MANE 6710 - Numerical Design Optimization Lab 1

Human 6966

September 24 2024

Table of Contents:

Executive Summery **3**

1 Analysis Introduction **3**
1.1 Methodology . . . 3
1.2 Assumptions . . . 3
1.3 Limitations . . . 4

2 Parameterization of the Geometry **4**

3 Optimization Method and Limitations **4**

4 Optimization Problem **5**

5 Results **6**

6 Conclusions **8**

References **8**

7 Appendix **9**
7.1 calcHeight.m . . . 9
7.2 objective.m . . . 10
7.3 run_opt.m . . . 11
7.4 plots.m . . . 13
7.5 fmincon Output . . . 15

Executive Summery

Heat exchangers are an important part of many modern systems from home air conditioning and heating, to car engines, to industrial processing facilities and power plants. Heat exchangers are so ubiquitous because they allow for the transfer of heat from one fluid to another without the fluids mixing. This is important because it allows for the design of modular closed-system heaters and coolers (no mass moves in or out of the system) that can affect other systems when coupled. This enables the design problems for complex systems to be broken down into the simpler design problems of subsystems. This lab focuses on choosing and using optimization algorithms in Matlab (fmincon()) to optimize the profile of a heat exchanger to transfer hear from hot water to air. A two-dimensional discretized version of Fourier's law was chosen to analyze the thermodynamic system, and the SQP algorithm for fmincon() was chosen to optimize the profile of the heat exchanger. The SQP algorithm was successfully able to optimize the heat exchanger within the problem constraints by maximizing the area of the heat exchanger touching the air.

1 Analysis Introduction

1.1 Methodology

To analyze the steady-state heat flux per unit length through the heat exchanger, the two-dimensional heat equation (Equation 1) was used to find the temperature gradient through the heat exchanger.

$$\nabla(k\nabla T) = 0, \forall x \in \Omega \tag{1}$$

With the boundary conditions:

$$T(x, y = 0) = T_{bottom}$$
$$T(x, y(x)) = T_{top}$$
$$T(x, y) = T(x + L, y)$$

To solve this equation numerically, Equation 1 was approximated using a finite-Volume discretization with a unit length resulting in Equation 2. Which when applied to a mesh, can be used to solve for the temerature gradient.

$$\int\int_V \nabla(k\nabla T)dV \approx \sum_{i=0}^{N_x-1}\sum_{j=0}^{N_y} k\frac{T_{i+1,j} - T_{i,j}}{\Delta x}\Delta y = 0 \tag{2}$$

From this temperature gradient, the heat flux/unit length could be calculated for the inlet surface using Fourier's law of conduction at $y = 0$ (Equation 3).

$$q = \int_A k\nabla T\vec{n}dA \approx \sum_{i=0}^{N_x-1} k\frac{T_{i+1,0} - T_{i,0}}{\Delta x}\Delta y \tag{3}$$

1.2 Assumptions

To simplify the physical model and fundamental equations, the following assumptions were made:

1. Constant water temperature in contact with the heat exchanger ($T_{water} = 90°\text{C}$)
2. Constant air temperature in contact with heat exchanger boundary ($T_{air} = 20°\text{C}$)
3. The temperature of the surface of the heat exchanger is the same as the fluid it is in contact with:

- $T_{top} = T_{air}$
- $T_{bottom} = T_{water}$

4. The heat exchanger is made of a steel alloy with a thermal conductivity of 20 W/(mK)
5. Steady state heat transfer
6. Negligable edge effects
7. No temperature gradient along the extruded length of the heat exchanger
8. No heat produced by the heat exchanger

1.3 Limitations

These assumptions limit this analysis method to static systems where any protrusions from the surface of the heat exchanger are sufficiently far apart to prevent the temperature of the fluids between them from changing relative to the rest of the fluid. Additionally, the model doesn't consider convection, so it only works when you know the temperature of both sides of the heat exchanger. Finally, the assumptions about constant fluid temperature, no temperature gradient along the extrusion direction, and negligible edge effects introduce error into the analysis, as these conditions would not be met under real-world circumstances.

2 Parameterization of the Geometry

The function chosen to define the shape of the top of the heat exchanger was

$$h(x) = a_0 + \sum_{k=1}^{n} a_k sin(\frac{2\pi kx}{L}) \tag{4}$$

because the function is smooth, periodic, and has many possible shapes. Equation 4 was then discretized into a two-dimensional mesh using Equation 5.

$$H(i,j) = h(i\frac{L}{N_x})\frac{j}{Ny} \tag{5}$$

3 Optimization Method and Limitations

The SQP (Sequential Quadratic Programming) Algorithm was chosen for fmincon() (Matlab's nonlinear optimization problem solver [1]) to solve the minimization problem. The SQP algorithm uses a quasi-Newton update method to approximate the Hessian, which is used to solve the quadratic subproblems, which are linearized around the current state. Then the algorithm does a line search to determine the step size in the direction of the solution for each node in the mesh. The algorithm iterates through the updated states until the solutions to the quadratic subproblems converge into a smooth gradient. To perform this analysis, the following assumptions were made:

1. Constraints are continuously twice differentiable
2. The objective function is continuously twice differentiable
3. There is a 'good' initial guess
4. There is a 'good' mesh

The methodology and assumptions impose the following limitations on this analysis method:

1. Due to the derivative assumptions, smooth objective and model functions are required
2. Due to ending conditions, a smooth mesh is required
    - No discontinuities can be in the mesh
    - The mesh must be fine enough to capture the gradient
3. Due to the gradient-driven update method, the solution may not recover from a step outside the constraints
    - Initial guess should start solver inside constraints (algorithm may or may not work otherwise)
    - A 'bad' step can put algorithm outside constraints
4. The algorithm won't know if the minimum it found is a local or global minimum
5. If the objective function behavior cannot be adequately captured in a localized linear model, this can create 'bad' steps due to deviation between actual and linearized models

4 Optimization Problem

The objective of this lab is to apply the priciples learned in lecture to optimize the shape of a simple heat exchanger model within the following criteria and constraints[2]:

1. The design is to be two-dimensional and:
    - Can be smoothly repeated horizontally to get heat exchangers of desired widths
    - Has a width of L = 5 cm (Figure 1)
2. The heat exchanger should heat air using hot water
    - The steady-state temperature of the water is 90°C
    - The steady-state temperature of the air is 20°C
3. The side of the heat exchanger that touches the water is flat and will not be altered.
4. The air-side shape can take on any shape provided:
    - The shape is a function of horizontal distance
    - The thickness of the heat exchanger is at least 1 cm at any point along the horizontal (Figure 1 h_{min})
    - The thickness of the heat exchanger is no more than 5 cm at any point along the horizontal (Figure 1 h_{max})
    - The shape is accurately captured by the heat-flux analysis method.

The objective of this project is to optimize the shape of the heat exchanger touching the air to maximize the heat flux per unit length transferred from the water to the air. The model geometry was parameterized and discretized (see section 2 for more details) and applied to a descretized version of Fourier's law to analyze the problem. The method chosen for the minimization problem was the SQP algorithm for the fmincon Matlab function (see section 3 for more details).

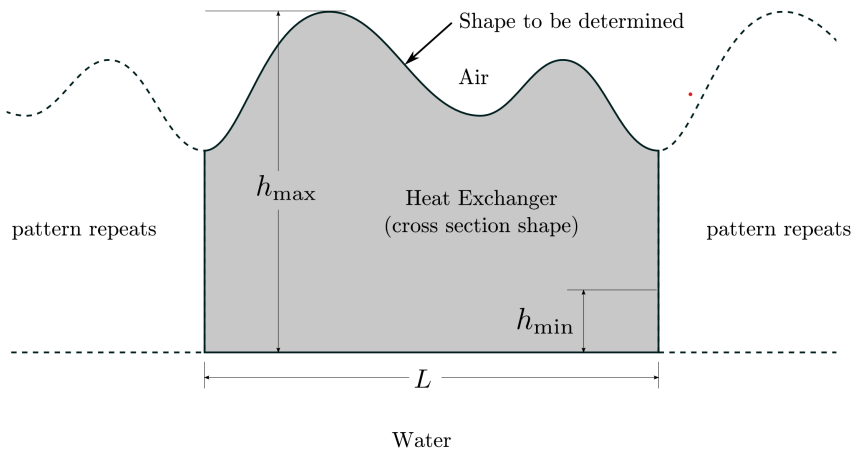

Figure 1: Heat Exchanger Problem Illistration ([2] Figure 1)

5 Results

The optimized shape of the heat exchanger with the chosen parameterization and the number of coefficients (16) is shown in Figure 2. To meet the minimum heat flux requirements from the rubric, the shape function requires more than ten design variables and Proffessor Hicken stated in class that less than twenty design variables should be used. With this in mind, sixteen design variables was chosen and the resulting design met requirements. This result makes sense, because it shows that the heat flux through the bottom of the heat exchanger is at its maximum when the surface area of the top of the heat exchanger is maximized. This is why many heat exchangers like the heat sinks used in computers use fins.

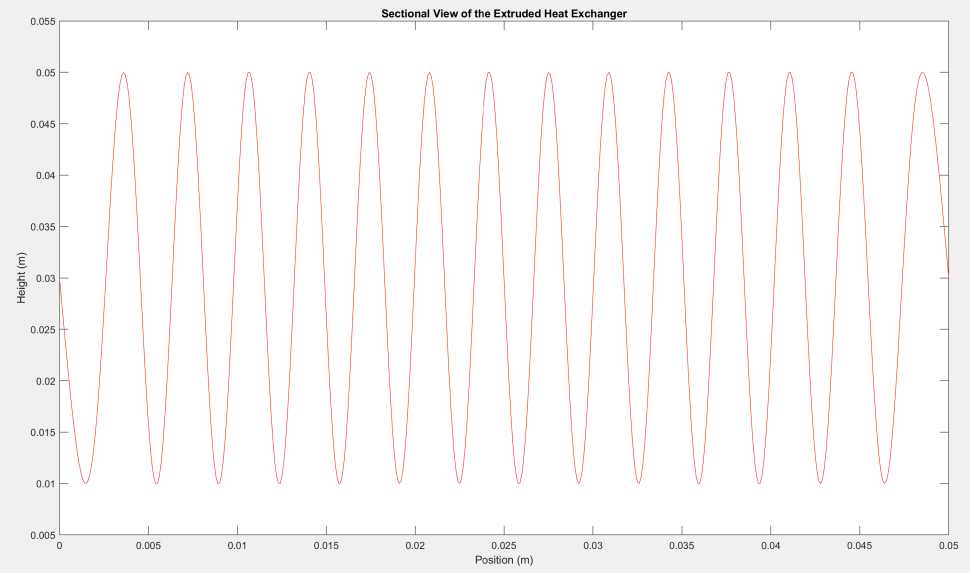

Figure 2: Plot of the Heat Exchanger Cross Section

The feasibility graph shows that the feasibility score of the current solution decreased rapidly over the first iteration and then stabilized over the remaining iterations as the First Order Optimality improved. The First Order Optimality graph shows a slow improvement over the iterations until the solver converged on a minimum causing a rapid improvement (see Figure 3 both Feasibility and First-Order Optimality decreased by sixteen orders of magnitude over the itterations).

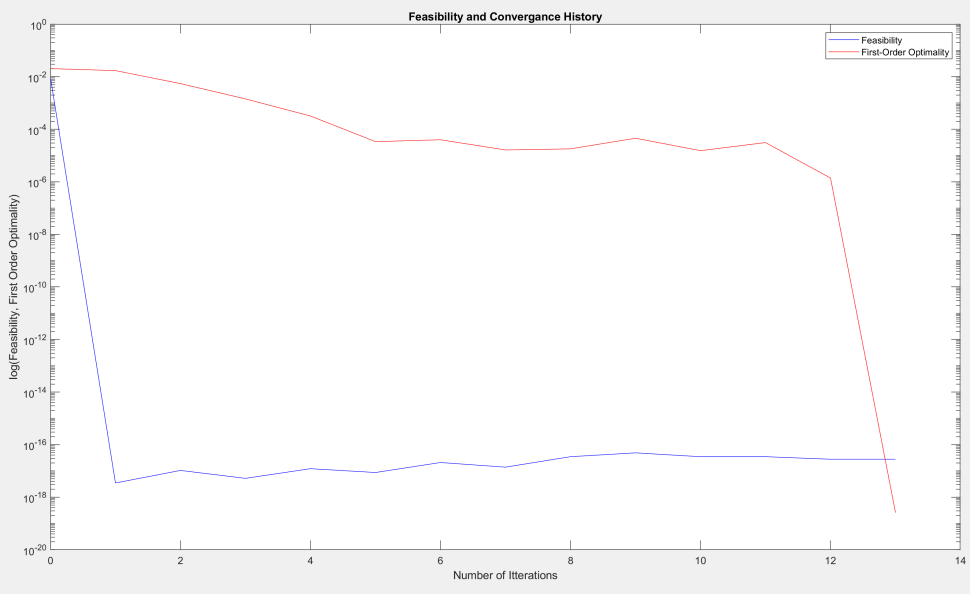

Figure 3: The Feasibility and Convergence Plots

The convergence plot shown in Figure 4 was calculated for the optimized shape function h(x) from Figure 2. The plot shows jagged movement that quickly smooths out to an exponential curve approaching a value of about 4.5 kW/m as the number of segments increases. This behavior for small numbers of segments makes sense because for small numbers of segments the mesh doesn't adequately represent the temperature gradient so it doesn't behave as expected. Additionally, as the number of segments increases the calculated flux converges on the analytical solution as expected, because the series approximation of the integral becomes more accurate. The acceptable percent deviation is set to 5% of the flux value as $N_x \to \infty$ because there are several curve fit equations for convection with similar accuracies.

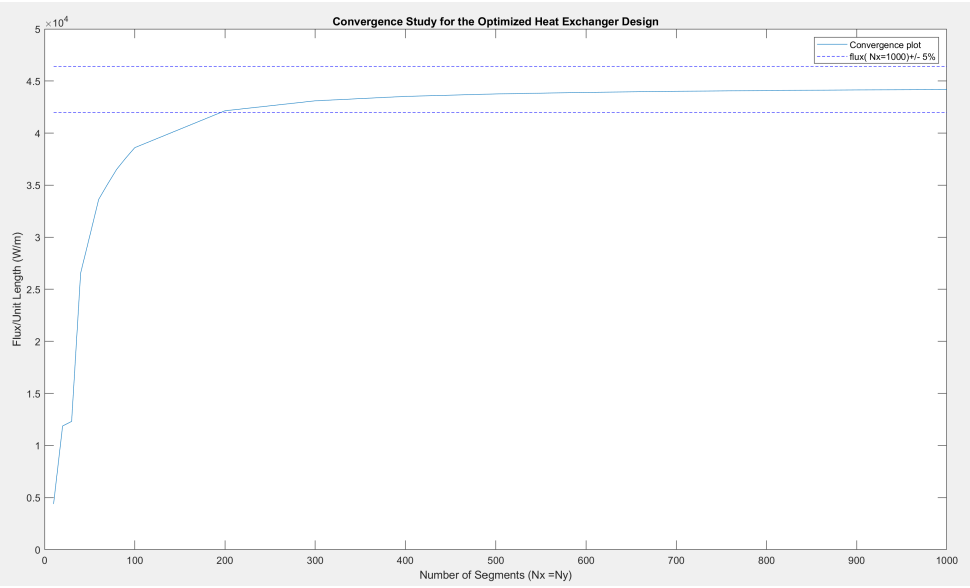

Figure 4: Plot of the Flux vs Nx

6 Conclusions

The implementation of the SQP optimization algorithm for this lab was successful. The resulting shape function for the optimized heat exchanger section profile made sense because it maximized the flux by maximizing the area in contact with the working fluids, which is how many similar devices are designed. The resulting calculated heat flux from the optimized heat exchanger was more than four times greater than the baseline design ($44190 > 28000$ W/m) while remaining within the problem constraints.

Basic optimization algorithms, such as the ones used in this lab, can be powerful tools for understanding the complex interactions between different design variables in engineering design problems and balancing the design variables to arrive at an optimal solution. Understanding how these algorithms work and the trade-offs between accuracy and run time shown in the convergence study is important when utilizing these algorithms to solve problems.

References

[1] *Fmincon*, MathWorks Help Website, 2023.

[2] D. J. Hicken, *Mane 4280/6710: Numerical design optimization lab 1*, MANE 6710 Course LMS Website, September, 2024.

7 Appendix

7.1 calcHeight.m

```
function calcHeight=calcHeight(a,L,X)
% Creates the height of the
% Inputs:
%   L - length of domain in x direction
%  a - the coefficient vector for the shape of the heat exchanger
%  X - the position array
% Outputs:
%   h - the height array that defines the shape of the upper part of the
    heat exchanger
%
-------------------------------------------------------------------------

%initialize the height array and set it to a0
h=ones(size(X))*a(1);
%loops through every position in X to calculate the height of the
for i = 2:max(size(X))
        %loops through the coefficients in a to calculate the h at a
        % specific position X(i)
         for j=1:max(size(a))
             %at each position X(i) and coefficient a(j) add the value of
             %the sinusoidal function a*sin((2*pi*(j-1)*X)/L)
             h(i) = h(i)+a(j)*sin(2*pi*(j-1)*X(i)/L);
         end
end
%h(1)=a(1);
%return the value of the height array
calcHeight=h;
end
```

7.2 objective.m

```
function [f]=objective(a,Nx, Ny, L, kappa, Ttop,Tbot)
% Solves for the heat flux using the CalcFlux Function, and returns the
% inverse heat flux per unit length from the water to the air
% Inputs:
%   L - length of domain in x direction
%  a - the coefficient vector for the shape of the heat exchanger
%   Nx - number of elements along the x direction
%   Ny - number of elements along the y direction
%   kappa - thermal conductivity
%   Ttop - the ambient air temperature along the top of the domain
%   Tbot - the fluid temperature along the bottom of the domain
% Outputs:
%   f -the inverse of heat flux per unit length out of top/bottom boundary
% Notes:
%   The domain has straight sides on the left and right and bottom.  The
%   boundary conditions are periodic on the left and right.  Prescribed
%   (i.e. Dirichlet) boundary conditions are applied along the top and
%   bottom.  The flux is computed along the bottom boundary.
%
```

```
%create the position array
X = [0:L/Nx:L].';
%create the height array that defines the upper part of the heat exchanger
h=calcHeight(a,L,X);
%calculate the heat flux/unit length using the CalcFlux function
[flux,T,dTdx,xy] = CalcFlux(L, h, Nx, Ny, kappa, Ttop, Tbot);
%calculate the inverse of the flux and set it to f
f=1/flux;
end
```

7.3 run_opt.m

```
function [a]=run_opt(Nx,Ny,a0)
% Sets up and runs the optimzation algorithm fmincon
% Inputs:
%   Nx - number of elements along the x direction
%   Ny - number of elements along the y direction
%   a0 - the initial guess for the coefficients
% Outputs:
%   a - the resulting optimized coefficients from the optimizer
%
% Notes:
%   The domain has straight sides on the left and right and bottom.  The
%   boundary conditions are periodic on the left and right.  Prescribed
%   (i.e. Dirichlet) boundary conditions are applied along the top and
%   bottom.  The flux is computed along the bottom boundary.
%
```

```
%temperature of the water
Twater=90;
%temperature of the air
Tair=20;
%length of the repeating segment
L=.05;
%thermal conductivity of the heat exchanger
kappa=20;
%create the position array
x = [0:L/Nx:L].';
%set fmincon options
options = optimoptions('fmincon','Display','iter','Algorithm','sqp');



%create anonomus function
func = @(a) objective(a,Nx, Ny, L, kappa, Tair,Twater);

%define size constraints
nvar = size(a0,1); % number of design variables
Aineq = zeros(2*(Nx+1),nvar);
bineq = zeros(2*(Nx+1),1);
for i = 1:(Nx+1)
  % first, the upper bound
   Aineq(i,1) = 1.0; % this coefficient corresponds to variable a_1
   for k = 2:nvar
     Aineq(i,k) = sin(2*pi*(k-1)*x(i)/L); % this coefficient corresponds to
         variable a_k
   end
   bineq(i) = 0.05; % the upper bound value
```

```
46    % next, the lower bound; we use ptr to shift the index in Aineq and
          bineq
47    ptr = Nx+1;
48    Aineq(ptr+i,1) = -1.0; % note the negative here!!! fmincon expects
          inequality in a form A*x < b
49    for k = 2:nvar
50      Aineq(ptr+i,k) = -sin(2*pi*(k-1)*x(i)/L); % again, a negative
51    end
52    bineq(ptr+i) = -0.01; % negative lower bound
53  end
54
55  %call fmincon and store the output in a
56  a = fmincon(func,a0,Aineq,bineq,[],[],[],[],[],options);
57  end
```

7.4 plots.m

```
%%This file runs the optimizer and calc flux functions and outputs graphs
%%of the resulting outputs.

%% get single optimiztion

%define Nx, Ny for the optimizer mesh
Nx=500;
Ny=Nx;
%define initial conditions
a0=[.012;ones(15,1)*.001];
%run optimization code
a=run_opt(Nx,Ny,a0);

%% plot function (Note: Nx and vector a of the coefficients used to define
% the height function must de defined prior to running this code. This can
% be done by runing the get single optimization section)
%define vector of positions
x = [0:L/Nx:L].';
%plot the height function as a function of x
plot(x,calcHeight(a,.05,x),'red')
hold on
%plot(x,calcHeight(a0,L,x), 'blue')
%add y axis label
ylabel("Height (m)")
%add x axis label
xlabel("Position (m)")
%add plot title
title("Sectional View of the Extruded Heat Exchanger")
hold off


%[flux,T,dTdx,xy] = CalcFlux(.05, calcHeight(a,.05,x), Nx, Ny, 20, 20, 90)
    ;
%flux

%% plot Convergence (Note: run get single optimization code and put the
% correct terminal output into feas)
%define array of itteration numbers
itter=0:1:13;
%define matrix of feasablity (first col) and first order optimality (
   second
%col) using printed output from fmincon
feas=[ 9.223e-03     ,     2.036e-02   ;    3.469e-18     ,     1.710e-02   ;
     1.041e-17     ,     5.476e-03   ;    5.204e-18     ,     1.431e-03   ;
      1.214e-17     ,     3.199e-04   ;    8.674e-18     ,    3.389e-05   ;
       2.082e-17     ,    3.985e-05 ;    1.388e-17     ,    1.634e-05   ;
    3.469e-17     ,     1.800e-05   ;   4.857e-17     ,     4.567e-05   ;
       3.469e-17     ,    1.536e-05   ;    3.469e-17     ,     3.119e-05   ;
```

```
      2.776e-17    ,    1.394e-06   ;    2.776e-17      ,     2.575e-19    ];
%plot log(feasibility) vs itteration number
semilogy(itter,feas(:,1),'blue')
hold on
%plot log(first order optimality) vs itteration number
semilogy(itter,feas(:,2),'red')
%add chart title
title("Feasibility and Convergance History")
%label x axis
xlabel("Number of Itterations")
%label y axis
ylabel("log(Feasibility, First Order Optimality)")
%add legend
legend(["Feasibility","First-Order Optimality"])
hold off

%% run convergence study (Note: vector a of the coefficients used to
   define
% the height function must de defined prior to running this code)
%setup initial and final nxs
nstart=10;
nfinal=1000;
%create vector of nxs to try
nx=[nstart:nstart:100,200:100:nfinal];
%initialize vector to store fluxes
fluxx=[];
for i=1:max(size(nx))
    %calculate x for the current nx
    x = [0:L/nx(i):L].';
    %calculate the heat flux for the current nx
    [flux,T,dTdx,xy] = CalcFlux(.05, calcHeight(a,.05,x), nx(i), nx(i),
       20, 20, 90);
    %append the current heat flux to the end of the vector storing the
       heat
    %fluxes
    fluxx=[fluxx,flux];
end


%% plot convergence study (Note: run convergence study code before running
% this part)
%plot flux vs nx
plot(nx,fluxx)
hold on
%plot lower bound of delta
plot(nx,ones(size(fluxx))*(fluxx(max(size(fluxx)))*.95),"--b")
%plot upperbound of delta
plot(nx,ones(size(fluxx))*(fluxx(max(size(fluxx)))*1.05), "--b")
%add title
title("Convergence Study for the Optimized Heat Exchanger Design")
```

```
%add x axis label
xlabel("Number of Segments (Nx =Ny)")
%add y axis labelflux
ylabel("Flux/Unit Length (W/m)")
%add legend
legend(["Convergence plot","flux( Nx=1000)+/- 5%"])
hold off
```

7.5 fmincon Output

```
>> plots
 Iter  Func-count            Fval   Feasibility   Step Length       Norm of   First-order
                                                                       step    optimality
    0          17    6.938845e-05     9.223e-03     1.000e+00     0.000e+00     2.036e-02
    1          34    5.094673e-05     3.469e-18     1.000e+00     5.278e-03     1.710e-02
    2          51    2.866578e-05     1.041e-17     1.000e+00     1.199e-02     5.476e-03
    3          68    2.623950e-05     5.204e-18     1.000e+00     3.589e-03     1.431e-03
    4          85    2.287892e-05     1.214e-17     1.000e+00     7.598e-03     3.199e-04
    5         102    2.287854e-05     8.674e-18     1.000e+00     1.292e-05     3.389e-05
    6         119    2.287639e-05     2.082e-17     1.000e+00     6.690e-05     3.985e-05
    7         136    2.287311e-05     1.388e-17     1.000e+00     8.598e-05     1.634e-05
    8         153    2.287182e-05     3.469e-17     1.000e+00     1.341e-04     1.800e-05
    9         170    2.285690e-05     4.857e-17     1.000e+00     4.595e-04     4.567e-05
   10         187    2.285359e-05     3.469e-17     1.000e+00     1.320e-04     1.536e-05
   11         204    2.285158e-05     3.469e-17     1.000e+00     1.797e-04     3.119e-05
   12         221    2.285139e-05     2.776e-17     1.000e+00     2.191e-05     1.394e-06
   13         223    2.285139e-05     2.776e-17     7.000e-01     3.348e-17     2.575e-19

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

Figure 5: The Outpt of fmincon