Design Optimization of Barrel Spring for Minimum Side Force

AAE: 550 Project Report

Author: Rahul Deshmukh
deshmuk5@purdue.edu
PUID: 0030004932

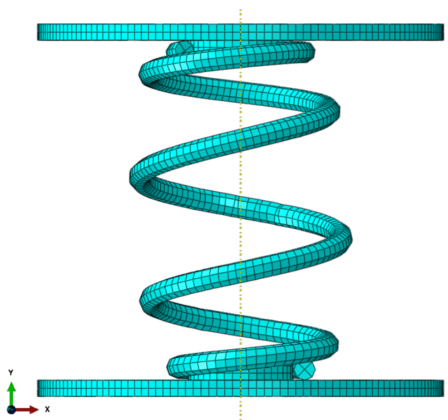
December 7, 2018

Contents

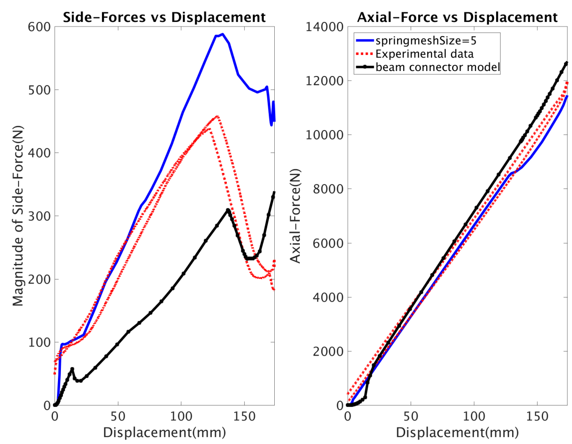
1 Introduction

Barrel Springs belong to the family of Helical springs but unlike regular helical springs they have varying spring-radius and pitch and can be constructed by joining two conical springs (refer to figure 1(a)).

During application, the spring is loaded axially and is required to exhibit a linear force-displacement response in the axial direction and minimum force in the lateral direction. The lateral forces, hereon to be called as Side-forces, are a consequence of friction and are observed to be highly non-linear (refer figure 1(b)). These side-forces cause wear and tear of the spring which reduces the life of the spring. Therefore, there is a need to find the optimum shape of spring which minimizes the side-forces.



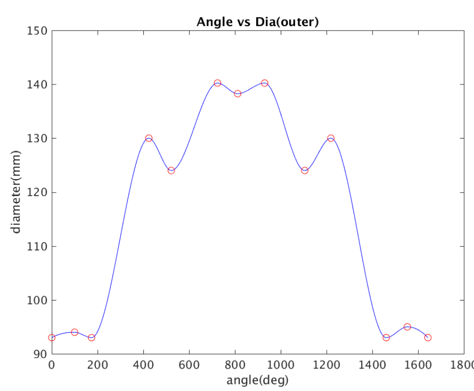
(a) Barrel Spring



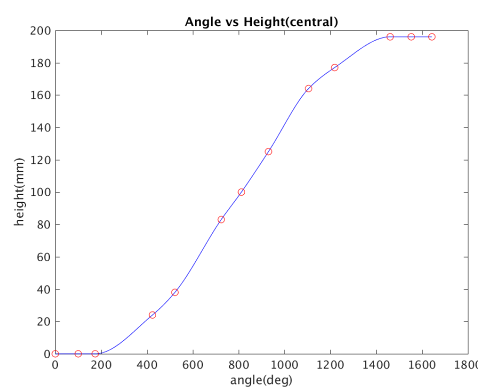
(b) Load Response

Figure 1: Spring and its Load response

The spring at hand has a particular spring-diameter vs angle (refer figure 2(a)) and pitch vs angle profile (refer figure 2(b)) defined which can be approximated by a *cubic spline* with **13** points which are the peaks and valleys of the diameter profile (refer figure 2(a)). These 13 points can serve as my design variables for the optimization problem.



(a) diameter profile



(b) height profile

Figure 2: Spring Profile

Note: I am using scripts for generating the solid model with given parameter and carry out the finite element simulation using ABAQUS. I have attached this script to an optimization routine.

2 Optimization Problem Formulation

2.1 Design Variables

My design variable $\vec{x}$ for this problem can be the diameter values at the 13 spline points, which can be quite expensive to start with. So, I have reduced the size by making assumption that the profile shown in the figure 2(a) remains symmetric. This reduces the size to **7**.

In addition to the above design variables, I have some fixed variables such as total number of turns of the spring $numTurn = 4.5$, Total Height of the spring $H = 196.5$ [mm], wire diameter $d = 13.5$[mm], height of the spring at the 13 points in figure 2(b), value of the angles for the 13 points in figure 2(a)&(b), Young's Modulus of the material $E = 300$[GPa], Poisson's ratio for the material $\nu = 0.3$ and friction coefficient $\mu = 0.6$.

2.2 Objective Function

As I am interested in finding a configuration with minimum side-force response I have scalarized the side-force vs displacement response using the area under the curve of side-force (using Trapezoidal rule). This is an **implicit function** given by:

$$\min_{\vec{x}} f(\vec{x}) = \int_0^{s=175mm} \text{SideForce}(\vec{x}) ds$$

where s is the displacement of the spring.

2.3 Constraints

2.3.1 Design Constraint

The Design dictates certain constraints on axial force at different displacement values as follows:

1) $\text{AxialForce}(\vec{x}, s = 175) \leq 12010$ [N] $\Rightarrow g_1(x) = \frac{\text{AxialForce}(\vec{x},s=175)}{12010} - 1 \leq 0$

2) $\text{AxialForce}(\vec{x}, s = b) \geq 8418$ [N] $\Rightarrow g_2(x) = 1 - \frac{\text{AxialForce}(\vec{x},s=b)}{8418} \leq 0$

3) $\text{AxialForce}(\vec{x}, s = a) \geq 4546$ [N] $\Rightarrow g_3(x) = 1 - \frac{\text{AxialForce}(\vec{x},s=a)}{4546} \leq 0$

Where $H = 196.5$ [mm] is the total height of the spring.

Note: Axial force is also an **implicit function** obtained from analysis and to get the above constraints I am using linear interpolation.

2.3.2 Geometric Constraint

I want the my design parameters to be in such a way that a **'valid'** spring which has **no self intersection** is generated. In my script I am using python's try and exception block to find a binary value for the constraint. Moreover, for the bounds considered (manufacturing tolerance) I always have a valid spring, I just need to ensure that I never generate an in-feasible solution at any point during the optimization.

2.3.3 Bounds

The spring has a manufacturing tolerance envelop on the profile such that the profile of the manufactured spring should not exceed a tolerance of $\pm 2.5\%$ from the master (baseline) spring values such that:

$$(1 - 0.025)x_{iM} \leq x_i \leq (1 + 0.025)x_{iM} ......\forall i$$

3 Choice of Optimization Method

When choosing the Optimization method for this project, I will have to keep in mind the following key-points:

(i) Since I don't have a function value for any in-feasible point, therefore I am restricted to choose between Interior Penalty method (SUMT), Methods of centers (direct methods) or Response Surface Approximations.

(ii) The axial force constraints are not really independent and cannot be treated directly which rules out methods of centers.

(iii) The objective function and constraints are implicit function therefore I can **only** find **numerical gradients** for these functions which adds to the computational cost. This makes any kind of derivative based methods unsuitable as function evaluations are costly.

From the above discussion, Response Surface Approximations comes out as a suitable method for optimization. Moreover, with D-optimal sets the method becomes less costly and more suitable for the problem. But it comes with a caveat that we may not be able to find the best solution.

Note: I had initially planned on using Methods of centers for the problem as stated in the abstract but, after discussion with TAs for AAE550 I came to the conclusion that response surfaces was the most suitable method.

4 Response Surface Approximation

As the computational cost of one analysis is quite high, I am using D-optimal set for generation of a Quadratic Response surface for the objective function and constraints. I am using Matlab's `cordexch` function for generating the D-optimal set with a total of $(7+1)(7+2)/2 = 36$ design points.

The tasks carried out can be succinctly listed out as follows:

1) Data Generation using D-optimal set which involves:

   i) Decode design to actual variables using Linear Transformation
   ii) Create geometry of spring using the actual variables
   iii) Carry-out FE simulation using ABAQUS
   iv) Read results, obtain objective function by integrating and constraints by interpolation.
   v) Save results.

2) Find Coefficients for Quadratic Response Surface for objective function and constraints

3) Scale the approximated constraints to the form $\hat{g}_j(\vec{x}) \leq 0$. It should be noted that as we have approximated the constraints using a quadratic approximation, therefore they will be Non-Linear Constraints.

4) Use SQP for solving the approximate problem with numerical gradients.

The Scripts for the tasks are attached in the appendix(7) .

Note: I have used Matlab's `fmincon` function with `optimoptions('fmincon','Display','iter','Algorithm','sqp')` to use the SQP algorithm with numerical gradients.

5 Results

The D-optimal set generated had a D-Efficiency of 42.3839%.
The value of the objective function ranged from $4.1174E + 04$ to $8.3107E + 04$ [N-mm].

With an initial solution of $x_0 = (LB + UB)/2$, SQP exited with an `exitflag`= 1 taking a total of 5 iterations and 52 function evaluations. The Final optimized solution came out as:

$$x^* = \begin{bmatrix} 90.8505 \\ 91.7865 \\ 95.3353 \\ 133.5062 \\ 123.0547 \\ 144.8530 \\ 140.9976 \end{bmatrix} \text{[mm]} \qquad \hat{f}^* = 3.6103e + 04\text{[N-mm]} \qquad \hat{g}_j(x^*) = \begin{bmatrix} -0.9413 \\ -0.1571 \\ -0.0276 \end{bmatrix} \text{[No-units]}$$

As can be observed the approximated constraints are satisfied for the above solution. Upon carrying out a final actual function evaluation for the optimal solution, the function value came as:

$f(x^*) = 4.5546E + 04$ [N-mm]

When compared to the value of the actual function for complete list of experiments it was observed that the optimal solution from the approximate problem was better than all points **except** for one (design point #15) which had an actual function value of $4.1174E + 04$ [N-mm].

6 Comments

Although, with the first trial of response surfaces I was not able to obtain a better solution but I was still able to obtain a solution which was better than all except one point.

This indicates that the method does have a potential to find out a better solution which can be carried out in the future by trying out the following ideas:

1. Trying Sequential Optimization for response surfaces.
2. Trying out other interpolating functions like Kriging functions.
3. I had reduced the number of design variables by assuming symmetry, on obtaining a workable solution I can increase the design variables back to 13.
4. Create a function wrapper for the analysis script and carry out optimization using gradient based methods (this will be time-taking).

7 Appendix

The Source code for the project is as follows:

7.1 Script for Data Generation

7.1.1 Main Script

```
%% AAE 550: Project: Part -1
% Design Optimization of Barrel Spring for Minimum Side Force
% Author: Rahul Deshmukh
% PUID: 0030004932
% email: deshmuk5@purdue.edu
%% Data Generation for Response Surfaces: Upfront work
% Main file for Candidate Generation and runnign the simulations
% Will be using D-Optimal set for generating the sample set
% Will run a FE simulation in ABAQUS for each sample in the set
% Store data pertaining to Objective function and all constraints in
% suitable format
clc; clear all;
close all; warning off;
% get Parent directory where to store the simulations
parent_dir = uigetdir('/export/home/a/deshmuk5/abaqus/MWspring/');
diary(strcat(parent_dir,'/Log.txt'));
%% D-Optimal Set
% specify the number of design variables
num_des_var = 7
nruns = (num_des_var+1)*(num_des_var+2)/2.0
% generate D-optimal set
[design_pts,X]= cordexch(num_des_var,nruns,'quadratic',...
                        'levels',3*ones(num_des_var,1),'tries',100);
%% Run Simulations
% initialize storing structures
original_x = []; % stores transformed value of design variable
obj_fun = [];
geo_const = [];
axf_max = [];
axf_ha = [];
axf_hb = [];

% Declare Simulation constants
wire_dia = 13.5% in mm
plate_thickness =10.0%in mm
friction_coeff = 0.6
springyoungmodulus = 300E3 %Mpa
platemeshSize = 10.0
springmeshSize = 5.0
N=500 %number of points for the coordinate file

fprintf('##--------------------Runs----------------------------------##\n');
%% Loop for nruns
for i=1:nruns
   fprintf(strcat('Run #',num2str(i),'\n'));
   % find ith coded design
   ix = design_pts(i,:) % row vector
   % convert to original variable
   [tr_x,dia_pts,angle_pts,height_pts,LB_x,UB_x] = transform2original(ix);
   original_x = [original_x;tr_x];
   %% create folder for this simulation in parent dir
   foldername= strcat('run_',num2str(i));
   folderpath= strcat(parent_dir,'/runs/',foldername);
   mkdir(folderpath);
   %% print Readme file of parameter values for this sample
   filepath = strcat(folderpath,'/');
   printreadme_solid(filepath,wire_dia,i,springyoungmodulus,...
                friction_coeff,springmeshSize,platemeshSize,...
                dia_pts,angle_pts,height_pts);
```

```
    fprintf('* Readme file printed *\n');
    %% print Coordinate file
    %(a): find XYZ coordinates using cubic spline
    [X,Y,Z,spring_height,spring_max_radius,spring_initial_radius]=...
        cubicspline_interpolate(filepath,N,dia_pts,angle_pts,height_pts,wire_dia);

    %(b): Print XYZ coordinates into a text file
    coordinatefile_solid(filepath,X,Y,Z);
    fprintf('* Coordinate file printed *\n');
    %% print FE script for this sample
    printpythoncode_solid(filepath,N,wire_dia,spring_height,spring_max_radius,...
                          spring_initial_radius,plate_thickness,springyoungmodulus,...
                          friction_coeff,springmeshSize,platemeshSize);
    fprintf('* Abaqus python script printed *\n');
    %% call Abaqus to run the script
    fprintf('* Submitting job to Abaqus *\n');
    cmdstatus = system(strcat('abaqus cae noGUI=',filepath,'MyModel_solid.py'));
    while cmdstatus
        %if license not fetched wait
        fprintf(2,'!!Error: Abaqus Licence not fetched, trying again in 5 sec!!\n');
        pause(5);% 5sec
        cmdstatus = system(strcat('abaqus cae noGUI=',filepath,'MyModel_solid.py')); %execute again
    end
    %% Read Side-Force, Axial-Force, Geometric constraint
    fprintf('* Reading results *\n');
    % geometric constraint
    fid = fopen(strcat(filepath,'notinterfering.txt'),'r');
    notinterfering = fscanf(fid,'%d');
    fclose(fid);
    geo_const  = [geo_const;notinterfering];
    %  Side-Force, Axial-Force
    [sideforcemag,sideforce,axialforce,displacement]=...
        readdata_solid(filepath,spring_height,wire_dia);
    %% Obtain Objective function and constraints
    % objective function: integrate the side force
    integrated_side_force = integrate_force(sideforcemag,displacement);
    obj_fun = [obj_fun;integrated_side_force];
    % constraints: interpolate axial force
    % axial force at maximum working height
    temp = sideforcemag(end);
    axf_max = [axf_max;temp];
    % axial force @ Ha = sping_height -63.5 [mm]
    temp = interpolate_fun(axialforce,displacement,spring_height-114.3);
    axf_ha = [axf_ha;temp];
    % axial force @ Hb = sping_height -114.3 [mm]
    temp = interpolate_fun(axialforce,displacement,spring_height-63.5);
    axf_hb = [axf_hb;temp];
end
%% save stored data to a mat file for next step
data = [original_x,obj_fun,axf_max,axf_ha,axf_hb,geo_const];
save(strcat(parent_dir,'/data.mat'),'data','LB_x','UB_x');
fprintf('#------------------Data Generation Complete----------------#\n');
diary off;
```

7.1.2 Functions

Transforming coded to original variables

```
function [tr_x,dia_pts,angle_pts,height_pts,LB_x,UB_x] = transform2original(x)
%% Transform the coded sample to orginal variables
% takes the coded sample ix which has 3 levels: -1,0,1
% transforms the coed to original variable value
%% Original Problem Bounds Definition
%--------------------------------------------------%
% format of x
% x: [d1,d2....d7]
% other dia assumed to be symmetric about the center
```

```
%--------------------------------------------------------%
dia = x; % all diameters
for i=1:length(x)-1
    dia = [dia,x(length(x)-i)];
end

% ----------------Master Spring configuration-----------------------------%
% Spring4
Master  = [93.18, 94.14, 93.01,130.25, 126.21, 141.32, 140.44, 142.70, 124.50, 130.95, 93.25, ...
    95.82, 92.76 ];
angle_pts = [0 100.8 173.9 424.0 522.2 724.5 812.7 930.8 1105.5 1219.3 1461.5 1553.8 1643.1];%in ...
    degree
height_pts= [0 0.9 0.1 24.1 38.3 83.4 100.8 125.1 164.2 177.9 196.7 195.3 196.5];% in mm
% -----------------------------------------------------------------------%

% manufacturing tolerance on master spring dimensions
tol = 2.5; % pm percentage

LB = (1-tol/100)*Master;
UB = (1+tol/100)*Master;
LB_x = LB(1:7);
UB_x = UB(1:7);

%% Linear Transfromation: -1,1 -> LB,UB
dia_pts = LB + (UB-LB).*(dia+1.0)/(2.0);
tr_x = dia_pts(1:7);
end
```

Integrating Side-Force: The Objective Function

```
function integrated_force = integrate_force(force,d)
% Integrates the sideforcemag vs displacement curve using Trapezoidal rule
% Input: force: nx1 vector
%        d: displacement vecto nx1
% Output: integrated_force: scalar value
%% Begin
% Approximating the curve with a linear approximation and then integrating
% the data with trapezoidal rule. The integration will be accurate for the
% linear apporoximation.
integrated_force = trapz(d,force); % takes x,y format
end
```

Interpolating Axial-Force: Constraints

```
function f =interpolate_fun(force,d,dq)
% Interpolates the function using a linear approximation and then finds
% the value of the function at the data point x = d
% Input: force: vector nx1
%        d: displacement nx1
%        dq: query point
% Output: f: scalar value
%% Begin
f = interp1(d,force,dq);
end
```

Note: I have deliberately not included the functions for generating the geometry and printing the python script for analysis because it would have increased the number of pages to a lot.

7.2 Script for Response Surfaces

7.2.1 Main Script

```
%% AAE 550: Project : Part-2
% Design Optimization of Barrel Spring for Minimum Side Force
% Author: Rahul Deshmukh
% PUID: 0030004932
% email: deshmuk5@purdue.edu
clc;
% clear all;
%% Load Data
% find location of the file
[filename,filepath]=uigetfile('*.mat','Select the Mat File');
% load mat file
load(strcat(filepath,filename));
% ------- variables loaded --------------%
% data= [X,obj_fun,axf_max,axf_ha,axf_hb,geo_const]
% LB_x, UB_x: bounds on the design variables
% --------------------------------------%
%% Find coefficients for response surfaces
% number of design variables
n = 7;
% obtain the objective function and constraint data points
obj_fun = data(:,n+1);
axf_max = data(:,n+2);
axf_ha  = data(:,n+3);
axf_hb  = data(:,n+4);
geo_const = data(:,n+5); % not using this constraint: assuming it to be
% always satisfied for the chosen bounds

% make the basis matrix X
X = data(:,1:n);
combinations = combnk(1:n,2);
XiXj = [];
for i=1:size(combinations,1)
    c1= combinations(i,1);
    c2= combinations(i,2);
    XiXj = [XiXj, X(:,c1).*X(:,c2)];
end
X = [ones(length(obj_fun),1), X, XiXj, X.^2];

% find coefficients for response surface
a_fun   = X\obj_fun;
a_axf_max = X\axf_max;
a_axf_ha = X\axf_ha;
a_axf_hb = X\axf_hb;

%% Using SQP to solve the Quadratic approximation problem
A = []; % no linear constraint all quad cons
b = [];
Aeq = []; % no equality constraint
beq = [];
LB = LB_x';
UB = UB_x';

options = optimoptions('fmincon','Display','iter','Algorithm','sqp');
% options = optimoptions('fmincon','Algorithm','sqp','Display','iter',...
%     'SpecifyObjectiveGradient',true,'SpecifyConstraintGradient',true,...
%     'DerivativeCheck','on');

x0=(LB+UB)/2.0;
[x_star,fval,exitflag,output] = fmincon(@(x)fun(x,a_fun),x0,A,b,Aeq,beq,...
 LB,UB,@(x)NonLinCon(x,a_axf_max,a_axf_ha,a_axf_hb),options);
%% Print Final solution
fprintf('***************Optimized solution*****************************\n');
x_star
fval
exitflag
```

```
fprintf('Constraints\n');
NonLinCon(x_star,a_axf_max,a_axf_ha,a_axf_hb)
```

7.2.2 Functions

Approximated Objective Function

```
function [f,grad_f]= fun(x,a_fun)
%% Function for finding the value of Objective Function for a particular x
%  Input: a_fun: col vector of coefficients of the approximation nx1
%  Output: f: function value
%          grad_f: gradient vector if requested
%% Build the X row and find f
n = length(x); %number of design variables
% generate the m = (n+2)*(n+1)/2 dim quadratic row
%  st [1;xi,xi*xj;xi^2]
combinations = combnk(1:n,2);
xixj = [];
for i=1:size(combinations,1)
    c1= combinations(i,1);
    c2= combinations(i,2);
    xixj = [xixj;x(c1)*x(c2)];
end
X = [1; x; xixj; x.^2];
% function value
f = a_fun'*X;
%% Analytical Gradient
if nargout > 1
    grad_f = [];
    for i=1:n
       % for a0 term
       igrad = [0] ;
       % for aixi terms
       temp = zeros(n,1);
       temp(i) = 1;
       igrad = [igrad;temp];
       % for aijxixj terms
       temp = zeros(size(combinations,1),1);
       for j =1:size(combinations,1)
           if ¬isempty(find(combinations(j,:)==i))
               temp_copy =  combinations(j,:);
               temp_copy(find(combinations(j,:)==i))='';
               temp(j) = x(temp_copy);
           end
       end
       igrad = [igrad;temp];
       % for xi^2 terms
       temp = zeros(n,1);
       temp(i) = 2*x(i);
       igrad = [igrad;temp];
       grad_f = [grad_f; a_fun'*igrad];
    end
end
end
```

Approximated Non-Linear Constraints

```
function [g,h,grad_g,grad_h]=NonLinCon(x,a_axf_max,a_axf_ha,a_axf_hb)
%% Function for generating Non-Linear Constraint for SQP
% Input: x: col vector of desing variables
%        a_axf_max,a_axf_ha,a_axf_hb: coefficients of response surfaces for
%        the constraints (as col vectors)
% Output: g,h: constraint values as vectors ineq and eq resp
%         grad_g,grad_h: gradient vectors of constraints if requested
%% Build the X row and find g,h
% find number of design variables
n = length(x);
% make the row for of basis vector
```

```
combinations = combnk(1:n,2);
xixj = [];
for i=1:size(combinations,1)
    c1= combinations(i,1);
    c2= combinations(i,2);
    xixj = [xixj;x(c1)*x(c2)];
end
X = [1; x; xixj; x.^2];
% inequality constraint
%--------------Limits for the inequality constraints --------------------%
Lim_axf_max = 12010; %[N] g(x)≤lim
Lim_axf_ha = 4546; %[N]   g(x)≥lim
Lim_axf_hb = 8418; %[N]   g(x)≥lim
Lim = [Lim_axf_max,Lim_axf_ha ,Lim_axf_hb];
Lim = kron(Lim,ones(length(a_axf_max),1));
%-----------------------------------------------------------------------%
% find the values of forces using resp coefficients and X
 a_g = [a_axf_max,-1*a_axf_ha,-1*a_axf_hb]./Lim;
% convert to the form of g_j(x)≤0
g = a_g'*X+[-1;+1;+1];

% equality constriant
h = [];

%% Analytical Gradient
if nargout > 2
    grad_g = [];% to be stacked as cols
    for ig =1:size(a_g,2)
        % for igth constraint
        grad_gi = [];
        for i=1:n
            % for a0 term
            igrad = [0] ;
            % for aixi terms
            temp = zeros(n,1);
            temp(i) = 1;
            igrad = [igrad;temp];
            % for aijxixj terms
            temp = zeros(size(combinations,1),1);
            for j =1:size(combinations,1)
                if ¬isempty(find(combinations(j,:)==i))
                    temp_copy =  combinations(j,:);
                    temp_copy(find(combinations(j,:)==i))='';
                    temp(j) = x(temp_copy);
                end
            end
            igrad = [igrad;temp];
            % for xi^2 terms
            temp = zeros(n,1);
            temp(i) = 2*x(i);
            igrad = [igrad;temp];
            grad_gi = [grad_gi; a_g(:,ig)'*igrad];
        end
        grad_g = [grad_g,grad_gi];
    end
    grad_h = [];
end
end
```