Problem 1

Minimize the function

$$f(x_1, x_2) = (x_2 - x_1)^4 + 12x_1x_2 - x_1 + x_2 - 3$$

using

- The PSO algorithm
- The canonical genetic algorithm

(a)

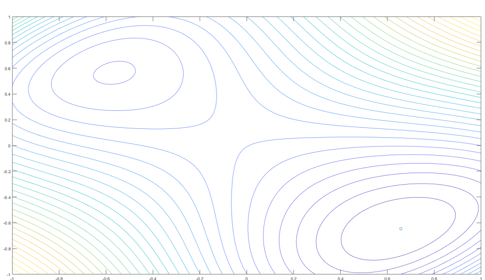
The PSO algorithm is initialized using a random population with random positions and velocities. The gbest version of the PSO algorithm is used to calculate new positions and velocities of the swarm particles while keeping a track of their pbest and gbest as follows. The inertial constant w is chosen to be slightly less than 1 (0.9 in this case) whereas the cognitive and social components c1 and c2 are chosen to be 1.9 in this case.
The MATLAB code for the PSO algorithm is attached below.

```
clc; close all;
%%Init
syms x1 x2;
w = 0.9;    %inertial coefficient
c1 = 1.9;   %cognitive coefficient
c2 = 1.9;   %social coeffiecient
numPos = 100;   %number of positions
epochs = 50;    %number of iterations
%define objective function
f = @(x1, x2) ((x2-x1).^4 + 12.*x1.*x2 - x1 + x2 - 3);
%empty cell arrays to store x_positions, p_positions, velocities and
%function evaluations and gbest, worst, average values
X_pos = cell(epochs,1);
p_pos = cell(epochs,1);
vel = cell(epochs,1);
fvalx = cell(epochs,1);
gBest = cell(epochs,1);
bestVal = zeros(epochs,1);
worsVal = zeros(epochs,1);
meanVal = zeros(epochs,1);
X_pos{1} = rand(numPos,2)*2.0-1;    %Initialize random positions of points
vel{1} = rand(numPos,2);    %initialize random velocities
%initialize random r and s
r = rand(numPos,2);
s = rand(numPos,2);
fvalx{1} = f(X_pos{1}(:,1),X_pos{1}(:,2));  %evaluate objfunc values
p_pos{1} = X_pos{1};    %set p_pos
[~,ind] = min(fvalx{1});    %get minimum func evaluation to set gbest
gBest{1} = X_pos{1}(ind,:);
bestVal(1) = f(gBest{1}(:,1), gBest{1}(:,2));
worsVal(1) = max(fvalx{1});
meanVal(1) = mean(fvalx{1});
%% Run for k iterations
for k = 1:epochs-1
    vel{k+1} = (w.*vel{k})+(c1.*r.*(p_pos{k}-X_pos{k}))...
        +(c2.*s.*(gBest{k}-X_pos{k}));  %update velocities
    X_pos{k+1} = X_pos{k}+vel{k+1}; %update positions
    fvalx{k+1} = f(X_pos{k+1}(:,1),X_pos{k+1}(:,2));    %evaluate function
    p_pos{k+1} = p_pos{k};  %update pbest
    for i = 1:numPos
        if fvalx{k+1}(i)<f(p_pos{k}(i,1),p_pos{k}(i,2))
            p_pos{k+1}(i,:) = X_pos{k+1}(i,:);  %update pbest
```

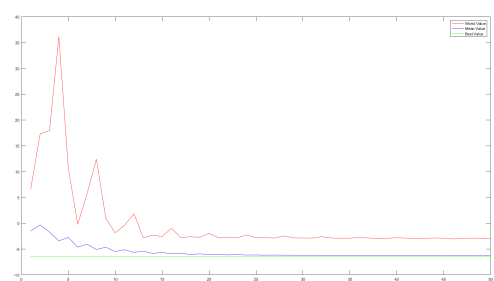
```
        end
    end
    [~,ind] = min(fvalx{k+1});  %min func eval for gbest
    gBest{k+1} = X_pos{k+1}(ind,:);
    bestVal(k+1) = f(gBest{k+1}(:,1), gBest{1}(:,2));
    worsVal(k+1) = max(fvalx{k+1});
    meanVal(k+1) = mean(fvalx{k+1});
end
%% Plotting
X = linspace(-1,1,1000);
Y = X;
[Xpt, Ypt] = meshgrid(X,Y);
Z = f(Xpt,Ypt);
figure;
contour(Xpt,Ypt, Z, 30); hold on;
plot(gBest{end}(1),gBest{end}(2), 'o');
figure;
xax = 1:epochs; %x axis
% worst, best and average value of the functions at every point
plot(xax, worsVal, 'r-', xax, meanVal, 'b-', xax, bestVal, 'g-');
legend('Worst Value', 'Mean Value', 'Best Value');

%% Scatter plot every iteration to visualize the convergence
% for i = 1:length(X_pos)
% figure;
% scatter(X_pos{i}(:,1), X_pos{i}(:,2));
% xlim([-1 1]); ylim([-1 1]);
% name = horzcat('it',num2str(i), '.png');
% saveas(gcf, name);
% end
```

Code 1: PSO implementation in MATLAB



(a) gbest after 50 epochs



(b) Plot of worst, mean and best values

Figure 1: PSO algorithm - results

(b)

For genetic algorithm the standard form asks to maximize the objective function. In our case, we have to find the minimizer of the function. We can flip the objective function and find the maxima using a Canonical Genetic algorithm and that point will be the minimizer of the negative of that function. The negative of the maximum function value can be taken as the minimum function value at that minima.
The crossover rate is selected to be 0.75 and mutation rate as 0.01. One point crossover is performed by choosing a pair from the mating pool and crossover is applied to that pair.
The MATLAB implementation of the algorithm is as given below.

```
clc; close all;
%% Init
generation_n = 100;
popuSize = 100;
xover_rate = 0.75;
mutate_rate = 0.01;
%standard form of canonical GA does maximization hence '-' sign to the
%objective function.
obj_func = @(x1, x2) -((x2-x1).^4 + 12.*x1.*x2 - x1 + x2 - 3);
var_n = 2;
range = [-1 1];
init_popu = rand(popuSize,2)*2.0-1; %initialize random population
%convert the random initialization into encoded binary string
r = 10^-4;  %resolution
%calculate the number of bits required to represent the float in a binary string
li = ceil(log2((max(init_popu)-min(init_popu)+r)/r));
% li = ceil(log2((1-(-1)+r)/r));
bit_n = sum(li);    %calculate total number of bits
n = ceil(bit_n/8);   % number bits for integer part of your number
m = bit_n-n;         % number bits for fraction part of your number
% binary number conversion for the intitial population
popu = [fix(rem(init_popu(:,1).*pow2(-(n-1):m),2)),...
    fix(rem(init_popu(:,2).*pow2(-(n-1):m),2))];
% popu = rand(popuSize, bit_n * var_n)>0.5;
upper = zeros(generation_n, 1);
average = zeros(generation_n, 1);
lower = zeros(generation_n, 1);
%% Run for n generations.
for i = 1:generation_n
    func_value = evalPopu(popu, bit_n, range, obj_func);
%     fill obj func matrices
    upper(i) = max(func_value);
    average(i) = mean(func_value);
    lower(i) = min(func_value);
    %calculate the next population
    popu = nextPopu(popu, func_value, xover_rate, mutate_rate);
end
% Convert Final population binary encoded to float numbers
final_popu = zeros(popuSize,2);
for i = 1:popuSize
    final_popu(i,1) = bit2num(popu(i,1:bit_n), range);
    final_popu(i,2) = bit2num(popu(i,bit_n+1:end), range);
end
[~,index] = max(obj_func(final_popu(:,1), final_popu(:,2)));
%% plotting
epochs = 1:popuSize;
X = linspace(-1,1,1000);
Y = X;
[Xpt, Ypt] = meshgrid(X,Y);
Z = obj_func(Xpt,Ypt);
figure;
contour(Xpt,Ypt, Z, 30); hold on;
plot(final_popu(index,1), final_popu(index,2), 'x');
figure;
plot(epochs, upper, 'r', epochs, average, 'b', epochs, lower, 'g');
legend('best','average','worst');
```

Code 2: CGA implementation in MATLAB

```
function [num] = bit2num(bit, range)
%BIT2NUM convert the binary encoded bits to actual numbers
%   Detailed explanation goes here
    integer = polyval(bit, 2);
    num = range(1) + integer*(range(2)-range(1))/(2^length(bit)-1);
```

```
end
```

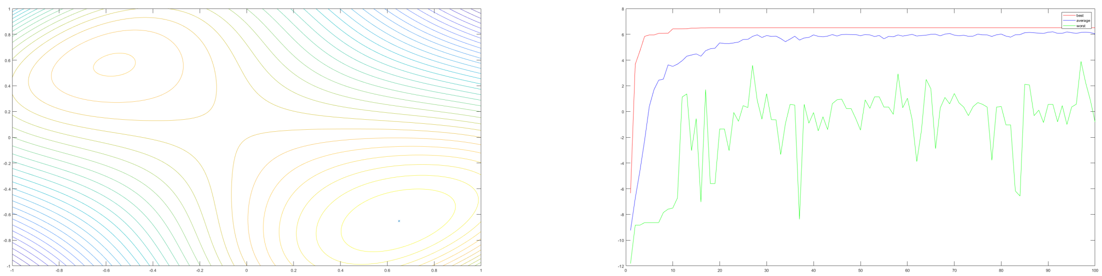
Code 3: bit2num conversion function

```
function [fitness] = evalPopu(popu,bit_n,range,obj_func)
%EVALPOPU Evaluates fitness function
%   Split the binary encoded string into two parts and then evaluates the
%   fitness at that points.
    [popu_s, string_length] = size(popu);
    fitness = zeros(popu_s,1);
    for i = 1:popu_s
        num1 = bit2num(popu(i,1:bit_n), range);
        num2 = bit2num(popu(i,bit_n+1:string_length), range);
        fitness(i) = obj_func(num1, num2);
    end
end
```

Code 4: Evaluate fitness function

```
function [new_popu] = nextPopu(popu, fitness, x_over,mut_rate)
%NEXTPOPU Generate the next generation population
%   Use the roulette wheel algorithm for selection and then does crossover
%   and mutation with the supplied probabilities to the function.
    new_popu = popu;
    [popu_s, string_length] = size(popu);
    %rescaling the fitness
    fitness = fitness-min(fitness);
    %find the best two and keep them
    tmp_fitness = fitness;
    [~, index1] = max(tmp_fitness); %find best
    tmp_fitness(index1) = min(tmp_fitness);
    [~, index2] = max(tmp_fitness); %find 2nd best
    new_popu([1 2], :) = popu([index1 index2],:);
    %for roulette wheel algo
    total = sum(fitness);
    if total == 0
        fitness = ones(popu_s, 1)/popu_s; %sum of probs is 1
    end
    cumprob = cumsum(fitness)/total;
    %selection and crossover;
    for i = 2:popu_s/2
        tmp = find(cumprob - rand>0);
        parent1 = popu(tmp(1), :);
        tmp = find(cumprob - rand>0);
        parent2 = popu(tmp(1), :);
        %do crossover
        if rand<x_over
            xover_pt = ceil(rand*(string_length-1));
            new_popu(i*2-1,:) = [parent1(1:xover_pt) parent2(xover_pt+1:string_length)];
            new_popu(i*2,:) = [parent2(1:xover_pt) parent1(xover_pt+1:string_length)];
        end
    end
    %mutation
    mask = rand(popu_s, string_length)<mut_rate;
    new_popu = xor(new_popu,mask);
    %restore elites
    new_popu([1 2], :) = popu([index1 index2],:);
end
```

Code 5: Evaluate new population



(a) minima after 100 epochs

(b) Plot of worst, mean and best values

Figure 2: CGA algorithm - results

Problem 2

The problem is to maximize total power transferred to the batteries. We can formulate the total power transferred to the batteries as the sum of the power transferred to each battery which can be given by $10I_2 + 6I_4 + 20I_5$ which can be the objective function in our case.
Thus we can formulate the problem as a linear program as follows

$$
\begin{aligned}
\text{maximize } & 10I_2 + 6I_4 + 20I_5 \\
\text{subject to } & I_1 = I_2 + I_3 \\
& I_3 = I_4 + I_5 \\
& I_1 \leq 4 \\
& I_2 \leq 3 \\
& I_3 \leq 3 \\
& I_4 \leq 2 \\
& I_5 \leq 2 \\
& I_1, I_2, I_3, I_4, I_5 \geq 0
\end{aligned}
$$

We can use MATLAB linprog function to solve the problem by taking it to a standard form as follows. We can treat the inequalities as lower and upper bounds on the individual currents.
We have to give the input to the linprog function as a function to minimize hence we have to negate the original function. Thus we supply a negative function vector f.
We supply, empty A, b matrices as we've used the inequalities as upper bounds, lower bounds.
We supply Aeq and beq (Equality constraints by taking all the variables to one side of the equality.

```
clc; close all;
f = [0 -10 0 -6 -20];
A = [];
b = [];
Aeq = [1 -1 -1 0 0; 0 0 1 -1 -1];
beq = [0;0];
lb = [0 0 0 0 0];
ub = [4 3 3 2 2];
[x,fval] = linprog(f, A, b, Aeq, beq, lb, ub);
for i = 1:length(x)
    fprintf('I%d = %d\n', i, x(i));
end
fprintf('The maximum value of function is %d\n', -fval);
```

Code 6: Solving the Linear problem using linprog

On solving the linear program we get the following answer.
Optimal solution found.
I1 = 4
I2 = 2
I3 = 2
I4 = 0
I5 = 2
The maximum value of function is 60.

Problem 3

The dual of the problem in the previous problem can be found as follows.
First we change the primal problem to its standard form

$$\begin{aligned} \text{minimize } & c^T x \\ \text{subject to } & Ax = b \\ & x \geq 0 \end{aligned}$$

The Asymmetric dual of this standard form can be given by

$$\begin{aligned} \text{maximize } & \lambda^T b \\ \text{subject to } & \lambda^T A \leq c^T \end{aligned}$$

Here, we have the given problem as

$$\begin{aligned} \text{maximize } & 10I_2 + 6I_4 + 20I_5 \\ \text{subject to } & I_1 = I_2 + I_3 \\ & I_3 = I_4 + I_5 \\ & I_1 \leq 4 \\ & I_2 \leq 3 \\ & I_3 \leq 3 \\ & I_4 \leq 2 \\ & I_5 \leq 2 \\ & I_1, I_2, I_3, I_4, I_5 \geq 0 \end{aligned}$$

We convert it to a standard form by changing the objective function to minimize and by introducing slack variables, thus converting the inequalities as follows.

$$\begin{aligned} \text{minimize } & -10I_2 - 6I_4 - 20I_5 \\ \text{subject to } & I_1 - I_2 - I_3 = 0 \\ & I_3 - I_4 - I_5 = 0 \\ & I_1 + I_6 = 4 \\ & I_2 + I_7 = 3 \\ & I_3 + I_8 = 3 \\ & I_4 + I_9 = 2 \\ & I_5 + I_{10} = 2 \\ & I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, I_{10} \geq 0 \end{aligned}$$

To convert to the dual problem we use the constraints as the objective function and then use the objective function as the constraints as shown above.

$$
\begin{aligned}
\text{maximize } & 4\lambda_3 + 3\lambda_4 + 3\lambda_5 + 2\lambda_6 + 2\lambda_7 \\
\text{subject to } & \lambda_1 + \lambda_3 = 0 \\
& -\lambda_1 + \lambda_4 = -10 \\
& -\lambda_1 + \lambda_2 + \lambda_5 = 0 \\
& -\lambda_2 + \lambda_6 = -6 \\
& -\lambda_2 + \lambda_7 = -20 \\
& \lambda_3 = 0 \\
& \lambda_4 = 0 \\
& \lambda_5 = 0 \\
& \lambda_6 = 0 \\
& \lambda_7 = 0
\end{aligned}
$$

Now we can solve this linear problem using linprog in MATLAB.

```
clc; close all;
b = [0 0 -4 -3 -3 -2 -2]';
Aeq = [];
beq = [];
A = [1 -1 -1 0 0 0 0 0 0 0; ...
    0 0 1 -1 -1 0 0 0 0 0; ...
    1 0 0 0 0 1 0 0 0 0; ...
    0 1 0 0 0 0 1 0 0 0;...
    0 0 1 0 0 0 0 1 0 0;...
    0 0 0 1 0 0 0 0 1 0;...
    0 0 0 0 1 0 0 0 0 1]';
c = [0 -10 0 -6 -20 0 0 0 0 0];
% lb = [0 0 0 0 0];
% ub = [4 3 3 2 2];
[lda,fval] = linprog(b, A, c, Aeq, beq);
for i = 1:length(lda)
    fprintf('Lambda%d = %d\n', i, lda(i));
end
fprintf('The maximum value of function is %d\n', fval);
```

Code 7: Solving the Linear problem using linprog

The output of the code is as shown below.
Optimal solution found.

Lambda1 = 10
Lambda2 = 10
Lambda3 = -10
Lambda4 = 0
Lambda5 = 0
Lambda6 = 0
Lambda7 = -10
The maximum value of function is 60

Thus, we can see that solving the dual problem also yielded the same solution of the maximum value as that obtained by the primal problem.