Numerical Design Optimization

Report 1: Heat Exchanger Geometry Optimization

Unique ID: 1059

Professor Hicken

1. Executive summary

Optimization of heat flux between heat and water allows less overall material achieve the same overall heat flux. This saves money and allows for good form factors. For this application, some constraints have to be put in place to ensure the final design is feasible. These constraints are

- Water side has to be flat
- There is no variation of shape in the z direction, uniform cross section
- h_{min} = 1cm and h_{max} = 5cm
- The shape is to repeat every 5cm (L = 5cm)

1.1 Overview of methods

Methods for creating, analysing, and optimizing geometry will be needed.

The creation of geometries is done by creating a sine series. This allows for fairly complex shapes, however for a relatively low number of design variables, nvar << 100, creating sharp curves is not very feasible, especially within constraints.

To analyse the geometry, a Finite-Volume Discretization and the 2D steady state heat equation is utilized.

For optimization of the geometry, an objective function needs to be defined. In this case, it is a wrapper function that calculates the height given a matrix of design variables and calculates the flux and then returns the inverse of the flux. Then the upper and lower bounds of the function first have to be found, this is done via a linear inequality constraint. The optimization then uses fmincon to find the minimum optimal combination of design parameters, hence why the objective function needed to return the inverse.

1.3 Overview of results

The optimal shape is one that finds as much surface area as possible to create favorable conditions for convection from the air.

2. Analysis method

The analysis method has 2 major parts, how the heat flux is calculated and how geometries are represented.

2.1 Heat flux

To calculate heat flux, the assumption of steady state conditions and a constant thermal conductivity in the heat exchanger were made. Heat flux is calculated using a Finite-Volume Discretization and the 2D steady state heat equation. This can be done by taking a matrix of evenly spaced points and creating quadrilaterals between them. This system of quadrilaterals can be considered a mesh. From here, a volumetric integral is equated to 4 surface integrals around the quadrilateral via the divergence theorem. The derivatives of temperature are then approximated to the difference in temperature at the center of that element and the adjacent one. When all the surface integrals are compiled, a linear system of equations can be used to solve for all the elements. To calculate heat flux, this equation is used over the domain:

$$k \int_{x=0}^{L} \frac{\partial T}{\partial y} dx = Q$$

Equation 2.1.1: heat flux over the domain 0 to L

Values for dx and dy are found using the number of subdivisions, Nx and Ny respectively, over the domain.

Values used in analysis are:

$T_{water} = 363K$

$T_{air} = 293K$

$K_{heat\ exhanchanger} = 20\ W / (mK)$

2.2 Geometry parameterization

Geometry parameterization is given by this equation:

$$h(x) = a_1 + \sum_{k=2}^{n} a_k sin(\frac{2\pi(k-1)x}{L})$$

Equation 2.2.1: Geometry parameterization using a sine series.

Where n is the number of design variables and a is an array of design variables. The optimization algorithm will try to optimize a to create the most heat flux. The more design variables that are defined the more complex the geometry can be and possibly output a greater result, however

computing cost goes up due to having more iterations needed within the height function, the constraint function, and within the optimization itself.

A limitation of this parametrization is sharp changes in shape, like rectangular fins, will not be represented well in this model, especially with the number of design variables less than 5. For simplification, the geometry is consistent along the

3 . Optimization

The optimization problem is to find a geometry that maximizes heat flux between hot water on one side of the heat exchanger and cold air on the other.

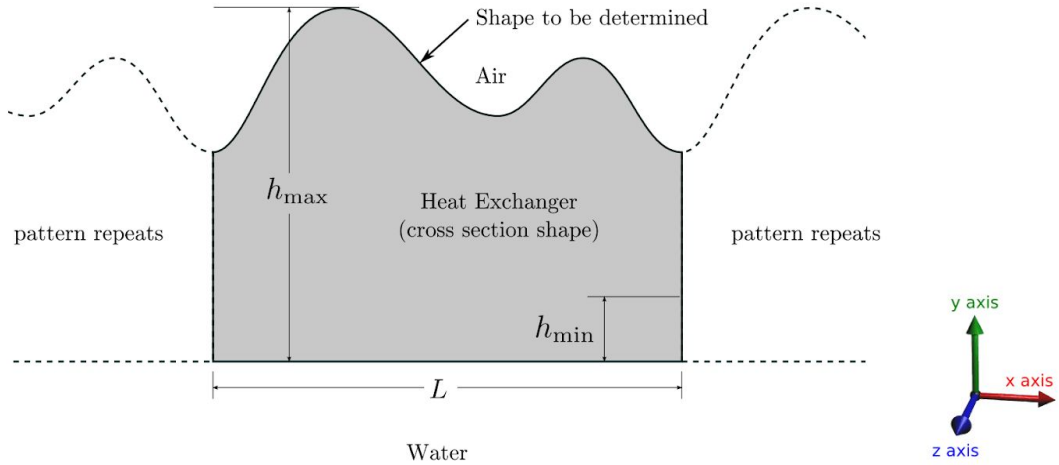

Figure 3.1: A sample showing the problem to be optimized

The optimization method chosen for this problem is in MatLab's optimization toolbox and is a function called "fmincon". Fmincon finds the minimum of a constrained function with multiple variables.

In order to do any optimization, an objective function needs to be created. In this case, the objective function took the form of a wrapper function that, given all parameters, calculated the height which it then used to find the flux. Since fmincon finds the minimum of a function and we are looking to maximize the flux, one could either negate the flux or take the reciprocal of it. After trial and error, taking the reciprocal was found to give higher flux values, all else being equal.

Some constraints for ease of manufacturability have to be put in place:

- Water side has to be flat
- There is no variation of shape in the z direction, uniform cross section

- h_{min} = 1cm and h_{max} = 5cm
- The shape is to repeat every 5cm (L = 5cm)

To enforce the height constraints, linear inequality constraints are used. Fmincon expects the constraints to be in the form Ax < b, where A, x, and b are matrices (Note: the "x" referred to here is not the matrix containing evenly spaced x values, but rather the matrix containing the design variables). Both the A and b matrices have twice as many rows as the number of subdivisions plus one, half of these values will ensure the optimized result does not go over 5cm and the other half will ensure it does not go below 1cm. The A matrix however, has as many columns as there are design variables. When calculating the upper bounds, iterate from 2 to the number of design variables, called k, and over each x value and evaluate $sin(\frac{2\pi(k-1)x_i}{L})$. These results are stored in the A matrix, in element (i,k). The b matrix is simply populated by filling each element with the upper bound. To find the lower bound populate the next batch of rows, starting at Nx + 1, in a similar fashion as before, but negate $sin(\frac{2\pi(k-1)x_i}{L})$ and negate the lower bound to populate the b matrix with. This is because fmincon expects the constraint in the form of Ax < b. Negating both sides ensures that the geometry stays above the lower bound.

To find reasonable Nx and Ny values, a mesh convergence study is done. The general process to find mesh convergence is to find the flux value at different Nx and Ny values and plot the Nx and Ny value against the flux values. Then based on computing needs find a value that is acceptably accurate and not too costly to run. The full code can be found in ***Appendix ____***

4. Results

The result of the mesh convergence study is shown in *Figure 4.1,* this shows a logarithmic growth between Nx and Ny values. To ensure the mesh is not too corse, the values chosen are: Nx = Ny = 100.

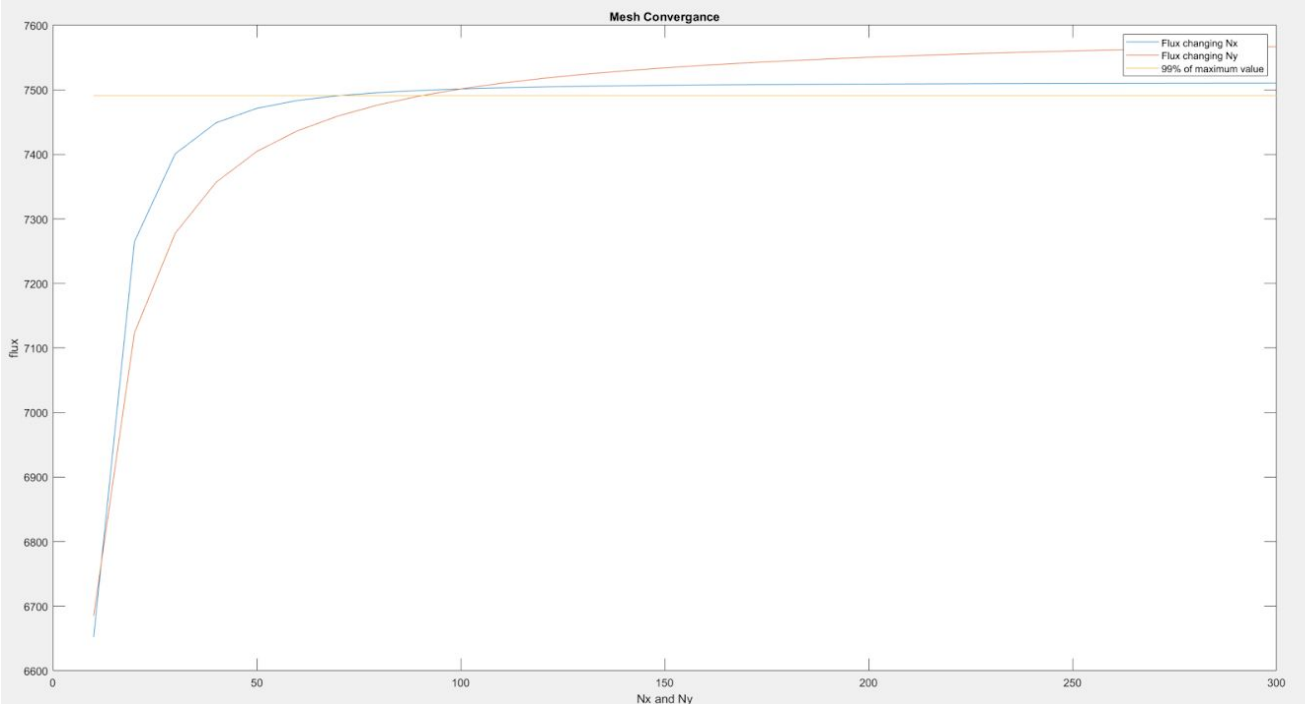

Figure 4.1: Mesh convergence

Next using the values found for Nx and Ny, the objective function is passed into fmincon. *Figure 4.2* shows the first order optimality. This is mostly to ensure that a reasonable minimizer has been found. This result is acceptable because the first order optimality is close to 0.

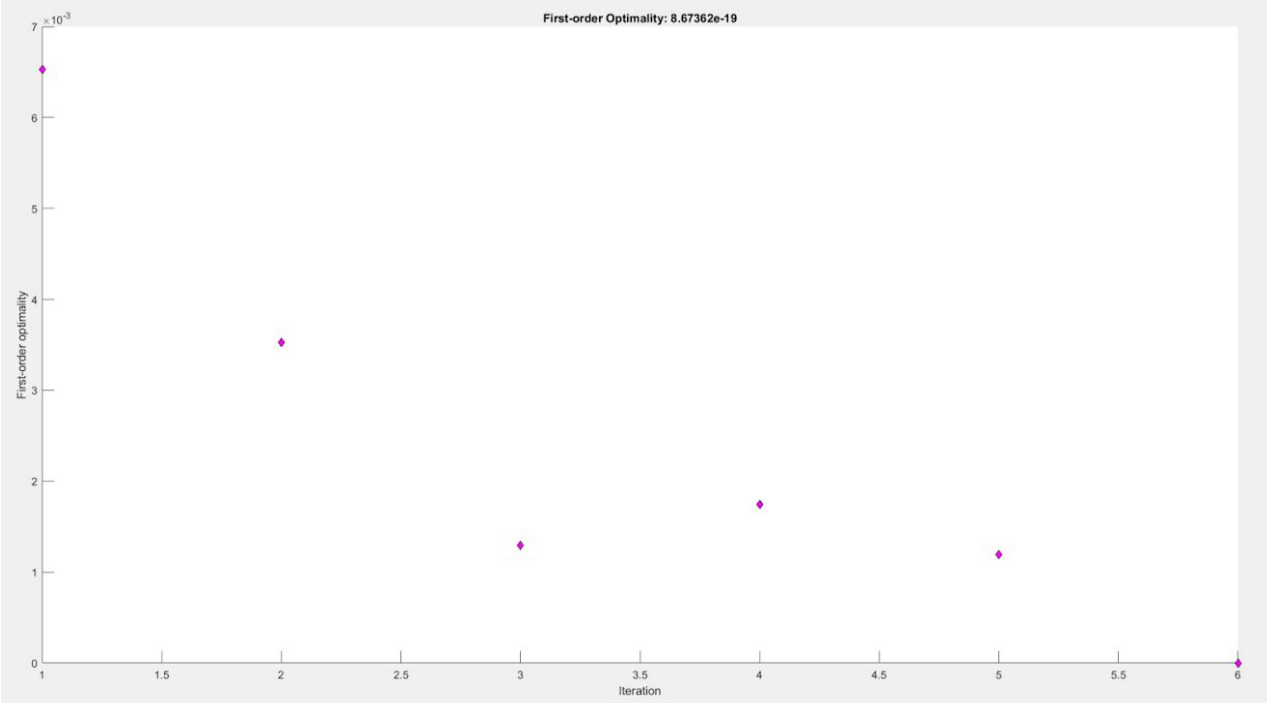

Figure 4.2: First order optimality

Finally the results, the optimization is run many times with many different initial conditions, a0. The following is a graph of just one of those runs. With any reasonable initial condition, reasonable being one that isn't too far out of the bounds of the problem, the same optimized geometry emerges.

In the case when a0 = [0.03; 0.0015; 0.0025; 0.003; 0.009], the initial flux is 6,158.3 W/ m, the final flux is 13,260, giving an improvement of 115.3225%. The optimized matrix of coefficients is [.03, 0,0,0 .02]. To find a good value for the number of design variables the number of elements in a0 was changed and the optimization was run multiple times. Each time a similar value for final flux was output. 5 design variables is good middle ground between an accurate result and computational cost. The initial and final 2D cross section is shown in *Figure 4.3.*

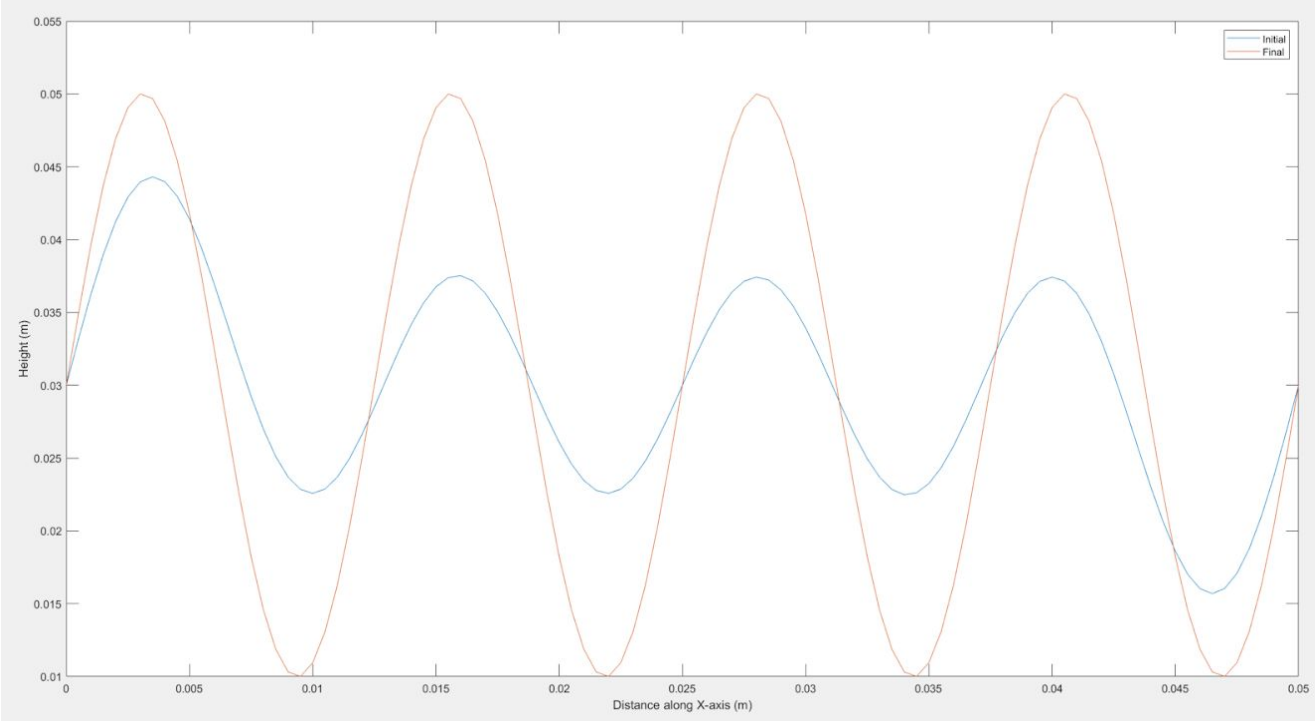

Figure 4.3: Initial and final (optimized) 2D cross section

5. Conclusions and Discussion of the Results

The final (optimal) shape as determined by fmincon makes sense. It is trying to create as much surface area as possible to create favorable conditions for convection from the air. The repetitive nature of the optimized shape will be easier to manufacture than a more complex shape which is good down the line. The optimization result could be made better by experimenting with different parameterizations of the geometry. This is not guaranteed to improve the result, however, there is no guarantee that the chosen parameterization is the optimal one.

6. Appendix

6.1 Source code, this code was copy and pasted in so it is easy to run

6.1.1 Height function

```
function [h] = height(a, L, x)
    % height calculates the y value at each x value using a sine series
    % parametrization
    % a is the matrix of coefficients
    % L is the length of the section
    % x is the matrix of evenly spaced points along the x-axis
    % h (output) is a matrix of the y value at each x value
    nvar = numel(a);
    h = zeros(size(x,1),1); %initialize a matrix of zeros that is the same size as x
    h = h+ a(1); % in the parametrization used, the first values, a1, is added to all elements
    for k = 2:nvar
        % loops over the number of design variables
        for i = 1:size(x,1)
            %loops over the number of x elements
            h(i) = h(i) + a(k)* sin(2*pi*(k-1)*x(i)/L);
            % the height of each element is iterativly added to get the
            % sine series
        end
    end
```

6.1.2 Constraint enforcement

```
function [Aineq, bineq] = limits(a, Nx, L, x)
    %creates linear constraints for the upper and lower bounds
    % a is the matrix of coefficients
    % Nx is the amount of subdivisions in the X Direction
    % L is the length of the section
    % x is the matrix of evenly spaced points along the x-axis
    % Ax < b
```

```
% Aineq is A in the inequality
% bineq is b in the inequality

nvar = numel(a); % 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
  % next, the lower bound; we use ptr to shift the index in Aineq and bineq
  ptr = Nx+1;
   Aineq(ptr+i,1) = -1.0; % note the negative here!!! fmincon expects inequality in a form A*x
< b
  for k = 2:nvar
    Aineq(ptr+i,k) = -sin(2*pi*(k-1)*x(i)/L); % again, a negative
  end
  bineq(ptr+i) = -0.01; % negative lower bound
end
```

6.1.3 Mesh Convergence

```
function out = test ()
        % takes no inputs, all inputs are is hard coded in
        % outputs the flux
        % graphs the effect of increasing values for Nx and Ny
        close all
        a0 = [0.0300, -0.0016, -0.0199, 0.0017];
        L = .05;
```

```
kappa = 20;
Ttop = 20 + 273;
Tbot= 90 + 273; % inputs to CalcFlux
start = 10; %starting Nx and Ny value
ending = 300; %ending Nx and Ny values
iter = 10;
n = ((ending - start)/ iter) +1; % find the total amount of
%Nx and Ny values to be testes
x_flux = zeros(n,1);
y_flux = zeros(n,1);
Nx = 100;
Ny = 100;
counter  = 0;
x_axis = zeros(n,1);
for i = start:iter: ending
   % calculate flux for Nx and Ny values
   counter = counter + 1;
   x_axis(counter) = i;
   x_x = [0:L/i:L].';
   x_y = [0:L/Nx:L].';
   h_y = height(a0, L, x_y);
   h_x = height(a0, L, x_x);
   fx = CalcFlux(L, h_x, i, Ny, kappa, Ttop, Tbot);
   fy = CalcFlux(L, h_y, Nx, i, kappa, Ttop, Tbot);
   x_flux(counter) = fx(1);
   y_flux(counter) = fy(1);
end
flux_max = zeros(n,1);
flux_max = flux_max + max(x_flux(n), y_flux(n));
plot(x_axis, x_flux,x_axis, y_flux, x_axis, flux_max * .99),
xlabel('Nx and Ny'), ylabel('flux'),
```

```
legend('Flux changing Nx','Flux changing Ny', '99% of maximum value'), title('Mesh Convergance')
out = flux_max;
```

6.1.4 Objective function

```
function [f] = objection_func(a, L, Nx, Ny, kappa, Ttop, Tbot, x)
% this wrapper function calculates the flux using all the givens in the
% problem statement as well as the matrix of design variables
  h = height(a, L, x); %need h to calculate the flux
  [flux,~,~,~] = CalcFlux(L, h, Nx, Ny, kappa, Ttop, Tbot);%suppress all
  %other outputs other than flux
  f = 1/flux; %b/c fmincon is going to find the min we need to give it
  % the reciprocal
```

6.1.5 Main function

```
function [a0, afinal, flux0, flux1, improval] = main()
  % takes no inputs all values are hard coded in
  % outputs:
  % a0: inital guess for design variables
  % afinal: final matrix of design variables
  % flux0: initial flux
  % flux1: final flux
  % improval: percent change in flux
  close all
  L = .05;
  Nx = 100;
  Ny = 100;
  x = [0:L/Nx:L].';
  kappa = 20;
  Ttop = 20 + 273;
  Tbot= 90 + 273; % hard coded knowns
```

```
fun = @(a) objection_func(a, L, Nx, Ny, kappa, Ttop, Tbot, x);
%line above: an anonymous function that eliminates all other inputs
%other than the one we want to optimize, the matrix of design variables
a0 = [.03; 0.01.*rand(4,1)]; % initial guess, randomized
[Aineq, Bineq] = limits(a0, Nx, L, x); %find constraint matrices
options = optimset('Display', 'iter', 'algorithm', 'sqp',"PlotFcn", 'optimplotfirstorderopt');
afinal = fmincon(fun,a0,Aineq,Bineq, [],[],[],[],[], options);
%line above: run fmincon from matlab optimization toolbox
h = height(a0, L, x);% calculate the initial and final conditions
%and determine the improvement
h1 = height(afinal, L, x);
flux0 = CalcFlux(L, h, Nx, Ny, kappa, Ttop, Tbot);
flux1 = CalcFlux(L, h1, Nx, Ny, kappa, Ttop, Tbot);
 plot(x,h, x,h1), legend("Initial", "Final"), ylabel('Height (m)') , xlabel('Distance along X-axis
(m)'), title('2D Cross section');
improval = (flux1 - flux0) / flux0 * 100;
```