MAE 5350: HW #3

Multidisciplinary Design Optimization

Hanfeng Zhai*

Sibley School of Mechanical and Aerospace Engineering,

Cornell University

November 3, 2021

Q1. Penalty Methods continued

Consider the problem

$$\begin{aligned} \min \ & x + y \\ \text{subject to} \quad & x^2 + y^2 = 2 \end{aligned} \tag{1}$$

(a) Apply KKT conditions to solve the problem

Solution: We first rewrite the problem in standard form:

$$\begin{aligned} \min \ & x + y \\ \text{subject to} \quad & x^2 + y^2 - 2 = 0 \end{aligned} \tag{2}$$

We first write out the pseudo objective:

$$(x + y) + \lambda(x^2 + y^2 - 2) = 0 \tag{3}$$

By applying the KKT condition:

$$\begin{aligned} 1 + 2x\lambda &= 0 \\ 1 + 2y\lambda &= 0 \\ \lambda(x^2 + y^2 - 2) &= 0 \\ \lambda &\geq 0 \end{aligned} \tag{4}$$

*Email: hz253@cornell.edu

By solving this problem with `vpasolve` in MATLAB® and generate the following code in an `.mlx` file to directly output the LaTeX font results:

```
syms x y lambda
eqn1 = 1+ 2*lambda*x == 0; eqn2 = 1+2*lambda*y == 0; eqn3 = lambda*(x^2 + y^2 - 2) == 0;
eqns = [eqn1,eqn2,eqn3];
vars = [x y lambda];
answer = vpasolve(eqns, vars);
[(answer.x),(answer.y),(answer.lambda)]
```

The solutions are:

$$\begin{pmatrix} 1.0 & 1.0 & -0.5 \\ -1.0 & -1.0 & 0.5 \end{pmatrix} \tag{5}$$

Since we already know $\lambda \geq 0$, therefore the solutions are obtained

$$x = y = -1; \ \lambda = 0.5 \tag{6}$$

(b) Formulate a quadratic penalty function for this problem. Derive the corresponding optimality conditions as a function of the penalty parameter ρ.

Solution: We first write out the Quadratic Penalty Function:

$$\Phi_Q(\mathbf{x}, \rho_p) = (x + y) + \rho_p(x^2 + y^2 - 2)^2 \tag{7}$$

We then first calculate the gradient of the pseudo-objective function:

$$\nabla\Phi_Q = \begin{pmatrix} 1 + 4\,\rho_p\,x\,(x^2 + y^2 - 2) \\ 1 + 4\,\rho_p\,y\,(x^2 + y^2 - 2) \\ (x^2 + y^2 - 2)^2 \end{pmatrix} = 0 \tag{8}$$

To obtain the optimal point we need $\nabla\Phi_Q = 0$, analyzing the three terms, we chose exterior penalty function method to let $\rho_p \to +\infty$ we know that $(x^3 + xy^2 - 2) = (x^2y + y^3 - 2) = (x^2 + y^2 - 2)^2 = 0$, in such case we deduce that $x = y = \pm 1$.

(c) Plot the contours of your quadratic penalty pseudo-objective for the case of $\rho = 0.5$ and $\rho = 5$. Graphically determine the optimal solution in each case and comment.

From Figure 1 we can deduce from the scheme boundary as the black dotted lines from sub figures **A** and **B** as ρ increases the cruve is approximating the point $(-1, -1)$. Therefore we can obtain that optimal point is $x = y = -1$.

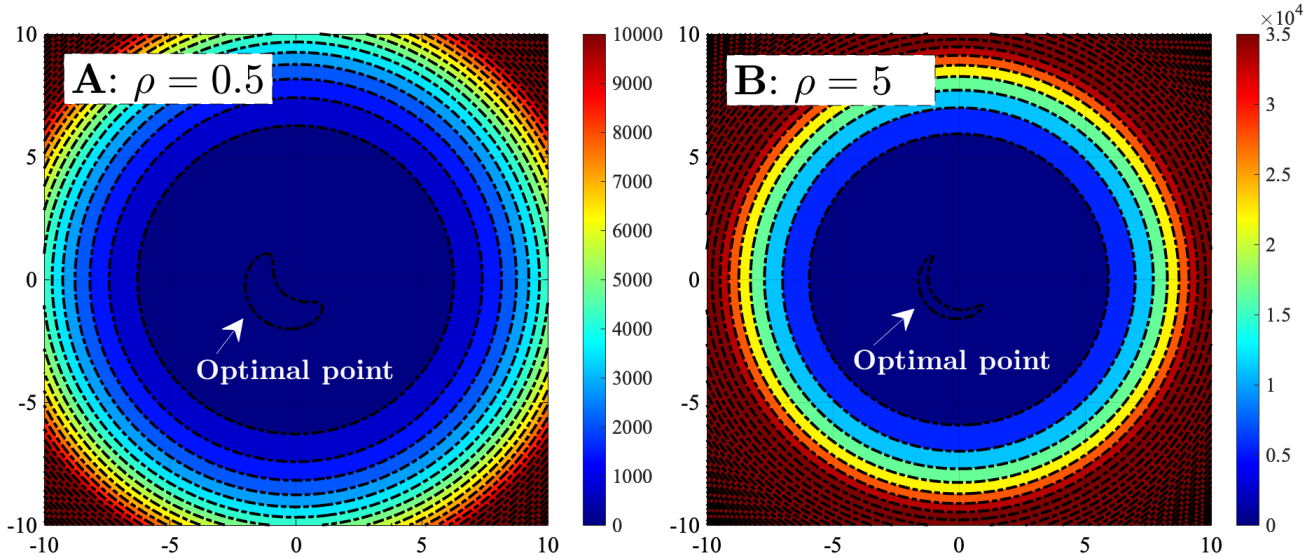
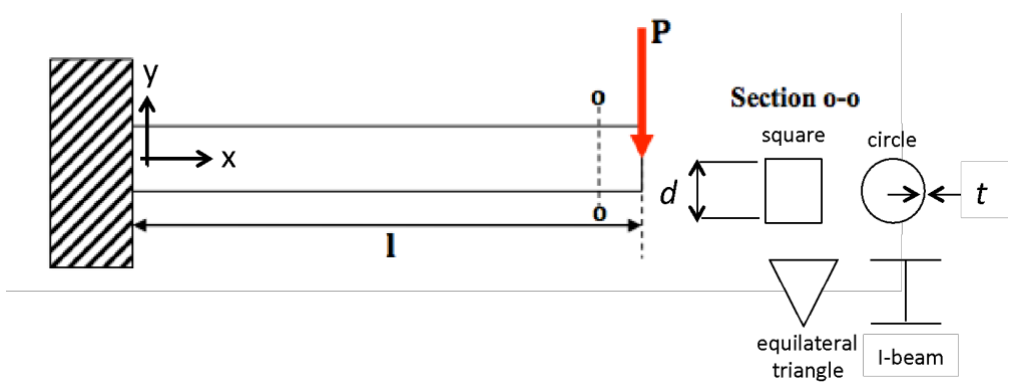


Figure 1: The contour plot for the two quadratic penalty pseudo-objective functions, when **A**. $\rho = 0.5$. **B**. $\rho = 5$.

Q2. Simulated Annealing

A cantilever beam of uniform cross-section with linear dimension d and thickness t (see figure below) has to be designed for minimum mass. The beam is of fixed length $l = 0.3m$ and carries a tip load of $P = 1kN$ at the free end. There are four different geometries available for the beam cross-section: square, circle, equilateral triangle, and I-beam. Finally, the beam can be made of steel, aluminum, or titanium.

There are two explicit constraints given for this design problem:



(1) Maximum bending stress in the beam has to be less than 90% of the yield stress of the material chosen:

$$\sigma_{\max,x} = \frac{Ply_s}{I_s} \leq 0.9S_m$$

where $\sigma_{\max,x}$ is the normal stress in the x-direction, y_s is the maximum distance from the shape centroid to the top or bottom edge, I_s is the cross-sectional moment of inertia of the shape chosen, and S_m is

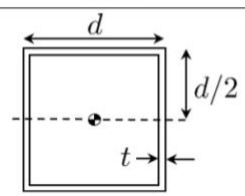
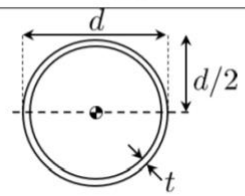
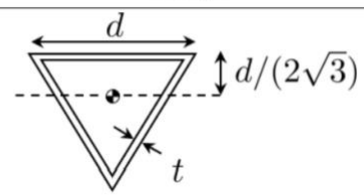
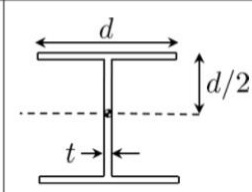
the yield stress of the material.

(2) Maximum tip deflection of the beam has to be less than 3mm:

$$\delta = \frac{Pl^3}{3E_m I_s} \leq 0.003$$

where E_m is the Young's modulus of the material chosen and I_s the same as in (1).

The table below summarizes key relationships for each of the different beam sections:

	Square	Circle	Triangle	I-beam
				
y_s	$d/2$	$d/2$	$\left(1 - \frac{1}{2\sqrt{3}}\right) d$	$d/2$
A_s	$d^2 - (d-2t)^2$	$\frac{\pi}{4}\left(d^2 - (d-2t)^2\right)$	$\frac{\sqrt{3}}{4}\left(d^2 - (d-2\sqrt{3}t)^2\right)$	$3dt - 2t^2$
I_s	$\frac{1}{12}\left(d^4 - (d-2t)^4\right)$	$\frac{\pi}{64}\left(d^4 - (d-2t)^4\right)$	$\frac{\sqrt{3}}{96}\left(d^4 - (d - t\cdot 2\sqrt{3})^4\right)$	$\frac{t}{12}\left((d-2t)^3 + 2dt^2 + 6d(d-t)^2\right)$

The table below summarizes key material properties of steel, aluminum, and titanium:

Material	Young's modulus E_m [GPa]	Yield stress S_m [MPa]	Density ρ_m [kg/m3]
Steel	200	300	7600
Aluminum	75	200	2700
Titanium	120	800	4400

(a) Find a constraint relating d and t as a function of the shape to upper-bound t, and a constraint relating d and l to upper-bound d. Finally, use engineering judgment to lower-bound t. Formulate the full optimization problem in standard form including the bounding constraints on t and d.

Solution: First, the constraints of yield stress and tip deflection can be simplified to:

$$\frac{y_s}{I_s} \leq 0.9\frac{S_m}{Pl}$$
$$I_s \geq \frac{1000Pl^3}{9E_m}$$

where y_s and I_s are related to t and d.

According to the given table, we can write $y_s = y_s(d)$, $A_s = A_s(t,d)$, and $I_s = I_s(t,d)$. Therefore we assume there exists functions f and g that $t = f(y_s, A_s, I_s)$ and $d = g(y_s, A_s, I_s)$.

The mass of the beam can be written as $m = \rho_m V_m$, where ρ_m is adapted from the given table.

The volume V_m can be calculated from $V_m = A_s y_s$, where for the three different beam sections:

$$\begin{aligned}
\text{square}: V_m &= (d^2 - (d - 2\,t)^2)l \\
\text{circle}: V_m &= \frac{l\pi\left(d^2 - (d - 2\,t)^2\right)}{4} \\
\text{triangle}: V_m &= -\frac{l\sqrt{3}\left(\left(d - 2\sqrt{3}\,t\right)^2 - d^2\right)}{4} \\
\text{I} - \text{beam}: V_m &= l(3\,d\,t - 2\,t^2)
\end{aligned} \tag{9}$$

We can then write out the problem in standard form

$$\begin{aligned}
\min\ & \rho_m V_m \\
\text{s.t.}\ & \frac{Ply_s}{I_s} \leq 0.9S_m \\
& \frac{Pl^3}{3E_m I_s} \leq 0.003
\end{aligned} \tag{10}$$

Also, considering the lower and upper bounds, we reconsider the schematic figures given in the instructions. We know that the geometry cannot violate basic physical sense; therefore we can write out the relations between t and d as for the upper bounds for t:

$$\begin{aligned}
\textbf{Upper bounds for t}: \quad & \text{For square}: t \leq \frac{d}{2} \\
& \text{For circle}: t \leq \frac{d}{2} \\
& \text{For triangle}: t \leq \frac{d}{2\sqrt{3}} \\
& \text{For I} - \text{beam}: t \leq \frac{d}{2}
\end{aligned}$$

For the upper bounds for d, we know that with the definition of a cantilever beam, the width cannot exceeds its length, therefore:

$$\textbf{Upper bounds for d}: \quad d \leq l$$

And for real-world applications, a beam usually has a thickness of 200 - 300 mm [Ref.]. Therefore we set the lower bound of t as 100 mm.

(b) Solve this constrained optimization problem using the Simulated Annealing (SA) algorithm.

Solution: To solve this problem, we first need to formulate the objective function (evaluation function in SA), nominated as `objectiveBeam.m` (As shown in the following codes). Here, we

impose the constraints as in the objective function (evaluation function), by using the absolute penalty function method: the pseudo objective can be written as $\mathcal{J} = J + \rho_p(|\texttt{constraint1}| + |\texttt{constraint2}| + ...)$, where $\mathcal{J}$ is the pseudo objective and J is the original objective function, ρ_p is the multiplier for the constraints[1]. However, it should be noted that this methods does not guarantee all the constraints will be perfectly satisfied, since there are multiple constraints and is we multiplying them to the same ρ_p there might be some solutions violating the constraints[2]. In this objective function, the input is taken as a vector containing four components of the design variables: t, d, materials and shapes. The parameters corresponding to the four input variables are given in the form of `if` loops. And the eventual objective is the `mass`, which also contain the constraints as we just mentioned.

```
function mass = objectiveBeam(x0)
Input = x0;
%% Input the vars.
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%% optimization constants
P = 1e3; l = .3;
%% judge loop for materials
if material == 1 % steel
    Em = 200e9;
  S_m = 300e6;
  rho_m = 7600;
elseif material == 2 % alum
  Em = 75e9;
  S_m = 200e6;
  rho_m = 2700;
elseif material == 3 % titanium
  Em = 120e9;
  S_m = 800e6;
  rho_m = 4400;
end
%% judge loop for shape
if shape == 1 % square
  y_s = d./2;
  A_s = d.^2 - (d - 2.*t).^2;
  I_s = (1/12).*(d.^4 - (d - 2.*t).^4);
elseif shape == 2 % circle
  y_s = d./2;
  A_s = (pi./4).*(d.^2 - (d - 2.*t).^2);
  I_s = (pi./64).*(d.^4 - (d - 2.*t).^4);
elseif shape == 3 % triangle
  y_s = (1 - 1./(2.*sqrt(3))).*d;
```

[1] In this way we impose the constraints to the evaluation function for simulated annealing.
[2] This will be further discussed in the next few sub questions.

```
  A_s = (sqrt(3)./4).*(d.^2 - (d - 2.*sqrt(3).*t).^2);
  I_s = (sqrt(3)./96).*(d.^4 - (d - 2.*sqrt(3).*t).^4);
else % I-beam
  y_s = d./2;
  A_s = 3.*d.*t - 2.*t.^2;
  I_s = (t./12).*((d - 2.*t).^3 + 2.*d.*t.^2 ...
            + 6.*d.*(d - t).^2);
end

%% judge if constraints are violated

sigma_max = (P.*l.*y_s)./(I_s);
delta = (P.*l.^3)./(3.*Em.*I_s);

rho_p = 1e7; % penalty function --> this value is very important!!

P = rho_p * abs(sigma_max-0.9*S_m) + rho_p * abs(delta-0.003) + rho_p * abs(2*t - d) + rho_p * abs
    (d - l);
mass=rho_m*A_s*l + P;

% relation of mass to t & d
end
```

And then we can write out the perturbation function, nominated as `perturbBeam.m`: taking similar strategy as the objective (evaluation) function, we first input the design variables in a vector form, then we use the MATLAB built-in `randi` function to randomly generate a number between 1 to 4 to judege which inout design variables to perturb. Then if the variables are shape or materials, we perturb them by randomly generate a new number within the design range. If the perturbed design variables are t or d, we first perturb either t or d by randomly generate a real number within the given range ([0, 1] in our problem), and link the other with the geometric inner constraint as we given in Q1:

```
function [xp]=perturbBeam(x0)
Input = x0;
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%
num = randi([1 4]);
%%
    if num == 3
        matrl = randi([1 3]);
        xp = [t d matrl shape];
    end

    if num == 4
        shap = randi([1 4]);
```

```
        xp = [t d material shap];
    end
%%
xp_t=x0(1);
xp_d=x0(2);
xlb=0;
xub=1;
%
    if num == 1
%        indx=round(rand(1)+1);
       dx=(xub-xlb)*rand(1)+xlb;
       xp_t=dx;
       if shape == 1 %square
           xp_d = 2*xp_t - dx;
       elseif shape == 2 %circle
           xp_d = 2*xp_t - dx;
       elseif shape == 3 %triangle
           xp_d = 2*sqrt(3)*xp_t - dx;
       elseif shape == 4 %I-beam
           xp_d = 2*xp_t - dx;
       end
       xp = [xp_t xp_d material shape];
    end

    if num == 2
%        indx=round(rand(1)+1);
       dx=(xub-xlb)*rand(1)+xlb;
       xp_d=dx;
       if shape == 1 %square
           xp_t = xp_d/2 + dx;
       elseif shape == 2 %circle
           xp_t = xp_d/2 + dx;
       elseif shape == 3 %triangle
           xp_t = xp_d/(2*sqrt(3)) + dx;
       elseif shape == 4 %I-beam
           xp_t = xp_d/2 + dx;
       end
       xp = [xp_t xp_d material shape];
    end

end
```

And to execute the main file, we input the following commands to run the main file with our evaluation and perturbation function. We

```
>> xo = [0.0500     0.2000     3.0000     1.0000];
>> file_eval = 'objectiveBeam';
>> file_perturb = 'perturbBeam';
```

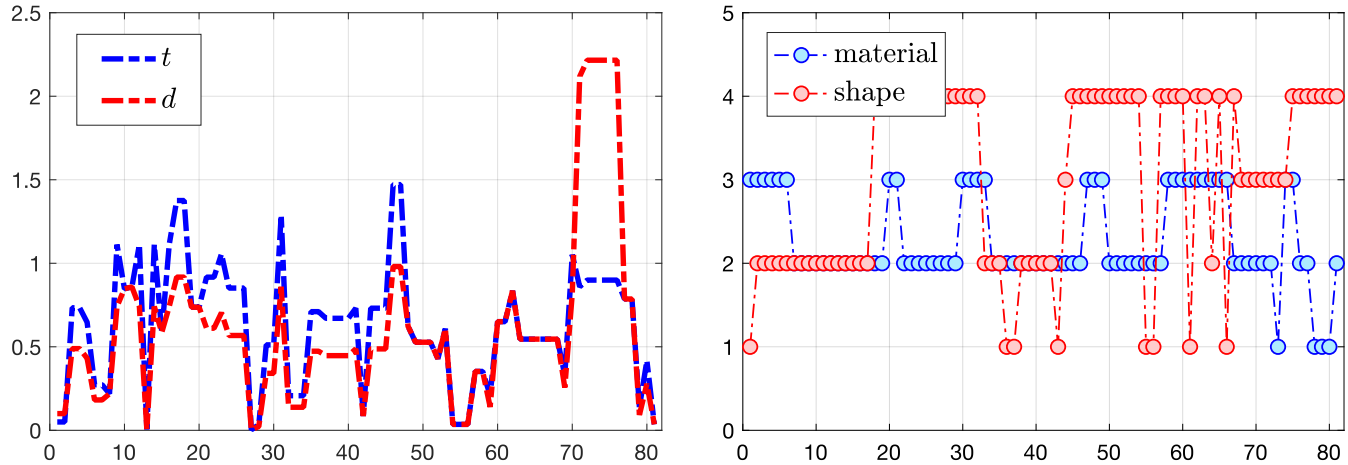


Figure 2: The changing history of the four design variables $[t, d, materials, shape]$ with the simulated annealing iterations.

```
>> [xbest,Ebest,xhist]=SA(xo,file_eval,file_perturb);
```

And by then we generate the eventual output with a history diagram as shown in Figure 2. The eventual optimal point is $[0.1044,\ 0.2572,\ 2.0000,\ 3.0000]$.

(c) Describe the optimal beam design in terms of geometry and material choice.

Solution: Based on our results, we know that for the optimized design, the thickness t and length d are: $t = d = 0.356$, in unit $[m]$. And the materials and shape obeys material $= 2$, which is Aluminum, and shape is I-beam. Obviously this violates our constraints on the geometric relation between t and d. This is because that the four constraints are imposed through Absolute Penalty Function methods, which does not guarantee all the constraints are perfectly satisfied. We thence increase the value of ρ_p to 10 times of its original value (manually), and the optimal point is manually changed as shown in Table 1.

(d) Explore how you can "tune" the SA algorithm (e.g., cooling/annealing schedule, initial guess, stopping criteria) and report your findings on their impact on the solution quality and computational time in a concise format.

Solution: Now, there are a couple of things that we can change in the SA algorithm: (1) the multiplier ρ_p we used in the Absolute Value Penalty Function (we used 1×10^7 in the original code). (2) the cooling methods (we used exponential cooling in the original code). (3) the initial point (we used $(0.05, 0.1, 3, 1)$ in the original design).

For the first parameters, we switched the value of ρ_p for a couple of values and generated the following results as in Table 1. From the results we first know that the *Absolute Value Penalty Function* penalty methods does not satisfies that the constraints are strongly enforced to optimiza-

ρ_p value	Final Design
10^3	(0.0522 0.1287 2.0000 3.0000)
10^4	(0.0500 0.1000 2.0000 1.0000)
10^5	(0.2035 0.5015 2.0000 3.0000)
10^6	(0.0486 0.1198 2.0000 3.0000)
10^7	(0.2435 0.2435 2.0000 4.0000)
10^8	(0.1844 0.1844 2.0000 4.0000)
10^9	(0.1495 0.1495 2.0000 4.0000)
10^{10}	(0.1230 0.3031 2.0000 3.0000)
10^{11}	(0.0500 0.1000 2.0000 4.0000)
10^{12}	(1.3878 0.9252 2.0000 1.0000)
10^{13}	(0.0703 0.1732 2.0000 3.0000)
10^{14}	(1.4589 0.9726 2.0000 1.0000)
10^{15}	(0.5949 1.4658 2.0000 3.0000)

Table 1: Different optimal points align with different ρ_p values.

Methods	Final Design
Exponential cooling	(0.0690 0.1700 2.0000 3.0000)
Linear cooling	(0.1199 0.1199 2.0000 4.0000)

Table 2: Different optimal points corresponding to the SA different cooling methods.

tion problems. Therefore some points in the optimal does not perfectly satisfy all the constraints. From Table 1 we can deduce the optimized materials and shape is $(2, 3)$, which are Aluminum and Triangle.

(2) We can also tune the cooling methods. Based on Table 1 we think $\rho_p = 10^{11}$ would be a good fit for the SA algorithm. The switched results are shown in Table 2.

Estimating the final design of the two cooling methods we can deduce that for Linear cooling the final mass is 11.6446 (here numerical value is sufficient to deduce optimality) and the exponential cooling mass is 8.4654. Hence we can deduce exponential cooling is the more preferred method (only for this case). Note that due to the constraints are imposed through the *Absolute Value Penalty Function* method, and since there are many constraints in this problem. We weight each constraints the same, thence there are some solutions (optimal point) does not perfectly satisfy the constraints.

(3) Later on, by tuning initial guesses, we generate Table 3 to show how different initial points generate different optimal points. Here, our strategy is we randomly give an initial point, and let the SA find the optimized point, and we take the optimal point as the initial again until the optimal point are no longer renewed.

(e) Based on your findings, discuss which SA tuning parameters most affect the solution quality and computation time.

Solution:

Initial Point	Final Design
(0.0100 0.2100 3.0000 1.0000)	(0.0100 0.2100 2.0000 3.0000)
(0.0100 0.2100 2.0000 3.0000)	(0.0745 0.0745 2.0000 4.0000)
(0.0050 0.1360 3.0000 4.0000)	(0.1441 0.3552 2.0000 3.0000)
(0.1441 0.3552 2.0000 3.0000)	(0.0649 0.0649 2.0000 4.0000)
(0.0500 0.2100 1.0000 1.0000)	(0.1504 0.3707 2.0000 3.0000)
(0.0160 0.2990 3.0000 4.0000)	(1.1067 0.7378 2.0000 1.0000)
(1.1067 0.7378 2.0000 1.0000)	(0.0330 0.0812 2.0000 3.0000)
(0.0330 0.0812 2.0000 3.0000)	(0.0317 0.0317 2.0000 4.0000)

Table 3: Different optimal points corresponds to different initial points.

Cooling method	CPU time
Exponential cooling	6.0984e+03
Linear cooling	6.5275e+03

Table 4: The results of the CPU time corresponding to different cooling methods.

Solution quality: From the above tables, it can be detected that cooling schedules (methods) strongly variate the results of the optimal point with same initial point and conditions.

CPU time: For computation time, to quantitatively describe the difference, we create the following tables: we first consider changing the cooling methods, and then we fix the method and change the initial points to see how the CPU computation time variate.

Based on the observations from Tables 4, 5, 6; we can deduce that initial points has very little influence on the CPU time, and for different ρ_p values, the change by the power of 10 only variate CPU time by the scale of 10^{-2}; but for different cooling schedules (methods), the CPU time are strongly variate by different methods. We therefore deduce that cooling methods playing a more significant role in both solution quality and computation time.

(f) What is your final suggested design?

Solution: As proposed and explained many times previously, due to the methods we chose and the paralleled multiple constraints, the final optimal points (suggested design may not perfectly satisfy all the constraints). Yet due to many attempts in the previous tables we can still pick a "acceptable" choice of design based on our general engineering background for the problem (a general "lightweight design" of the beam that can take the loading and satisfies the constraints can be accepted).

Initial point	CPU time
(0.0100 0.2100 3.0000 1.0000)	6.9235e+03
(0.0200 0.2800 3.0000 4.0000)	6.9407e+03
(0.0160 0.1200 1.0000 1.0000)	6.9614e+03
(0.0136 0.2300 1.0000 3.0000)	6.9740e+03
(0.0310 0.2970 2.0000 1.0000)	6.9875e+03

Table 5: The results of the CPU time corresponding to different initial points.

ρ_p value	CPU time
10^{10}	7.0713e+03
10^{20}	7.0820e+03
10^{30}	7.0879e+03
10^{40}	7.0937e+03
10^{50}	7.1016e+03
10^{60}	7.1794e+03
10^{70}	7.1848e+03
10^{80}	7.1905e+03
10^{90}	7.1951e+03

Table 6: The results of the CPU time corresponding to different ρ_p values.

During the attempts to optimize the design problem, we notice a design in Table 4 (the red marked one) presents a generally "good" design as we explained previously: (1) It satisfies all the engineering constraints. (2) The choice of material and shape agrees with most of the results as displayed in Tables 3, 4, 5, 6. (3) It already go through three optimization processes to this point. We therefore pick it as the final design. Hence, the final suggested design is thickness $t = 0.033$, in meters, cross section diameters $d = 0.0812$, in unit meters, shape as circle and materials as Titanium. **Based on these parameters and variables, the optimized mass is 1.9280kg.**

For this problem, I discussed with Mads and Gabrielle, and also asked Prof. Maha Haji for help. I also helped my teammates after I generated the results.

After a short discussion with Mads on Sunday, I found out that my methods may not be the best methods since enforcing the four constraints simultaneously to the objective may be the reason that some solutions violates the constraints, as I stated and explained previously. Therefore I think it is important to add this part to my Q2 as a patch to my individual part.

If we take all the methods same as previously, yet only changing the strategy that applying four constraints simultaneously to the evaluation, to only enforce the given two constraints in the instructions, and take the quadratic penalty function instead of absolute penalty function, the evaluation function writes:

```
function mass = objectiveBeam(x0)
Input = x0;
%% Input the vars.
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%% optimization constants
P = 1e3; l = .3;
%% judge loop for materials
if material == 1 % steel
    Em = 200e9;
```

```
  S_m = 300e6;
  rho_m = 7600;
elseif material == 2 % alum
  Em = 75e9;
  S_m = 200e6;
  rho_m = 2700;
elseif material == 3 % titanium
  Em = 120e9;
  S_m = 800e6;
  rho_m = 4400;
end
%% judge loop for shape
if shape == 1 % square
  y_s = d./2;
  A_s = d.^2 - (d - 2.*t).^2;
  I_s = (1/12).*(d.^4 - (d - 2.*t).^4);
elseif shape == 2 % circle
  y_s = d./2;
  A_s = (pi./4).*(d.^2 - (d - 2.*t).^2);
  I_s = (pi./64).*(d.^4 - (d - 2.*t).^4);
elseif shape == 3 % triangle
  y_s = (1 - 1./(2.*sqrt(3))).*d;
  A_s = (sqrt(3)./4).*(d.^2 - (d - 2.*sqrt(3).*t).^2);
  I_s = (sqrt(3)./96).*(d.^4 - (d - 2.*sqrt(3).*t).^4);
else % I-beam
  y_s = d./2;
  A_s = 3.*d.*t - 2.*t.^2;
  I_s = (t./12).*((d - 2.*t).^3 + 2.*d.*t.^2 ...
            + 6.*d.*(d - t).^2);
end

%% judge if constraints are violated

sigma_max = (P.*l.*y_s)./(I_s);
delta = (P.*l.^3)./(3.*Em.*I_s);

rho_p = 1e50; % penalty function --> this value is very important!!

P = rho_p * max([0, sigma_max-0.9*S_m])^2 + rho_p * max([0, delta-0.003])^2;
% P = rho_p * abs(sigma_max-0.9*S_m) + rho_p * abs(delta-0.003) + rho_p * abs(2*t - d) + rho_p *
    abs(d - l);
mass=rho_m*A_s*l + P;

% relation of mass to t & d
end
```

And we also do a slight change for the constraint function, by enforcing different perturbation

for variables t and d:

```
function [xp]=perturbBeam(x0)
Input = x0;
l = .3;
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%
num = randi([1 4]);
%%
    if num == 3
        matrl = randi([1 3]);
        xp = [t d matrl shape];
    end

    if num == 4
        shap = randi([1 4]);
        xp = [t d material shap];
    end
%%
xp_t=x0(1);
xp_d=x0(2);
xlb=0;
xub=1;
%

    if num == 1
        xp = [t d material shape];
        if shape==3
            xub=d/(2*sqrt(3));
            xlb=0.01;
        else
            xub=d/2;
            xlb=0.01;
        end
        indx=round(1);
        dx=(xub-xlb)*rand(1)+xlb;
        xp(indx)=dx;
    end
    if num == 2
        xp = [t d material shape];
        if shape==3
            xub=l;
            xlb=t*(2*sqrt(3));
        else
            xub=l;
            xlb=t*2;
        end
        indx=round(2);
```

```
        dx=(xub-xlb)*rand(1)+xlb;
        xp(indx)=dx;
    end
end
```

In this way as many tries the output won't violate the constraints:

```
...
Counter: 158 Temp: 591.3979 P(dE)= 0.99141
Counter: 158 Temp: 591.3979 Accepted inferior configuration (uphill)
Counter: 158 Temp: 591.3979 Need to reach equilibrium at this temperature
Counter: 158 Temp: 591.3979 Perturbing configuration

xp =

    0.0238    0.1268    1.0000    2.0000

Counter: 159 Temp: 591.3979 P(dE)= 0.99387
Counter: 159 Temp: 591.3979 Accepted inferior configuration (uphill)
Counter: 159 Temp: 591.3979 System nearly frozen

tt =

  230.8100

Counter: 159 Temp: 532.2581 System frozen, SA ended
Best configuration:

xbest =

    0.0197    0.0815    2.0000    3.0000

Lowest System Energy: 2.2659
```

As stated, this is a patch or supplementary to my original solution in Q2, and I hope this can provide a reference.

Q3. Heuristic Optimization (PSO or GA)

Problem B – Genetic Algorithm

Consider the problem

$$\min f(x_1, x_2, x_3) = |x_1|^{0.8} + 5\sin(x_1^3) + |x_2|^{0.8} + 5\sin(x_2^3) + |x_3|^{0.8} + 5\sin(x_3^3)$$
$$\text{subject to} \quad -5 \le x_i \le 5, \text{ for } i = 1, 2, 3$$

(a) Use the Genetic Algorithm to solve this problem.

Solution: We apply the Optimization Toolbox in MATLAB®:

We first define the objective function in MATLAB, nominated as `objectiveFcn`:

```
function f = objectiveFcn(x,a)
a = 0.8;
f = abs(x(1)).^a + 5*sin(x(1) .^3) + abs(x(2)).^a + 5*sin(x(2) .^3) + abs(x(3)).^a + 5*sin(x(3)
    .^3);
end
```

And running the genetic algorithm optimization by calling the MATLAB built-in function `ga`, by recalling the previously defined objective function `objectiveFcn`, first select the `nonlinear` class for `Objective`, and define the lower and upper bounds `lb = -5` & `ub = 5`:

```
clear; clc
a = 0.8;
% Pass fixed parameters to objfun
objfun5 = @(x)objectiveFcn(x,a);

% Set nondefault solver options
options = optimoptions('ga', ...
        'CrossoverFcn',{@crossoverheuristic,1.6},'Display',...
        'iter', ...
        'FunctionTolerance', 1e-6, ...
        'PopulationSize', 50, ...
        'CrossoverFraction', 0.7,...
        'MaxGenerations', 2000,...
        'ConstraintTolerance', 1e-6,...
        'MutationFcn',{@mutationadaptfeasible});
bo = 3;
% Solve
[solution,objectiveValue] = ga(objfun5,bo,[],[],[],[],repmat(-5,bo,1),...
    repmat(5,bo,1),[],[],options);
% Clear variables
clearvars objfun5 options5
% Display the results
solution
objectiveValue
```

The results of the optimization is shown in Figure 3. The optimal point is obtained as $x_1 = x_2 = -1.1527$ & $x_3 = 1.6745$, as shown in sub figure **D** in Figure 3.

(b) Explore various "tuning" parameters (e.g., number of generations, population size, mutation rate) and report your findings on their impact on the solution quality and computational time in a concise format.

Solution: Based on the instructions and the functionality of the MATLAB `ga` function, the

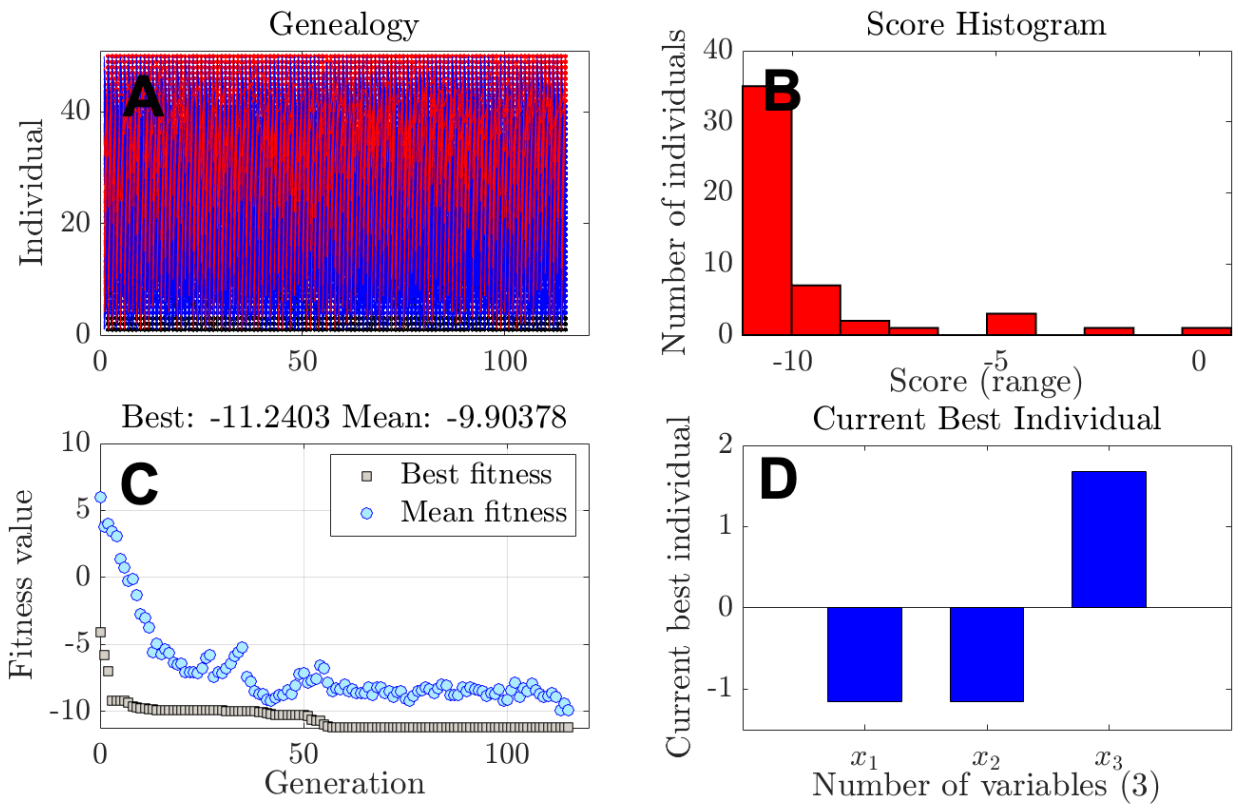


Figure 3: Calculation results of optimization based on genetic algorithm.

Function Tolerance	CPU time	Solution	Objective
10^{-3}	1.3907e+04	(-1.1527 -1.1527 -1.1528)	-11.6273
10^{-4}	1.3968e+04	(-1.1529 -1.1527 1.6745)	-11.2403
10^{-5}	1.3990e+04	(-1.1527 1.6745 -1.1527)	-11.2403
10^{-6}	1.4013e+04	(-1.1527 -1.1527 1.6745)	-11.2403
10^{-7}	1.4029e+04	(-1.1527 -1.9868 3.4872)	-9.4272
10^{-8}	1.4097e+04	(-1.1527 1.6745 -1.1527)	-11.2403
10^{-9}	1.4111e+04	(-1.1527 -1.1527 -1.1527)	-11.6273

Table 7: The CPU time, solution and objectives corresponding to function tolerance.

options for tuning the parameters include (1) Function Tolerance; (2) Population Size; (3) Crossover Fraction; (4) Max Generations; and (5) Constraint Tolerance. According to multiple attempts, we observe that usually the optimization stops at about 100 iterations, which is evidently smaller than the original max ierations, 2000. And due to we don't have a highly nonlinear constraint in this problem, taking the constraint tolerance at a low value (10^{-6}) and fix it is reasonable.

Now, by tuning the above four parameters, we provide the solutions, the objective values and CPU time as shown in the following tables (Tables 7, 8, 9).

We here took similar strategies as in Q2: we first fix the rest parameters and only adjust one to check how it influence the solutions and check all the adjustable parameters manually. For function tolerance, we first fix population size = 50, crossover fraction = 0.7, max generations = 2000, and

Population Size	CPU time	Solution	Objective
50	1.4170e+04	(1.6745 1.6745 -1.1527)	-10.8534
60	1.4193e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
70	1.4207e+04	(-1.9868 -1.1527 -1.1527)	-11.0193
80	1.4225e+04	(-1.1527 -1.9868 1.6745)	-10.6323
90	1.4249e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
100	1.4260e+04	(-1.1528 -1.1527 -1.1527)	-11.6273

Table 8: The CPU time, solution and objectives corresponding to population size.

Crossover Fraction	CPU time	Solution	Objective
0.5	1.4921e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
0.6	1.4935e+04	(-2.7330 -1.1527 1.6745)	-10.1293
0.7	1.4946e+04	(-1.1527 1.6745 -1.1527)	-11.2403
0.8	1.4958e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
0.9	1.4968e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
1	1.4978e+04	(-1.1527 -1.1527 1.6745)	-11.2403

Table 9: The CPU time, solution and objectives corresponding to crossover fraction.

constraint tolerance = 1e-6, and change the function tolerance to generate Table 7. From Table 7 we can observe that for each change of function tolerance of 10^{-1} there is a change of approximately 0.05×10^4 for CPU time, and there are some evident solution change for x_3 when function tolerance is about 10^{-8}.

We then fix the function tolerance to 10^{-9} and fix others to the same and change the population size, we then generate Table 8.

Similar as before, we fix the population size to 100 and fix others to the same and change the population size, we then generate Table 9.

(c) Based on your findings from (b), discuss which GA tuning parameters most affect the solution quality and computation time.

Solution: Based on our test data from Tables 7, 8, 9, we can conclude that the function tolerance influence both the CPU time and solution quality most evidently (Table 7: evident solution change from 10^{-8} to 10^{-10}.)

Appendix. Supplementary Code & Data

Main function of SA (SA.m):

```
Initial configuration:
function [xbest,Ebest,xhist]=SA(x0,file_eval,file_perturb);
% [xbest,Ebest,xhist]=SA(x0,file_eval,file_perturb,options);
%
% Single Objective Simulated Annealing (SA) Algorithm
%
% This function is a generic implementation of the single objective
% Simulated Annealing (SA) algorithm first proposed by Kirkpatrick,
% Gelatt and Vecchi. The algorithm tries to improve upon an initial
% configuration, x0, by evaluating perturbed configurations. When the
% system reaches the "frozen" state, the algorithm stops and the best
% configuration and search history are returned. The user can choose
% from one of two cooling schedules: linear or exponential.
%
% Input:
% x0            initial configuration of the system (a row vector)
% file_eval     file name (character string) of configuration evaluator;
%               assumes that E='file_eval'(x) is a legitimate function
%               call; set up function such that (scalar) output E will be
%               minimized.
% file_perturb file name (character string) of configuration perturbator;
%               assumes that xp='fname_perturb'(x) is a legitimate function
%               call. This function creates a "neighboring" configuration.
% options       algorithm option flags. Uses defaults, [ ], if left blank
%    (1)        To - initial system temperature - automatically determined if
%               left blank ([]). To should be set such that the expression
%               exp(-E(x0)/To)>0.99 is true, i.e. the initial system is "melted"
%    (2)        Cooling Schedule: linear=1, exponential=[2]
%    (3)        dT Temp. increment, e.g. [dT=0.9] for exp. cooling Tk=dT^k*To,
%               abs. temperature increment for linear cooling (Tk+1=Tk-dT);
%    (4)        neq = equilibrium condition, e.g. number of rearrangements
%               attempted to reach equilibrium at a given temperature, neq=[5]
%    (5)        frozen condition - sets up SA exit criterion
%               nfrozen = non-integer, e.g. 0.1 SA interprets this numbers as Tmin,
%               the minimum temperature below which the system is frozen.
%               nfrozen = integer ,e.g. 1,2..  SA interprets this as # of successive
%               temperatures for which the number of desired acceptances defined
%               under options(4) is not achieved, default: nfrozen=[3]
%    (6)        set to 1 to display diagnostic messages (=[1])
%    (7)        set to 1 to plot progress during annealing (=[0])
%
% Output:
% xbest         Best configuration(s) found during execution - row vector(s)
% Ebest         Energy of best configuration(s) (lowest energy state(s) found)
```

```
% xhist        structure containing the convergence history
%   .iter      Iteration number (number of times file_eval was called)
%   .x         current configuration at that iteration
%   .E         current system energy at that iteration
%   .T         current system temperature at that iteration
%   .k         temperature step index k
%   .C         specific heat at the k-th temperature
%   .S         entropy at the the k-th temperature
%   .Tnow      temperature at the k-th temperature step
%
% User Manual (article):   SA.pdf
%
% Demos:         SAdemo0 - four atom placement problem
%                SAdemo1 - demo of SA on MATLAB peaks function
%                SAdemo2 - demo of SA for Travelling Salesman Problem (TSP)
%                SAdemo3 - demo of SA for structural topology optimization
%                SAdemo4 - demo of SA for telescope array placement problem
%
% dWo,(c)  MIT 2004
%
% Ref: Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P., "Optimization
% by Simulated Annealing", Science, Vol. 220, Number 4598, pp. 671-680, May
% 1983


%
dT=0.9;
neq=5;
nmax=neq*round(sqrt(max(size(x0)))); % nmax - maximum number of steps at one temperature, while
%                                       trying to establish thermal equilibrium
%
nfrozen=3;
diagnostics=1;
eval(['Eo=' file_eval '(x0);']);
To=abs(-Eo/log(0.99));

if nfrozen==round(nfrozen)
    % nfrozen is integer - look for nfrozen successive temperatures without
    % neq acceptances
    Tmin=0;
else
    Tmin=nfrozen; nfrozen=3;
end



% Step 1 - Show initial configuration
if diagnostics==1
```

```
disp('Initial configuration: ')
x0
end

% Step 2 - Evaluate initial configuration
eval(['Eo=' file_eval '(x0);']);
counter=1;
xnow=x0; Enow=Eo; nnow=1;
xhist(nnow).iter=counter;

xhist(nnow).x=x0;
xhist(nnow).E=Enow;
xhist(nnow).T=To;
%  still need to add .S         current entropy at that iteration
xbest=xnow;
Ebest=Enow;
Tnow=To;
if diagnostics==1
    disp(['Energy of initial configuration Eo: ' num2str(Eo)])
end

frozen=0;  % exit flag for SA
naccept=1; % number of accepted configurations since last temperature change
Tlast=1;   % counter index of last temperature change
k=1;       % first temperature step
ET=[];     % vector of energies at constant system temperature

% start annealing
while (frozen<nfrozen)&(Tnow>Tmin)

%Step 3 - Perturb xnow to obtain a neighboring solution

if diagnostics
    disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow) ' Perturbing configuration'])
end

eval(['xp=' file_perturb '(xnow);']);
xp
%Step 4 - Evaluate perturbed solution
eval(['Ep=' file_eval '(xp);'])
counter=counter+1;

%Step 5 - Metropolis Step

dE=Ep-Enow; % difference in system energy
PdE=exp(-dE/Tnow);
if diagnostics
    disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow) ' P(dE)= ' num2str(PdE)])
```

```
end

%Step 6 - Acceptance of new solution
if dE<=0 % energy of perturbed solution is lower , automatically accept
    nnow=nnow+1;
    xnow=xp; Enow=Ep;
    xhist(nnow).iter=counter;
    xhist(nnow).x=xp;
    xhist(nnow).E=Ep;
    xhist(nnow).T=Tnow;
    naccept=naccept+1;
   if diagnostics
   disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Automatically accept better configuration (downhill)'])
   end

else
   % energy of perturbed configuration is higher, but might still accept it
   randomnumber01=rand;
    if PdE>randomnumber01
        nnow=nnow+1;
        xnow=xp; Enow=Ep;
        xhist(nnow).iter=counter;
        xhist(nnow).x=xp;
        xhist(nnow).E=Ep;
        xhist(nnow).T=Tnow;
      if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Accepted inferior configuration (uphill)'])
      end

    else
        % keep current configuration
        xnow=xnow;
        Enow=Enow;
      if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Kept the current configuration'])
      end
    end
end
       ET=[ET; Enow];


if Enow<Ebest
    % found a new 'best' configuration
    Ebestlast=Ebest;
    Ebest=Enow;
```

```
    xbest=xnow;
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' This is a new best configuration'])
    end

elseif Enow==Ebest
    same=0;
    for ib=1:size(xbest,1)
        if xbest(ib,:)==xnow
             if diagnostics
          disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Found same best configuration'])
             end
        same=1;
        end
    end

     if same ==0
       Ebestlast=Ebest;
       Ebest=Enow;
       xbest=[xbest ; xnow];
        if diagnostics
          disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Found another best configuration'])
        end
    end
end

%Step 7 - Adjust system temperature
Told=Tnow;
if (naccept<neq)&(counter-Tlast)<nmax
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Need to reach equilibrium at this temperature'])
    end
    % continue at the same system temperature
elseif (naccept<neq)&(counter-Tlast)>=nmax
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' System nearly frozen'])
    end

    Eavg=mean(ET);
    Evar=mean(ET.^2);
    C=(Evar-Eavg^2)/Tnow^2;     % specific heat
    S=log(nmax*length(unique(ET))/length(ET));
    xhist(k).k=k;
    xhist(k).C=C;
```

```
    xhist(k).S=S;
    xhist(k).Tnow=Tnow;


    frozen=frozen+1;
    Tlast=counter;
    naccept=0;


%     switch schedule
%         case 1
%             % linear cooling
%             Tnow=Tnow-dT;
%               if Tnow<0
%                   frozen=nfrozen; %system temperature cannot go negative, exit
%               end
%         case 2
            % exponential cooling
              Tnow=dT*Tnow;
%         case 3
%               Tindex=Tindex+1;
%               if Tindex>size(Tuser,1)
%                   frozen=nfrozen; % have run through entire user supplied cooling schedule
%               else
%               Tnow=Tuser(Tindex,1);
%               neq=Tuser(Tindex,2);
%               end
%         otherwise
%               disp('Erroneous cooling schedule choice - option(2) - illegal')
%     end


    k=k+1;

    ET=[];

 elseif (naccept==neq)
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' System reached equilibrium'])
    end

    Eavg=mean(ET);
    Evar=mean(ET.^2);
    C=(Evar-Eavg^2)/Tnow^2;     % specific heat
    S=log(nmax*length(unique(ET))/length(ET));
    xhist(k).k=k;
    xhist(k).C=C;
    xhist(k).S=S;
```

```
    xhist(k).Tnow=Tnow;


    Tlast=counter;
    naccept=0;

%     switch schedule
%         case 1
%             % linear cooling
%             Tnow=Tnow-dT;
%               if Tnow<0
%                   frozen=nfrozen; %system temperature cannot go negative, exit
%               end
%         case 2
            % exponential cooling
              Tnow=dT*Tnow;
%         case 3
%             % user supplied cooling
%               Tindex=Tindex+1;
%               if Tindex>size(Tuser,1)
%                   frozen=nfrozen; %have run through entire user supplied cooling schedule
%               else
%               Tnow=Tuser(Tindex,1);
%               neq=Tuser(Tindex,2);
%               end
%
%         otherwise
%               disp('Erroneous cooling schedule choice - option(2) - illegal')
%     end


     k=k+1;

  ET=[];
  % xbest];

 end

end %while (frozen<nfrozen)&(Tnow>tmin)


    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' System frozen, SA ended'])
       disp(['Best configuration: '])
       xbest
       disp(['Lowest System Energy: ' num2str(Ebest) ])
    end
```