaluate_objective' to suit your needs.
[M, V, min_range, max_range] = objective_description_function();

%% Initialize the population
% Population is initialized with random values which are within the
```

```
% specified range. Each chromosome consists of the decision variables. Also
% the value of the objective functions, rank and crowding distance
% information is also added to the chromosome vector but only the elements
% of the vector which has the decision variables are operated upon to
% perform the genetic operations like corssover and mutation.
chromosome = initialize_variables(pop, M, V, min_range, max_range);

%% Sort the initialized population
% Sort the population using non-domination-sort. This returns two columns
% for each individual which are the rank and the crowding distance
% corresponding to their position in the front they belong. At this stage
% the rank and the crowding distance for each chromosome is added to the
% chromosome vector for easy of computation.
chromosome = non_domination_sort_mod(chromosome, M, V);

%% Start the evolution process
% The following are performed in each generation
% * Select the parents which are fit for reproduction
% * Perfrom crossover and Mutation operator on the selected parents
% * Perform Selection from the parents and the offsprings
% * Replace the unfit individuals with the fit individuals to maintain a
%     constant population size.
f = zeros(gen,V+M+1);
for i = 1 : gen
    % Select the parents
    % Parents are selected for reproduction to generate offspring. The
    % original NSGA-II uses a binary tournament selection based on the
    % crowded-comparision operator. The arguments are
    % pool - size of the mating pool. It is common to have this to be half the
    %             population size.
    % tour - Tournament size. Original NSGA-II uses a binary tournament
    %             selection, but to see the effect of tournament size this is kept
    %             arbitary, to be choosen by the user.
    pool = round(pop/2);
    tour = 2;
    % Selection process
    % A binary tournament selection is employed in NSGA-II. In a binary
    % tournament selection process two individuals are selected at random
    % and their fitness is compared. The individual with better fitness is
    % selcted as a parent. Tournament selection is carried out until the
    % pool size is filled. Basically a pool size is the number of parents
    % to be selected. The input arguments to the function
    % tournament_selection are chromosome, pool, tour. The function uses
    % only the information from last two elements in the chromosome vector.
```

```
% The last element has the crowding distance information while the
% penultimate element has the rank information. Selection is based on
% rank and if individuals with same rank are encountered, crowding
% distance is compared. A lower rank and higher crowding distance is
% the selection criteria.
parent_chromosome = tournament_selection(chromosome, pool, tour);

% Perfrom crossover and Mutation operator
% The original NSGA-II algorithm uses Simulated Binary Crossover (SBX) and
% Polynomial  mutation. Crossover probability pc = 0.9 and mutation
% probability is pm = 1/n, where n is the number of decision variables.
% Both real-coded GA and binary-coded GA are implemented in the original
% algorithm, while in this program only the real-coded GA is considered.
% The distribution indeices for crossover and mutation operators as mu = 20
% and mum = 20 respectively.
mu = 20;
mum = 20;
offspring_chromosome = ...
      genetic_operator(parent_chromosome, ...
      M, V, mu, mum, min_range, max_range);

% Intermediate population
% Intermediate population is the combined population of parents and
% offsprings of the current generation. The population size is two
% times the initial population.

[main_pop,temp] = size(chromosome);
[offspring_pop,temp] = size(offspring_chromosome);
% temp is a dummy variable.
clear temp
% intermediate_chromosome is a concatenation of current population and
% the offspring population.
intermediate_chromosome(1:main_pop,:) = chromosome;
intermediate_chromosome(main_pop + 1 : main_pop + offspring_pop,1 : M+V) = ...
      offspring_chromosome;

% Non-domination-sort of intermediate population
% The intermediate population is sorted again based on non-domination sort
% before the replacement operator is performed on the intermediate
% population.
intermediate_chromosome = ...
      non_domination_sort_mod(intermediate_chromosome, M, V);
% Perform Selection
% Once the intermediate population is sorted only the best solution is
```

```
    % selected based on it rank and crowding distance. Each front is filled in
    % ascending order until the addition of population size is reached. The
    % last front is included in the population based on the individuals with
    % least crowding distance
    chromosome = replace_chromosome(intermediate_chromosome, M, V, pop);
    if ~mod(i,100)
        clc
        fprintf('%d generations completed\n',i);
    end
    f_tmp = chromosome(:,1:V+M);
    % Stored the cost function at the end column of the result.
    for n=1:size(chromosome,1)
        f_tmp(n,V+M+1) = 0.4*chromosome(n,V+1)+0.6*chromosome(n,V+2);
    end
    % Select the best function**************************
    min_Nr = find(f_tmp(:,V+M+1) == min(f_tmp(:,V+M+1)));
    f(i,:) = f_tmp(min_Nr,:);
end

end
```

3) Sub Matlab function program of objective description function: objective_description_function.m

```
function [number_of_objectives, number_of_decision_variables, min_range_of_decesion_variable,
max_range_of_decesion_variable] = objective_description_function()

%% function [number_of_objectives, number_of_decision_variables, min_range_of_decesion_variable,
max_range_of_decesion_variable] = objective_description_function()
% This function is used to completely describe the objective functions and
% the range for the decision variable space etc. The user is prompted for
% inputing the number of objectives, numebr of decision variables, the
% maximum and minimum range for each decision variable and finally the
% function waits for the user to modify the evaluate_objective function to
% suit their need.

%   Copyright (c) 2009, Aravind Seshadri
%   All rights reserved.
%
%   Redistribution and use in source and binary forms, with or without
%   modification, are permitted provided that the following conditions are
%   met:
%
%       * Redistributions of source code must retain the above copyright
```

```
%            notice, this list of conditions and the following disclaimer.
%        * Redistributions in binary form must reproduce the above copyright
%            notice, this list of conditions and the following disclaimer in
%            the documentation and/or other materials provided with the distribution
%
%   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
%   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
%   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
%    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
%   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
%   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
%    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
%   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
%   POSSIBILITY OF SUCH DAMAGE.

% Obtain the number of objective function
number_of_objectives = 2;
% Obtain the number of decision variables
number_of_decision_variables = 2;
% Obtain the minimum possible value for each decision variable
min_range_of_decesion_variable = [0 0];
% Obtain the minimum possible value for each decision variable
max_range_of_decesion_variable = [10 10];
```

4) Sub Matlab function program for initializing the chromosomes: initialize_variables.m

```
function f = initialize_variables(N, M, V, min_range, max_range)

%% function f = initialize_variables(N, M, V, min_tange, max_range)
% This function initializes the chromosomes. Each chromosome has the
% following at this stage
%            * set of decision variables
%            * objective function values
%
```

```
% where,
% N - Population size
% M - Number of objective functions
% V - Number of decision variables
% min_range - A vector of decimal values which indicate the minimum value
% for each decision variable.
% max_range - Vector of maximum possible values for decision variables.

%    Copyright (c) 2009, Aravind Seshadri
%    All rights reserved.
%
%    Redistribution and use in source and binary forms, with or without
%    modification, are permitted provided that the following conditions are
%    met:
%
%        * Redistributions of source code must retain the above copyright
%            notice, this list of conditions and the following disclaimer.
%        * Redistributions in binary form must reproduce the above copyright
%            notice, this list of conditions and the following disclaimer in
%            the documentation and/or other materials provided with the distribution
%
%    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
%    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
%    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
%      ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
%    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
%    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
%      CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
%    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
%    POSSIBILITY OF SUCH DAMAGE.

min = min_range;
max = max_range;
```

```
% K is the total number of array elements. For ease of computation decision
% variables and objective functions are concatenated to form a single
% array. For crossover and mutation only the decision variables are used
% while for selection, only the objective variable are utilized.

K = M + V;

%% Initialize each chromosome
% For each chromosome perform the following (N is the population size)
for i = 1 : N
    % Initialize the decision variables based on the minimum and maximum
    % possible values. V is the number of decision variable. A random
    % number is picked between the minimum and maximum possible values for
    % the each decision variable.
    while (1)
        for j = 1 : V
            f(i,j) = min(j) + (max(j) - min(j))*rand(1);
        end
        if constraint(f(i,1:V))==0
            break;
        end
    end
    % For ease of computation and handling data the chromosome also has the
    % vlaue of the objective function concatenated at the end. The elements
    % V + 1 to K has the objective function valued.
    % The function evaluate_objective takes one chromosome at a time,
    % infact only the decision variables are passed to the function along
    % with information about the number of objective functions which are
    % processed and returns the value for the objective functions. These
    % values are now stored at the end of the chromosome itself.
    f(i,V + 1: K) = evaluate_objective(f(i,:), M, V);
end
```

5) Sub Matlab function program of objective functions: evaluate_objective.m

```
function f = evaluate_objective(x, M, V)

%% function f = evaluate_objective(x, M, V)
% Function to evaluate the objective functions for the given input vector
% x. x is an array of decision variables and f(1), f(2), etc are the
% objective functions. The algorithm always minimizes the objective
% function hence if you would like to maximize the function then multiply
% the function by negative one. M is the numebr of objective functions and
% V is the number of decision variables.
```

```
%
% This functions is basically written by the user who defines his/her own
% objective function. Make sure that the M and V matches your initial user
% input. Make sure that the
%
% An example objective function is given below. It has two six decision
% variables are two objective functions.

% f = [];
% %% Objective function one
% % Decision variables are used to form the objective function.
% f(1) = 1 - exp(-4*x(1))*(sin(6*pi*x(1)))^6;
% sum = 0;
% for i = 2 : 6
%         sum = sum + x(i)/4;
% end
% %% Intermediate function
% g_x = 1 + 9*(sum)^(0.25);
%
% %% Objective function two
% f(2) = g_x*(1 - ((f(1))/(g_x))^2);

%% Kursawe proposed by Frank Kursawe.
% Take a look at the following reference
% A variant of evolution strategies for vector optimization.
% In H. P. Schwefel and R. Männer, editors, Parallel Problem Solving from
% Nature. 1st Workshop, PPSN I, volume 496 of Lecture Notes in Computer
% Science, pages 193-197, Berlin, Germany, oct 1991. Springer-Verlag.
%
% Number of objective is two, while it can have arbirtarly many decision
% variables within the range -5 and 5. Common number of variables is 3.
% Decision variables are used to form the objective function.
a = x;
x = a(1);
y = a(2);
f1 = 1-exp(-(x-4).^2 - (y-6).^2) + 2-2*exp(-(x-3).^2 - (y-3).^2);
f2 = 1-exp(-(x-6).^2 - (y-4).^2);
f = [f1 f2];
```

6) Sub Matlab function program for sorting the current popultion: non_domination_sort_mod.m

```
function f = non_domination_sort_mod(x, M, V)

%% function f = non_domination_sort_mod(x, M, V)
```

```
% This function sort the current popultion based on non-domination. All the
% individuals in the first front are given a rank of 1, the second front
% individuals are assigned rank 2 and so on. After assigning the rank the
% crowding in each front is calculated.

%   Copyright (c) 2009, Aravind Seshadri
%   All rights reserved.
%
%   Redistribution and use in source and binary forms, with or without
%   modification, are permitted provided that the following conditions are
%   met:
%
%       * Redistributions of source code must retain the above copyright
%         notice, this list of conditions and the following disclaimer.
%       * Redistributions in binary form must reproduce the above copyright
%         notice, this list of conditions and the following disclaimer in
%         the documentation and/or other materials provided with the distribution
%
%   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
%   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
%   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
%    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
%   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
%   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
%    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
%   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
%   POSSIBILITY OF SUCH DAMAGE.

[N, m] = size(x);
clear m

% Initialize the front number to 1.
front = 1;
```

```
% There is nothing to this assignment, used only to manipulate easily in
% MATLAB.
F(front).f = [];
individual = [];

%% Non-Dominated sort.
% The initialized population is sorted based on non-domination. The fast
% sort algorithm [1] is described as below for each

% ?for each individual p in main population P do the following
%     ?Initialize Sp = []. This set would contain all the individuals that is
%       being dominated by p.
%     ?Initialize np = 0. This would be the number of individuals that domi-
%       nate p.
%     ?for each individual q in P
%           * if p dominated q then
%                 ?add q to the set Sp i.e. Sp = Sp ? {q}
%           * else if q dominates p then
%                 ?increment the domination counter for p i.e. np = np + 1
%     ?if np = 0 i.e. no individuals dominate p then p belongs to the first
%       front; Set rank of individual p to one i.e prank = 1. Update the first
%       front set by adding p to front one i.e F1 = F1 ? {p}
% ?This is carried out for all the individuals in main population P.
% ?Initialize the front counter to one. i = 1
% ?following is carried out while the ith front is nonempty i.e. Fi != []
%     ?Q = []. The set for storing the individuals for (i + 1)th front.
%     ?for each individual p in front Fi
%           * for each individual q in Sp (Sp is the set of individuals
%             dominated by p)
%               ?nq = nq?1, decrement the domination count for individual q.
%               ?if nq = 0 then none of the individuals in the subsequent
%                  fronts would dominate q. Hence set qrank = i + 1. Update
%                  the set Q with individual q i.e. Q = Q ? q.
%     ?Increment the front counter by one.
%     ?Now the set Q is the next front and hence Fi = Q.
%
% This algorithm is better than the original NSGA ([2]) since it utilize
% the informatoion about the set that an individual dominate (Sp) and
% number of individuals that dominate the individual (np).

%
for i = 1 : N
    % Number of individuals that dominate this individual
    individual(i).n = 0;
```

```
        % Individuals which this individual dominate
        individual(i).p = [];
        for j = 1 : N
            dom_less = 0;
            dom_equal = 0;
            dom_more = 0;
            for k = 1 : M
                if (x(i,V + k) < x(j,V + k))
                    dom_less = dom_less + 1;
                elseif (x(i,V + k) == x(j,V + k))
                    dom_equal = dom_equal + 1;
                else
                    dom_more = dom_more + 1;
                end
            end
            if dom_less == 0 && dom_equal ~= M
                individual(i).n = individual(i).n + 1;
            elseif dom_more == 0 && dom_equal ~= M
                individual(i).p = [individual(i).p j];
            end
        end
        if individual(i).n == 0
            x(i,M + V + 1) = 1;
            F(front).f = [F(front).f i];
        end
    end
    % Find the subsequent fronts
    while ~isempty(F(front).f)
        Q = [];
        for i = 1 : length(F(front).f)
            if ~isempty(individual(F(front).f(i)).p)
                for j = 1 : length(individual(F(front).f(i)).p)
                    individual(individual(F(front).f(i)).p(j)).n = ...
                        individual(individual(F(front).f(i)).p(j)).n - 1;
                    if individual(individual(F(front).f(i)).p(j)).n == 0
                        x(individual(F(front).f(i)).p(j),M + V + 1) = ...
                            front + 1;
                        Q = [Q individual(F(front).f(i)).p(j)];
                    end
                end
            end
        end
        front =  front + 1;
        F(front).f = Q;
```

```
end

[temp,index_of_fronts] = sort(x(:,M + V + 1));
for i = 1 : length(index_of_fronts)
    sorted_based_on_front(i,:) = x(index_of_fronts(i),:);
end
current_index = 0;

%% Crowding distance
%The crowing distance is calculated as below
% ?For each front Fi, n is the number of individuals.
%    ?initialize the distance to be zero for all the individuals i.e. Fi(dj ) = 0,
%       where j corresponds to the jth individual in front Fi.
%    ?for each objective function m
%          * Sort the individuals in front Fi based on objective m i.e. I =
%            sort(Fi,m).
%          * Assign infinite distance to boundary values for each individual
%            in Fi i.e. I(d1) = ? and I(dn) = ?
%          * for k = 2 to (n ? 1)
%               ?I(dk) = I(dk) + (I(k + 1).m ? I(k ? 1).m)/fmax(m) - fmin(m)
%               ?I(k).m is the value of the mth objective function of the kth
%                  individual in I

% Find the crowding distance for each individual in each front
for front = 1 : (length(F) - 1)
%      objective = [];
    distance = 0;
    y = [];
    previous_index = current_index + 1;
    for i = 1 : length(F(front).f)
        y(i,:) = sorted_based_on_front(current_index + i,:);
    end
    current_index = current_index + i;
    % Sort each individual based on the objective
    sorted_based_on_objective = [];
    for i = 1 : M
        [sorted_based_on_objective, index_of_objectives] = ...
            sort(y(:,V + i));
        sorted_based_on_objective = [];
        for j = 1 : length(index_of_objectives)
            sorted_based_on_objective(j,:) = y(index_of_objectives(j),:);
        end
        f_max = ...
            sorted_based_on_objective(length(index_of_objectives), V + i);
```

```
            f_min = sorted_based_on_objective(1, V + i);
            y(index_of_objectives(length(index_of_objectives)),M + V + 1 + i)...
                = Inf;
            y(index_of_objectives(1),M + V + 1 + i) = Inf;
             for j = 2 : length(index_of_objectives) - 1
                next_obj    = sorted_based_on_objective(j + 1,V + i);
                previous_obj    = sorted_based_on_objective(j - 1,V + i);
                if (f_max - f_min == 0)
                    y(index_of_objectives(j),M + V + 1 + i) = Inf;
                else
                    y(index_of_objectives(j),M + V + 1 + i) = ...
                        (next_obj - previous_obj)/(f_max - f_min);
                end
             end
        end
        distance = [];
        distance(:,1) = zeros(length(F(front).f),1);
        for i = 1 : M    %todo
            distance(:,1) = distance(:,1) + y(:,M + V + 1 + i);
        end
        y(:,M + V + 2) = distance;
        y = y(:,1 : M + V + 2);
        z(previous_index:current_index,:) = y;
    end
    f = z();

    %% References
    % [1] *Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan*, |A Fast
    % Elitist Multiobjective Genetic Algorithm: NSGA-II|, IEEE Transactions on
    % Evolutionary Computation 6 (2002), no. 2, 182 ~ 197.
    %
    % [2] *N. Srinivas and Kalyanmoy Deb*, |Multiobjective Optimization Using
    % Nondominated Sorting in Genetic Algorithms|, Evolutionary Computation 2
    % (1994), no. 3, 221 ~ 248.
```

7) Sub Matlab function program of selecting the individuals for the mating pool: tournament_selection.m

```
function f = tournament_selection(chromosome, pool_size, tour_size)

%% function tournament_selection(chromosome, pool_size, tour_size)
% is the selection policy for selecting the individuals for the mating
% pool. The selection is based on tournament selection. Argument
% |chromosome| is the current generation population from which the
```

```
% individuals are selected to form a mating pool of size |pool_size| after
% performing tournament selection, with size of the tournament being
% |tour_size|. By varying the tournament size the selection pressure can be
% adjusted. But for NSGA-II the tour_size is fixed to two, but the user may
% feel free to experiment with different tournament size. Also it has been
% observed that a tournament size of more than five has no significant
% meaning.
%
%% Tournament selection process
% In a tournament selection process n individuals are selected at random,
% where n is equal to |tour_size|. From these individuals only one is selected
% and is added to the mating pool, where size of the mating pool is
% |pool_size|. Selection is performed based on two criteria. First and
% foremost is the rank or the front in which the solutions reside.
% Individuals with lower rank are selected. Secondly if the rank of two
% individuals are the same then, the crowding distance is compared.
% Individuals with greater crowding distance is selcted.


%   Copyright (c) 2009, Aravind Seshadri
%   All rights reserved.
%
%   Redistribution and use in source and binary forms, with or without
%   modification, are permitted provided that the following conditions are
%   met:
%
%       * Redistributions of source code must retain the above copyright
%           notice, this list of conditions and the following disclaimer.
%       * Redistributions in binary form must reproduce the above copyright
%           notice, this list of conditions and the following disclaimer in
%           the documentation and/or other materials provided with the distribution
%
%   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
%   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
%   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
%    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
%   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
%   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```
%    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
%    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
%    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
%    POSSIBILITY OF SUCH DAMAGE.

% Get the size of chromosome. The number of chromosome is not important
% while the number of elements in chromosome are important.
[pop, variables] = size(chromosome);
% The peunltimate element contains the information about rank.
rank = variables - 1;
% The last element contains information about crowding distance.
distance = variables;

% Until the mating pool is filled, perform tournament selection
for i = 1 : pool_size
    % Select n individuals at random, where n = tour_size
    for j = 1 : tour_size
        % Select an individual at random
        candidate(j) = round(pop*rand(1));
        % Make sure that the array starts from one.
        if candidate(j) == 0
            candidate(j) = 1;
        end
        if j > 1
            % Make sure that same candidate is not choosen.
            while ~isempty(find(candidate(1 : j - 1) == candidate(j)))
                candidate(j) = round(pop*rand(1));
                if candidate(j) == 0
                    candidate(j) = 1;
                end
            end
        end
    end
    % Collect information about the selected candidates.
    for j = 1 : tour_size
        c_obj_rank(j) = chromosome(candidate(j),rank);
        c_obj_distance(j) = chromosome(candidate(j),distance);
    end
    % Find the candidate with the least rank
    min_candidate = ...
        find(c_obj_rank == min(c_obj_rank));
```

```
        % If more than one candiate have the least rank then find the candidate
        % within that group having the maximum crowding distance.
        if length(min_candidate) ~= 1
            max_candidate = ...
            find(c_obj_distance(min_candidate) == max(c_obj_distance(min_candidate)));
            % If a few individuals have the least rank and have maximum crowding
            % distance, select only one individual (not at random).
            if length(max_candidate) ~= 1
                max_candidate = max_candidate(1);
            end
            % Add the selected individual to the mating pool
            f(i,:) = chromosome(candidate(min_candidate(max_candidate)),:);
        else
            % Add the selected individual to the mating pool
            f(i,:) = chromosome(candidate(min_candidate(1)),:);
        end
end
```

8) Sub Matlab function program for producing offsprings: genetic_operator.m

```
function f    = genetic_operator(parent_chromosome, M, V, mu, mum, l_limit, u_limit)

%% function f    = genetic_operator(parent_chromosome, M, V, mu, mum, l_limit, u_limit)
%
% This function is utilized to produce offsprings from parent chromosomes.
% The genetic operators corssover and mutation which are carried out with
% slight modifications from the original design. For more information read
% the document enclosed.
%
% parent_chromosome - the set of selected chromosomes.
% M - number of objective functions
% V - number of decision varaiables
% mu - distribution index for crossover (read the enlcosed pdf file)
% mum - distribution index for mutation (read the enclosed pdf file)
% l_limit - a vector of lower limit for the corresponding decsion variables
% u_limit - a vector of upper limit for the corresponding decsion variables
%
% The genetic operation is performed only on the decision variables, that
% is the first V elements in the chromosome vector.

%    Copyright (c) 2009, Aravind Seshadri
%    All rights reserved.
%
%    Redistribution and use in source and binary forms, with or without
```

```
%    modification, are permitted provided that the following conditions are
%    met:
%
%        * Redistributions of source code must retain the above copyright
%          notice, this list of conditions and the following disclaimer.
%        * Redistributions in binary form must reproduce the above copyright
%          notice, this list of conditions and the following disclaimer in
%          the documentation and/or other materials provided with the distribution
%
%    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
%    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
%    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
%     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
%    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
%    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
%     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
%    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
%    POSSIBILITY OF SUCH DAMAGE.

[N,m] = size(parent_chromosome);

clear m
p = 1;
% Flags used to set if crossover and mutation were actually performed.
was_crossover = 0;
was_mutation = 0;


for i = 1 : N
      % With 90 % probability perform crossover
      if rand(1) < 0.9
            % Initialize the children to be null vector.
            child_1 = [];
            child_2 = [];
```

```
% Select the first parent
parent_1 = round(N*rand(1));
if parent_1 < 1
    parent_1 = 1;
end
% Select the second parent
parent_2 = round(N*rand(1));
if parent_2 < 1
    parent_2 = 1;
end
% Make sure both the parents are not the same.
while isequal(parent_chromosome(parent_1,:),parent_chromosome(parent_2,:))
    parent_2 = round(N*rand(1));
    if parent_2 < 1
        parent_2 = 1;
    end
end
% Get the chromosome information for each randomnly selected
% parents
parent_1 = parent_chromosome(parent_1,:);
parent_2 = parent_chromosome(parent_2,:);
% Perform corssover for each decision variable in the chromosome.
while (1)
    for j = 1 : V
        % SBX (Simulated Binary Crossover).
        % For more information about SBX refer the enclosed pdf file.
        % Generate a random number
        u(j) = rand(1);
        if u(j) <= 0.5
            bq(j) = (2*u(j))^(1/(mu+1));
        else
            bq(j) = (1/(2*(1 - u(j))))^(1/(mu+1));
        end
        % Generate the jth element of first child
        child_1(j) = ...
            0.5*(((1 + bq(j))*parent_1(j)) + (1 - bq(j))*parent_2(j));
        % Generate the jth element of second child
        child_2(j) = ...
            0.5*(((1 - bq(j))*parent_1(j)) + (1 + bq(j))*parent_2(j));
        % Make sure that the generated element is within the specified
        % decision space else set it to the appropriate extrema.
        if child_1(j) > u_limit(j)
            child_1(j) = u_limit(j);
        elseif child_1(j) < l_limit(j)
```

```
                child_1(j) = l_limit(j);
            end
            if child_2(j) > u_limit(j)
                child_2(j) = u_limit(j);
            elseif child_2(j) < l_limit(j)
                child_2(j) = l_limit(j);
            end
        end
        if constraint(child_1(1:V))==0 && constraint(child_2(1:V))==0
            break;
        end
    end
    % Evaluate the objective function for the offsprings and as before
    % concatenate the offspring chromosome with objective value.
    child_1(:,V + 1: M + V) = evaluate_objective(child_1, M, V);
    child_2(:,V + 1: M + V) = evaluate_objective(child_2, M, V);
    % Set the crossover flag. When crossover is performed two children
    % are generate, while when mutation is performed only only child is
    % generated.
    was_crossover = 1;
    was_mutation = 0;
    % With 10 % probability perform mutation. Mutation is based on
    % polynomial mutation.
else
    % Select at random the parent.
    parent_3 = round(N*rand(1));
    if parent_3 < 1
        parent_3 = 1;
    end
    % Get the chromosome information for the randomnly selected parent.
    child_3 = parent_chromosome(parent_3,:);
    % Perform mutation on eact element of the selected parent.
    while (1)
        for j = 1 : V
            r(j) = rand(1);
            if r(j) < 0.5
                delta(j) = (2*r(j))^(1/(mum+1)) - 1;
            else
                delta(j) = 1 - (2*(1 - r(j)))^(1/(mum+1));
            end
            % Generate the corresponding child element.
            child_3(j) = child_3(j) + delta(j);
            % Make sure that the generated element is within the decision
            % space.
```

```
                    if child_3(j) > u_limit(j)
                        child_3(j) = u_limit(j);
                    elseif child_3(j) < l_limit(j)
                        child_3(j) = l_limit(j);
                    end
                end
                if constraint(child_3(1:V))==0
                    break;
                end
            end
            % Evaluate the objective function for the offspring and as before
            % concatenate the offspring chromosome with objective value.
            child_3(:,V + 1: M + V) = evaluate_objective(child_3, M, V);
            % Set the mutation flag
            was_mutation = 1;
            was_crossover = 0;
        end
        % Keep proper count and appropriately fill the child variable with all
        % the generated children for the particular generation.
        if was_crossover
            child(p,:) = child_1;
            child(p+1,:) = child_2;
            was_cossover = 0;
            p = p + 2;
        elseif was_mutation
            child(p,:) = child_3(1,1 : M + V);
            was_mutation = 0;
            p = p + 1;
        end

end
f = child;
```

9) Sub Matlab function program for replacing chromosomes: replace_chromosome.m

```
function f    = replace_chromosome(intermediate_chromosome, M, V,pop)

%% function f    = replace_chromosome(intermediate_chromosome,pro,pop)
% This function replaces the chromosomes based on rank and crowding
% distance. Initially until the population size is reached each front is
% added one by one until addition of a complete front which results in
% exceeding the population size. At this point the chromosomes in that
% front is added subsequently to the population based on crowding distance.
```

```
%    Copyright (c) 2009, Aravind Seshadri
%    All rights reserved.
%
%    Redistribution and use in source and binary forms, with or without
%    modification, are permitted provided that the following conditions are
%    met:
%
%        * Redistributions of source code must retain the above copyright
%            notice, this list of conditions and the following disclaimer.
%        * Redistributions in binary form must reproduce the above copyright
%            notice, this list of conditions and the following disclaimer in
%            the documentation and/or other materials provided with the distribution
%
%    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
%    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
%    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
%      ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
%    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
%    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
%      CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
%    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
%    POSSIBILITY OF SUCH DAMAGE.

[N, m] = size(intermediate_chromosome);

% Get the index for the population sort based on the rank
[temp,index] = sort(intermediate_chromosome(:,M + V + 1));

clear temp m

% Now sort the individuals based on the index
for i = 1 : N
    sorted_chromosome(i,:) = intermediate_chromosome(index(i),:);
end
```

```
% Find the maximum rank in the current population
max_rank = max(intermediate_chromosome(:,M + V + 1));

% Start adding each front based on rank and crowing distance until the
% whole population is filled.
previous_index = 0;
for i = 1 : max_rank
    % Get the index for current rank i.e the last the last element in the
    % sorted_chromosome with rank i.
    current_index = max(find(sorted_chromosome(:,M + V + 1) == i));
    % Check to see if the population is filled if all the individuals with
    % rank i is added to the population.
    if current_index > pop
        % If so then find the number of individuals with in with current
        % rank i.
        remaining = pop - previous_index;
        % Get information about the individuals in the current rank i.
        temp_pop = ...
            sorted_chromosome(previous_index + 1 : current_index, :);
        % Sort the individuals with rank i in the descending order based on
        % the crowding distance.
        [temp_sort,temp_sort_index] = ...
            sort(temp_pop(:, M + V + 2),'descend');
        % Start filling individuals into the population in descending order
        % until the population is filled.
        for j = 1 : remaining
            f(previous_index + j,:) = temp_pop(temp_sort_index(j),:);
        end
        return;
    elseif current_index < pop
        % Add all the individuals with rank i into the population.
        f(previous_index + 1 : current_index, :) = ...
            sorted_chromosome(previous_index + 1 : current_index, :);
    else
        % Add all the individuals with rank i into the population.
        f(previous_index + 1 : current_index, :) = ...
            sorted_chromosome(previous_index + 1 : current_index, :);
        return;
    end
    % Get the index for the last added individual.
    previous_index = current_index;
end
```

10) Sub Matlab function program of constraints: constraint.m

```
function flag = constraint(x)
flag = 0;
a = x;
x = a(1);
y = a(2);
if (x/2)^2 + (y/2)^2 <9
      flag = 1;
end
end
```

Appendix C: The results of the cost function and the objective functions in fmincon Function Solver

1) Results of Cost Function (from the first to the 100 times):

3.38622190000000 2.98198970000000 3.40234360000000 2.98709310000000
3.00006430000000 2.98589920000000 3.46268830000000 3.08440110000000
3.64498970000000 3.06250800000000 3.46078910000000 3.24724870000000
2.98298720000000 2.99334180000000 2.98263440000000 3.01908790000000
3.45319380000000 5.29874010000000 2.97923200000000 3.00401460000000
3.45667750000000 3.04460320000000 3.00564390000000 3.49880880000000
3.00442800000000 3.40121100000000 3.52147700000000 3.00348980000000
3.09945330000000 2.95860460000000 2.97963310000000 3.00047940000000
3.43408350000000 3.47701410000000 3.44346740000000 3.40361510000000
3.40864200000000 3.41753910000000 2.98014850000000 2.99909690000000
3.43175350000000 3.00330740000000 2.97784890000000 3.06455010000000
2.97444940000000 3.39054940000000 3.45600670000000 2.97542590000000
2.99445010000000 2.98698540000000 2.99585060000000 2.98529950000000
8.72795590000000 3.05156530000000 3.11275100000000 3.05724190000000
3.39933590000000 3.45622750000000 2.98703700000000 2.96681330000000
2.95981300000000 3.10029400000000 2.97823120000000 3.00325340000000
3.50908540000000 2.95860460000000 2.97392450000000 2.98524890000000
2.95966830000000 2.99769320000000 2.98158650000000 2.98274980000000
3.08924240000000 3.53619780000000 3.04716210000000 3.04668820000000
2.98815800000000 3.02283780000000 3.09726150000000 2.99566290000000
3.05473650000000 2.99552230000000 3.48567010000000 3.40073450000000
2.99584970000000 2.95860460000000 3.33175360000000 3.07010840000000
2.95979120000000 3.06811330000000 2.99065310000000 2.99163290000000
3.34135270000000 3.48436490000000 2.98736380000000 3.50578180000000
3.10769260000000 3.00458860000000 3.06938060000000 2.99633190000000

2) Results of Total traveling time (z_1) (from the first to the 100 times):

8.57519820000000 6.02786480000000 8.71117020000000 6.15237520000000
6.15630620000000 6.01159980000000 9.01627200000000 6.67849530000000
9.51245620000000 6.60340940000000 9.00538920000000 5.50061820000000
6.02871970000000 5.78059830000000 6.11783720000000 6.19129390000000
8.92896220000000 10.5184940000000 5.99989920000000 5.80667240000000
8.90570820000000 6.43104810000000 6.15939510000000 9.05339380000000
6.25876030000000 8.75217200000000 9.28442620000000 6.27280060000000

5.10371440000000 5.58382330000000 5.99368470000000 5.90480450000000
8.86139120000000 8.82114360000000 8.73139520000000 8.71084360000000
8.71448740000000 8.75731520000000 5.82372770000000 6.21430600000000
8.79183350000000 6.32899420000000 6.07554760000000 6.56828310000000
6.01283380000000 8.67749210000000 8.75956040000000 5.88629940000000
6.17717460000000 6.05765920000000 6.13402530000000 6.11397660000000
27.6083810000000 6.56895920000000 6.45443750000000 6.34436340000000
8.66515620000000 8.88873940000000 6.16257530000000 5.83882320000000
5.68229030000000 6.62572310000000 6.00484440000000 6.20025090000000
9.11540530000000 5.58382910000000 5.98095510000000 6.09023070000000
5.62249300000000 6.22775540000000 5.79944020000000 5.95261570000000
6.63852520000000 9.28079790000000 5.59365120000000 6.55666690000000
6.15412700000000 5.74715120000000 6.51167710000000 6.14653620000000
6.32201440000000 5.70960900000000 8.93182330000000 8.71125800000000
6.20161000000000 5.58382350000000 8.04691840000000 6.14441780000000
5.68204570000000 6.60818190000000 5.80978050000000 6.14157210000000
8.09942500000000 8.94726620000000 6.17988570000000 9.00933110000000
6.81623710000000 5.58456800000000 6.64279200000000 6.17990890000000

3) Results of Total energy involved in the motion (z_2) (from the first to the 100 times):

2581.66100000000 3536.44800000000 2512.53260000000 3443.33380000000
3535.63760000000 3636.32100000000 2416.16490000000 3217.11760000000
2443.23110000000 3286.92460000000 2423.96060000000 3873.92620000000
3502.71430000000 3736.33110000000 3470.83990000000 3527.35930000000
2467.22220000000 5967.12380000000 3585.34000000000 3680.05460000000
2497.29580000000 3396.08700000000 3505.42300000000 2497.97330000000
3439.95580000000 2470.05770000000 2341.46050000000 3387.73030000000
4152.74290000000 3839.15080000000 3553.50770000000 3655.27160000000
2468.24970000000 2607.94510000000 2564.94230000000 2518.98230000000
2544.39660000000 2512.64040000000 3631.80960000000 3431.50150000000
2503.98410000000 3342.82780000000 3499.61790000000 3295.73760000000
3545.45050000000 2505.42920000000 2587.04520000000 3645.37930000000
3480.05020000000 3498.57900000000 3531.42610000000 3508.24240000000
1471.20680000000 3271.77730000000 3362.80080000000 3413.80050000000
2542.81090000000 2527.88720000000 3457.72110000000 3644.29500000000
3770.44750000000 3407.19060000000 3595.43050000000 3452.58200000000
2486.71180000000 3839.14630000000 3551.18730000000 3534.26750000000

3806.52640000000 3443.67760000000 3681.28110000000 3592.88340000000
3282.39710000000 2370.56710000000 3938.56110000000 3293.69010000000
3446.85990000000 3687.58570000000 3337.44800000000 3518.33710000000
3466.31700000000 3729.43730000000 2539.34710000000 2505.70780000000
3445.87330000000 3839.15040000000 2873.04170000000 3740.34050000000
3763.89510000000 3268.79200000000 3629.85300000000 3496.05110000000
2801.91540000000 2494.81660000000 3459.51810000000 2592.64530000000
3261.10980000000 3842.65310000000 3254.75760000000 3484.52200000000

4) Results of Integral of joint jerks squared (z_3) (from the first to the 100 times):

0.660388510000000 2.87838380000000 0.626845140000000 2.72390220000000
2.09136590000000 2.11826560000000 0.577724120000000 2.97462610000000
0.918597940000000 2.28381910000000 0.529364880000000 14.0522130000000
3.14764470000000 3.41061060000000 2.58664750000000 2.59672690000000
0.581890000000000 11.3104290000000 2.43244200000000 4.16439870000000
0.615851030000000 2.30697770000000 2.60158800000000 0.554158420000000
2.22310170000000 0.618070660000000 0.605983130000000 2.65281410000000
9.55233470000000 3.44654110000000 2.78186730000000 3.38131320000000
0.627726700000000 0.890453020000000 1.03995360000000 0.623445340000000
0.532330470000000 0.651714200000000 3.89641370000000 2.51433110000000
0.783894090000000 2.54741570000000 2.59992920000000 2.57564850000000
2.66093630000000 0.634900720000000 0.986502580000000 2.78781640000000
2.38420510000000 3.19934340000000 2.32100120000000 2.42201500000000
0.0140525930000000 2.46369240000000 4.73632970000000 3.33410170000000
0.667706340000000 0.531259710000000 2.40068520000000 3.11891220000000
3.20255060000000 2.10126080000000 2.32415850000000 2.66146600000000
0.507582950000000 3.44652750000000 2.88381490000000 2.35601070000000
3.40899210000000 2.33500990000000 3.52272780000000 2.86613100000000
2.79920450000000 0.735145590000000 4.89178700000000 2.08340360000000
2.52487530000000 5.26140630000000 3.70519650000000 2.26795470000000
3.11942140000000 4.39881720000000 0.790060510000000 0.637315920000000
2.35817490000000 3.44654220000000 1.05334070000000 2.65389380000000
3.25040470000000 2.55243470000000 4.16289810000000 2.43457520000000
1.49798610000000 0.972263100000000 2.30822670000000 0.374500670000000
1.95110900000000 4.53312240000000 2.42199080000000 2.19553290000000

5) Results of Integral of joint accelerations squared (z_4) (from the first to the 100 times):

0.129781820000000	0.275640890000000	0.113479550000000	0.269511800000000
0.253269070000000	0.245788240000000	0.107013760000000	
0.281123750000000	0.199557720000000	0.248657170000000	
0.106984260000000	0.769205530000000	0.303020520000000	
0.350253740000000	0.271491550000000	0.257786780000000	
0.113503390000000	0.634231680000000	0.275056790000000	
0.373542710000000	0.117296150000000	0.251615530000000	
0.270833810000000	0.111577620000000	0.251755610000000	
0.110903650000000	0.107956770000000	0.261371420000000	
0.657347050000000	0.313907580000000	0.299189960000000	
0.324151690000000	0.113183430000000	0.152392280000000	
0.168835430000000	0.110992250000000	0.101651030000000	
0.117590890000000	0.327795080000000	0.275339990000000	
0.136651560000000	0.254062400000000	0.266503430000000	
0.269154640000000	0.268726980000000	0.114874890000000	
0.161474690000000	0.300829680000000	0.247992540000000	
0.280635580000000	0.248975710000000	0.250967310000000	
0.0189933920000000	0.250352560000000	0.364264660000000	
0.315553620000000	0.117958100000000	0.103071560000000	
0.264671970000000	0.297498160000000	0.299703320000000	
0.241973260000000	0.258327830000000	0.270870980000000	
0.0914931110000000	0.313906910000000	0.283027130000000	
0.253896450000000	0.313913410000000	0.247815430000000	
0.334577610000000	0.308887190000000	0.284908180000000	
0.130427090000000	0.448313230000000	0.249564890000000	
0.286821940000000	0.435943270000000	0.342100240000000	
0.254208890000000	0.279243550000000	0.379160210000000	
0.142588810000000	0.114491990000000	0.272206030000000	
0.313907620000000	0.173494060000000	0.258015300000000	
0.305004070000000	0.269862290000000	0.377525580000000	
0.258272260000000	0.204954330000000	0.161473970000000	
0.246890850000000	0.0884944580000000	0.227331990000000	
0.428799180000000	0.258379250000000	0.265459920000000	