Table of Contents

PSO With Variable Accelerator Coefficients


```
% Author: Dr. Davide Conte
% Contact Info: davide.conte90@gmail.com
% Co-Author: Vittorio Baraldi
% Contact: baraldiv@my.erau.edu
```


Clearing memory, screen, etc.

```
close all; clear all; clc;

global P J JBest PBest GG N_particles N_elements V BLv BUv BLp BUp
global N_iterations GBest lambda
% useful if making any of the following parts into functions

%-----------------------------------------------------------------------
%
mu_m=42828.375214;              %mars' gravitational parameter [km^3/
s^2]
mu_p=7.113588120963051e-4;      %phobos' gravitational parameter
 [km^3/s^2]
R=9376;                         %average distance from m1 and m2 [km]
lambda=mu_p/(mu_m+mu_p);        %mass parameter

%Given DRO for AX=100km
Vy=-0.045620256764708;
T=2.731044880670166e4;
LU=R;
TU=sqrt(R^3/(mu_m+mu_p));

%find DRO for following Ax
Ax=125;
%nondimensional parameters
AX=Ax/LU;
T1=T/TU;
VY=Vy*TU/LU;
%-----------------------------------------------------------------------
%
```

Basic PSO initialization

```
N_particles = 90;
% [ ] number of particles to initiate

N_elements = 2;

N_iterations = 800;
% [ ] number of maximum iterations before PSO is stopped
```

Particle initialization, particle "position" and "velocity"

```
BLp(1,1) = 1.5*VY; BUp(1,1) = VY;
BLp(2,1) = T1; BUp(2,1) = 2*pi;
% [ ] set lower and upper bounds on unknowns (particle elements)

% create particles using uniform distribution according to BLp and BUp
for k = 1:N_particles
    for j=1:N_elements
        P(k,j)=unifrnd(BLp(j,1),BUp(j,1));
        % could also use the built-in function "unifrnd"
    end
end

PBest = zeros(N_particles,N_elements);
% [ ] initialize the "best" for each particle to all zeros (arbitrary)

GGstar = zeros(1,N_iterations);
% [ ] initialize the vector of best J's for all iterations

V = zeros(N_particles, N_elements);
% [ ] initialize particle velocity for each particle

BUv = BUp-BLp; BLv = -BUv;
% [ ] determine velocity bounds from position

c_I = (1 + rand)/2;
c_C = 1.49445*rand;
c_S = 1.49445*rand;
% [ ] acceleration coefficients (vary with application)
```

Cost function

```
J = zeros(N_particles);
for kk = 1:N_particles
    JBest(kk) = inf;
end
GG = inf;
% [ ] initialization of cost function (J), best overall particle
 (JBest),
```

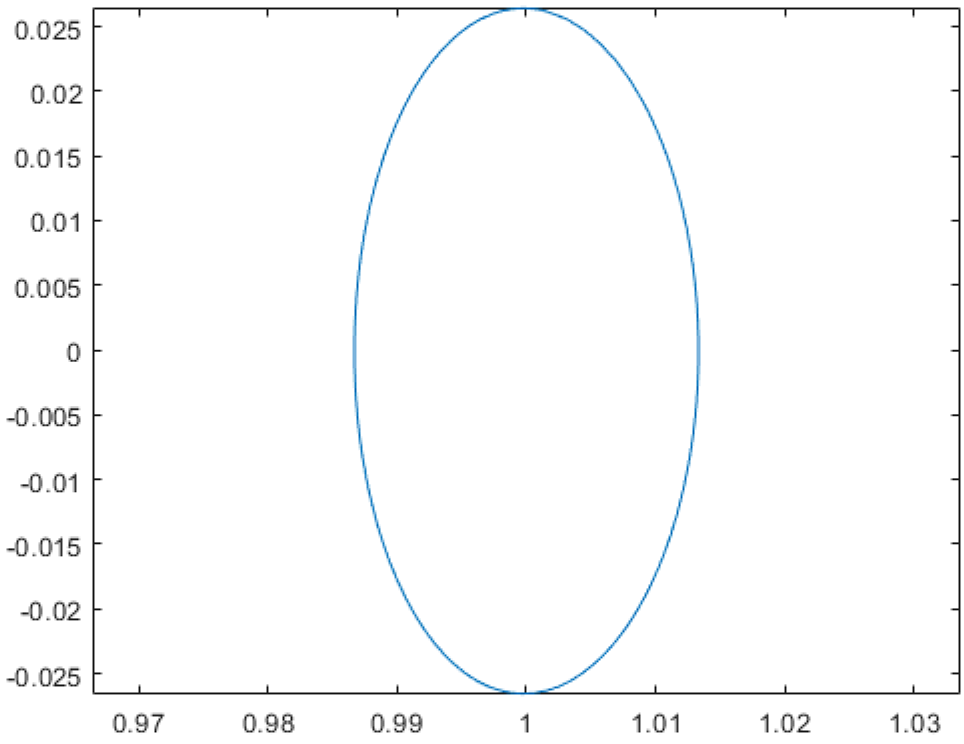
```
% and overall best cost function obtained (GG)
% NB: change "inf" to "-inf" if maximizing J instead of minimizing
```

Main PSO computations

```
tic;
% keep track of time it takes to run PSO computations (use profiler to
 see
% where improvements can be made

%initializing r0 vector
r0=[AX+(1-lambda);0];
for j = 1:N_iterations


  %-----------------------------------------------------------------------
%
    % Evaluate cost function J for each particle in current iteration
    for i = 1:N_particles

    %integrating the CR3BP EOMs
    v0=[0;P(i,1)];
    y0=[r0;v0];
    timespan=[0 P(i,2)];
    options=odeset('RelTol',1e-12,'AbsTol',1e-12);
    [~,yf]=ode113(@orbitfunCR3BP,timespan,y0,options);
    pos=[yf(end,1);yf(end,2)];
    vel=[yf(end,3);yf(end,4)];
    %evaluating cost function
    J(i)= norm(pos-r0)+norm(vel-v0);
    end

  %-----------------------------------------------------------------------
%

    % Evaluate PBest and GBest (best particle and best cost)
    for i = 1:N_particles
        if J(i) < JBest(i)
            PBest(i,:) = P(i,:);
            JBest(i) = J(i);
        end
    end

    for i = 1:N_particles
        if J(i) < GG
           GG = J(i);
           GBest = P(i,:);
        end
    end


    % Update particle velocity
    for i = 1:N_particles
```

```
        V(i,:) = c_I*V(i,:) + c_C*(PBest(i,:) - P(i,:)) + ...
            c_S*(GBest - P(i,:));
        % new particle velocity

        % check if boundaries are being respected
        for k = 1:N_elements
            if V(i,k) < BLv(k)
                V(i,k) = BLv(k);
            end
            if V(i,k) > BUv(k)
                V(i,k) = BUv(k);
            end
        end
    end


 % update particle position for each particle
    for i = 1:N_particles

        P(i,:) = P(i,:) + V(i,:);
        % new particle position

        % check position boundaries
        for k = 1:N_elements
            if P(i,k) < BLp(k)
                P(i,k) = BLp(k);
                V(i,k) = 0;
            end
            if P(i,k) > BUp(k)
                P(i,k) = BUp(k);
                V(i,k) = 0;
            end
        end
    end

    GGstar(j) = GG;

    % give progress updates to the screen every 10%
    if mod(j/N_iterations*100,10) == 0
        clc; fprintf('Progress: %2.0f %%\n',j/N_iterations*100)
    end

end

plot(yf(:,1),yf(:,2))
axis equal
PSOtime = toc;
% [s] time it takes to run PSO

time = datestr(now, 'yyyy_mm_dd');
filename = sprintf('PSO_%s.mat',time);
save( fullfile('', filename) )
% save workspace with today's date
```

```
Progress: 10 %
Progress: 20 %
Progress: 30 %
Progress: 40 %
Progress: 50 %
Progress: 60 %
Progress: 70 %
Progress: 80 %
Progress: 90 %
Progress: 100 %
```

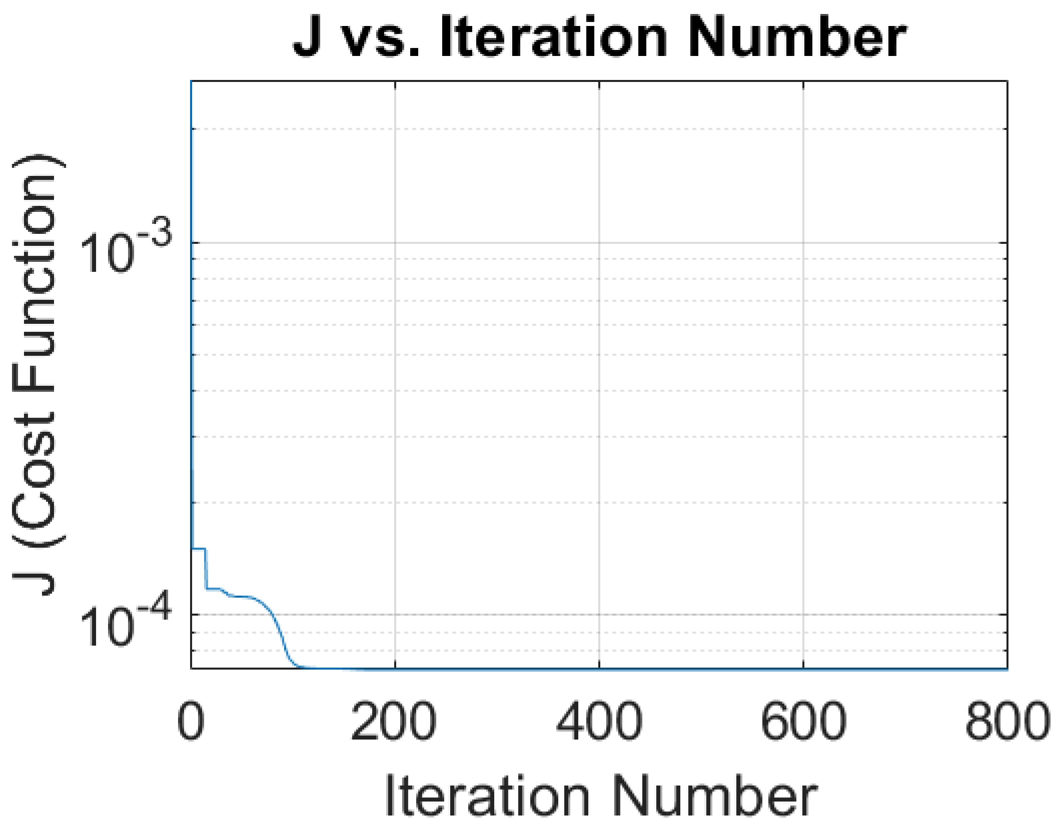


Final Results

```
fprintf('\nThe particle with best J is: \n'); disp(GBest)
fprintf('\nAnd its corresponding best J is: \n'); disp(GG)
fprintf('The y component of the velocity for Ax = %i is:\n%1.8f km/s
\n',Ax,GBest(1,1)*LU/TU)
fprintf('The orbital period for Ax = %i is:\n%1.8f s
\n',Ax,GBest(1,2)*TU)
% output run time to the screen
if PSOtime > 3600
    fprintf('\nThe PSO algorithm took %2.2f hours to complete
\n',PSOtime)
elseif PSOtime > 60
    fprintf('\nThe PSO algorithm took %2.2f minutes to complete
\n',PSOtime)
```

```
else
    fprintf('\nThe PSO algorithm took %2.4f seconds to complete
\n',PSOtime)
end

%figure of J vs. iteration number (semilog plot if J > 0 for all J's)
if GGstar(end) > 0
    figure; semilogy(1:N_iterations, GGstar)
    title('J vs. Iteration Number')
    xlabel('Iteration Number'); ylabel('J (Cost Function)');
    grid on; set(gca,'Fontsize',20);
else
    figure; plot(1:N_iterations, GGstar)
    title('J vs. Iteration Number')
    xlabel('Iteration Number'); ylabel('J (Cost Function)');
    grid on; set(gca,'Fontsize',20);
end


% script ends here
```

```
The particle with best J is:
   -0.0266    6.2477


And its corresponding best J is:
   7.1492e-05

The y component of the velocity for Ax = 125 is:
-0.05688311 km/s
The orbital period for Ax = 125 is:
27408.20516624 s

The PSO algorithm took 513.24 minutes to complete
```



Published with MATLAB® R2020a