ME 7015 Design Optimization Project

Instructor : Assoc. Prof. Aykut Kentli

Student : Hasan Şener

ID :524617038

Contents

[1. Introduction 2](#_Toc11626610)

[2. The Algorithm 3](#_Toc11626611)

[3. MATLAB Implementation 5](#_Toc11626612)

[4. Results 6](#_Toc11626613)

[5. Discussion 11](#_Toc11626614)

Solving Engineering Optimization Problems

Particle Swarm Optimization

# 1. Introduction

Particle Swarm Optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions (particles), and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best-known position, but is also guided toward the best-known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

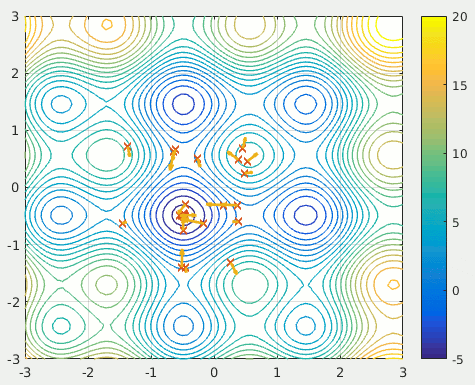


Figure Particle Swarm Opimization Simulation

As you see in the Fig.1. the **x** shows the where the particles are. These are our solutions in search space. Then the **arrows** in figure are velocities of each particle.

# 2. The Algorithm

PSO simulates the behaviors of bird flocking. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So, what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space. We call it "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called **pbest** (particle best). Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called **gbest** (global best).

The PSO algorithm consists of just three steps:

1) Evaluate fitness of each particle

2) Update individual and global bests

3) Update velocity and position of each particle with respect to given equations

The pseudocode of PSO can be seen below:

**For each** particle

Initialize particle

**End For**

**While** iterations < max iterations

**For each** particle

Calculate fitness value

If the fitness value is better than the best fitness value (pBest) in history

set current value as the new pBest

**End For**

\* Choose the particle with the best fitness value of all the particles as the gBest

**For each** particle

Calculate particle velocity according equation (a)

Update particle position according equation (b)

**End For**

**End While**

The equations for PSO are:

**// Velocity update equation**

is the particle index

is the inertial coefficient

are acceleration coefficients,

are random values regenerated every velocity update

is the particle’s velocity at time

is the particle’s position at time

is the particle’s individual best solution as of time

is the swarm’s best solution as of time

Since velocity update function is known, each particle’s position can be updated using the below equation:

**//** **Position update equation**

# 3. MATLAB Implementation

First define the PSO parameters,

swarm\_size = 8; % Swarm Size

no\_design\_variable = 4; % # of Design Variables

x\_max = 3; % Maximum Position

iter\_max = 30; % Maximum no. of Iterations

iter = 0; % Current Iterataion

particle\_best = cell(1,swarm\_size); % Particle Best Position

particle\_best\_objective = ones(1,swarm\_size)\*1E50; % Particle Best Position Objective Value

global\_best\_objective = 1E50; % Global Best Position Objective Value

global\_best = zeros(1,no\_design\_variable); % Global Best Position

particle\_position = cell(1,swarm\_size); % Particle Position

obj\_fun\_val\_particle = zeros(1,swarm\_size);% Objective Function Value of the Particle

particle\_velocity = cell(1,swarm\_size); % Particle\_Velocity

dummy = zeros(1,length(no\_design\_variable)); % Dummy List

dt = 1; % Time Step

Secondly, the problem definition will be taken into account;

P = 6000; % Applied Tip Load

E = 30e6; % Young Modulus

G = 12e6; % Shear Modulus

L = 14; % Length of the Beam

PCONST = 1000000; % Penalty Function Constant

TAUMAX = 13600; % Max Allowed Shear Stress

SIGMAX = 30000; % Max Allowed Bending Stress

DELTMAX = 0.25; % Max Allowed Tip Perfection

M = @(x) P\*(L+x(2)/2); % Bending Moment at Welding Point

R = @(x) sqrt((x(2)^2)/4+((x(1)+x(3))/2)^2); % Constant

J = @(x) 2\*(sqrt(2)\*x(1)\*x(2)\*((x(2)^2)/12+((x(1)+x(3))/2)^2)); % Polar Moment of Inertia

objective\_function = @(x) 1.10471\*x(1)^2\*x(2)+0.04811\*x(3)\*x(4)\*(14+x(2));

sigma = @(x) (6\*P\*L)/(x(4)\*x(3)^2); % Bending Stress

delta = @(x) (4\*P\*L^3)/(E\*x(4)\*x(3)^3); % Tip Deflection

Pc = @(x) 4.013\*E\*sqrt((x(3)^2\*x(4)^6)/36)\*(1-x(3)\*sqrt(E/(4\*G))/(2\*L))/(L^2); % Buckling Load

tau\_p = @(x) P/(sqrt(2)\*x(1)\*x(2)); % Tau\_prime

tau\_pp = @(x) (M(x)\*R(x))/J(x); % Tau\_double\_prime

tau = @(x) sqrt(tau\_p(x)^2+2\*tau\_p(x)\*tau\_pp(x)\*x(2)/(2\*R(x))+tau\_pp(x)^2); % Tau (Shear Stress)

Then define the constraints :

% Constraints

g1 = @(x) tau(x)-TAUMAX;

g2 = @(x) sigma(x)-SIGMAX;

g3 = @(x) x(1)-x(4);

g4 = @(x) 0.10471\*x(1)^2+0.04811\*x(3)\*x(4)\*(14+x(2))-5;

g5 = @(x) 0.125-x(1);

g6 = @(x) delta(x)-DELTMAX;

g7 = @(x) P-Pc(x);

Define penalty function to create a barrier:

penalty\_function = @(x) objective\_function(x) + PCONST\*(max(0,g1(x))^2+max(0,g2(x))^2+max(0,g3(x))^2+...

+max(0,g4(x))^2+max(0,g5(x))^2+...

max(0,g6(x))^2+max(0,g7(x))^2); % Penalty Function

Third step is initializing particles;

for i = 1:swarm\_size

for j = 1:no\_design\_variable

dummy(j) = rand()\*x\_max;

end

particle\_position{i} = dummy;

particle\_velocity{i} = particle\_position{i}/dt;

end

Lastly, main loop of PSO to calculate velocities & positions:

while iter < iter\_max

C1 = 1.8; % personal learning factor

C2 = 1.8; % social learning factor

W = 0.8; % inertia factor

iter = iter + 1;

for i = 1:swarm\_size

obj\_fun\_val\_particle(i) = penalty\_function(particle\_position{i});

if obj\_fun\_val\_particle(i) < particle\_best\_objective(i) && obj\_fun\_val\_particle(i) >= 0 % Best Local

particle\_best{i} = particle\_position{i};

particle\_best\_objective(i) = obj\_fun\_val\_particle(i);

end

end

if min(obj\_fun\_val\_particle) < global\_best\_objective && min(obj\_fun\_val\_particle) >= 0 % Best Global

global\_best = particle\_position{obj\_fun\_val\_particle == min(obj\_fun\_val\_particle)};

global\_best\_objective = min(obj\_fun\_val\_particle);

end

for i = 1:swarm\_size % Update Particle Positon

R1 = rand(); % Random Number

R2 = rand(); % Random Number

particle\_velocity{i} = W\*particle\_velocity{i}+C1\*R1\*(particle\_best{i}-particle\_position{i})/dt+C2\*R2\*(global\_best-particle\_position{i})/dt; % Update Particle Velocity

particle\_position{i} = particle\_position{i}+particle\_velocity{i}\*dt; % Update Particle Position

end

disp(['BEST PARTICLE VALUE >> ' num2str(global\_best\_objective)]);

end

disp(['BEST PARTICLE POSITION >> ' num2str(global\_best)]);

# 4. Results

Four engineering optimization problems solved with Particle Swarm Optimization. The results for the best particles can be seen below and the script files for the problems as stated below:

* **Welded Beam Design Optimization Problem (pso\_welded\_beam.m)**

In this problem the objective is to find the minimum fabrication cost, considerating four design variables: and constraints of shear stress, bending stress, buckling load and the deflection of the beam.

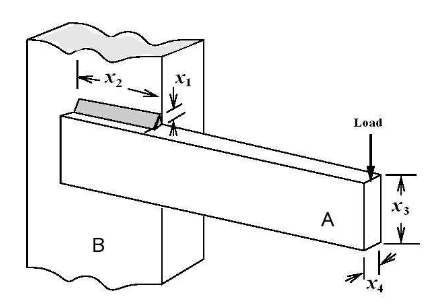
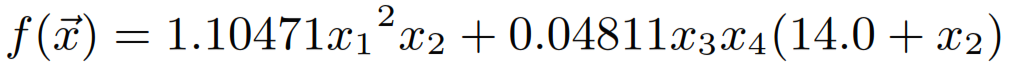
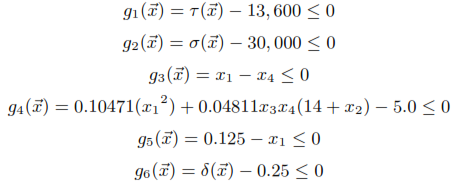


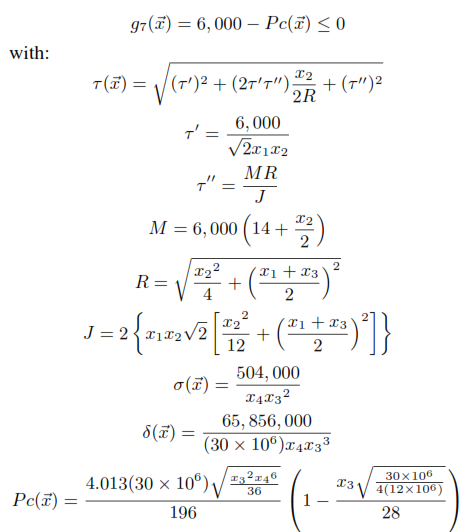
Figure Welded Beam

*Objective Function:*



*Constraints:*







Results found in PSO are (x1, x2, x3, x4):

***>> 0.72576 1.94 3.451 1.6656***

* **Pressure Vessel Design Optimization Problem (pressure\_vessel.m)**

In this problem the objective is to minimize the total cost, including the cost of materials forming the welding. Considering four design variables: which thickness, thickness of the head, the inner radius and the length of the cylindrical section of the vessel.

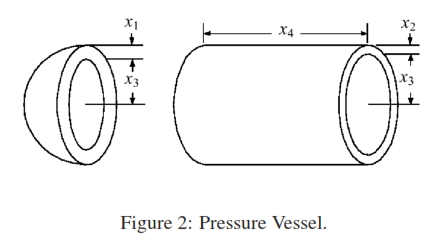
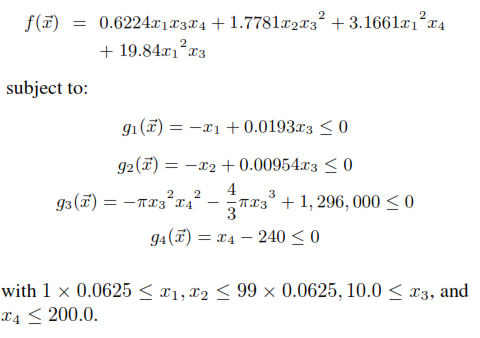


Figure Pressure Vessel

The objective function and constraints are below:



Results found in PSO are (x1, x2, x3, x4):

***>> 2.63319 13.4306 26.4714 23.9948***

* **Speed Reducer Design Optimization Problem (speed\_reducer.m)**

Considering the design variables face width , module of the teeth , number of teeth on pinion , length of the second shaft between bearings , length of the second shaft between bearings , diameter of the first shaft and the diameter of the first shaft . The weight of the speed reducer is to be minimized in this problem.

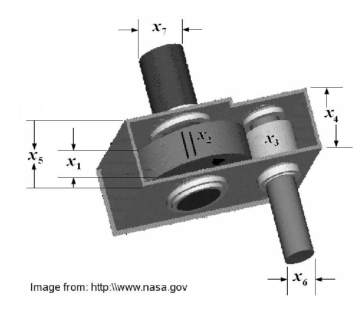
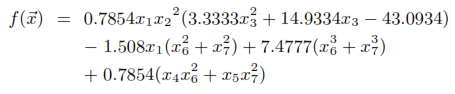
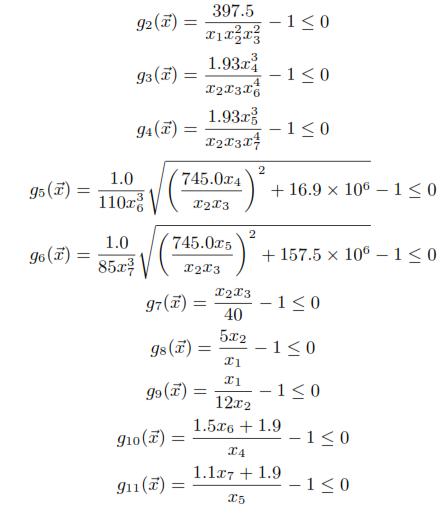


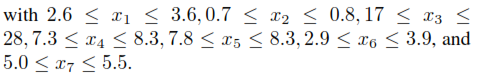
Figure Speed Reducer

The objective function and constraints are:









Results found in PSO are (x1, x2, x3, x4,x5,x6,x7):

***>> 8.24242 1.51559 15.6705 12.9359 7.92578 7.41818 5.61471***

* **Tension Spring Design Optimization Problem (tension\_spring.m)**

This problem minimizes the weight of tension spring subject to constraints of minimum deflection, shear stress, surge frequency, and limits on outside diameter and on design variables. There are three design variables in this problem: the wire diameter , the mean coil diameter and the number of active coils .

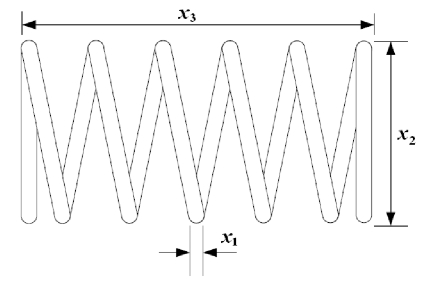
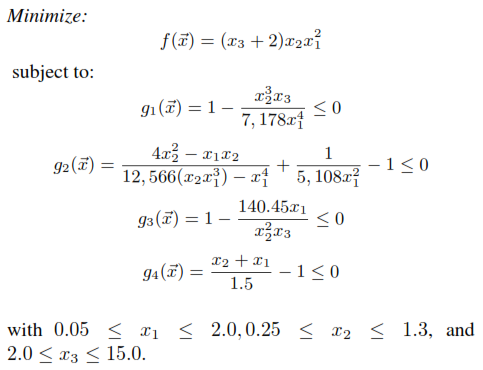


Figure Tension/Compression Spring

The objective function and constraints are:



Results found in PSO are (x1, x2, x3):

***>> 0.055896 0.51442 2.4015***

# 5. Discussion

The Particle Swarm Optimization bases swarm intelligence. The particles (solutions) were initialized randomly at search space then while changing each particle’s direction (velocity) every particle holds its best value. Then the best value of all personal best values is taken as global best value (swarm best value).

To accomplish swarm optimization below mathematical equations are used:

In engineering problems stated above, the PSO parameters taken from the given paper\*.

Swarm Size : 8 Particles

Neighborhood Size : 3

Inertia Factor : 0.8

Personal Learning Factor : 1.8

Social Learning Factor : 1.8

All the code can be found in “.m” files scripts.

\*: *Solving Engineering Optimization Problems with the Simple Constrained Particle Swarm Optimizer, Leticia C. Cagnina, Susana C. Esquivel, LIDIC, Universidad Nacional de San Luis, Argentina*