Topological Optimization Using the SIMP Method

©2021

Mikal William Nelson

B.S. Mathematics, University of Minnesota, 2013

Submitted to the graduate degree program in Department of Mathematics and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Arts.

Paul Cazeaux, Chairperson

Committee members

Mat Johnson

Dionyssios Mantzavinos

Yannan Shen

Date defended: July 26, 2021

The Thesis Committee for Mikal William Nelson certifies
that this is the approved version of the following thesis :

Topological Optimization Using the SIMP Method

Paul Cazeaux, Chairperson

Date approved: July, 2021

Abstract

The Volume-to-Point (VP) Heat Conduction Problem is concerned with the optimal distribution of conductive material over a heat generating domain. This thesis begins with an overview of the heat equation and its discretization using the Finite Volume Method. We proceed to give a summary of the topic of mathematical optimization and optimization techniques. After completing this brief presentation of background information we proceed to present the application of the Solid Isotropic Material with Penalization (SIMP) method to the VP heat conduction problem to create designs with optimal conductive material placement which minimize average temperature.

Acknowledgements

Dedicated to my mother, my first math teacher.

Thank you to my wife, father, brothers, and all the teachers and professors who have helped me grow along the way.

Contents

List of Figures

Chapter 1

Background

1.1 PDE Discretization

Multidimensional topological optimization problems often involve the use of partial differential equations (PDEs) which model the physical properties of the materials involved. Most of these PDEs cannot be uniquely solved analytically, so we turn to numerical methods in order to approximate their solutions. The first step in many of these methods is to discretize our domain; that is, we want to choose some scheme to divide our continuous domain into a finite number of pieces over which we will apply a particular method to approximate solutions to the PDE.

The implementation of the SIMP method introduced in §2.1 uses the Finite Volume Method to discretize and approximate solutions to the heat equation for the heat generating medium. We will introduce the Heat Equation and then proceed to give an overview of the Finite Volume Method.

1.1.1 The Heat Equation

Consider heat flow through a stationary, inhomogeneous object. The temperature at any point in the interior of the object will depend on the spatial position chosen as well as the time we measure the temperature at that point. Therefore, the temperature (T) at any point in such an object is a function of both space ($\mathbf{x}$) and time (t) coordinates: $T(\mathbf{x}, t)$. Physical

principles require that such a temperature function must satisfy the equation

$$\frac{\partial T}{\partial t} = \nabla \cdot (k(\mathbf{x})\nabla T)\,, \tag{1.1}$$

where ∇ is the gradient operator and the function k represents the thermal diffusivity at a point in our object.

Equation (1.1) is commonly referred to as the Heat or Diffusion Equation. If we were to have a constant thermal diffusivity throughout our object on a simple domain (such as a square or circle), it would be possible to analytically find a solution to this partial differential equation. However, as in the VP heat conduction problem, when k is not constant we must turn to numerical methods to find approximate solutions for the function T.

1.1.2 The Finite Volume Method

For the numerical approximations of PDEs in this paper the Finite Volume Method (FVM) was implemented, which will be described in this section.

As with many other numerical method to solve PDEs, we must first discretize our domain by creating a mesh. One major advantage of the finite volume method is that it allows for a great amount of freedom in mesh choice. When using FVM the domain can be discretized into a mesh of arbitrary polygons, but uniform squares or rectangles were chosen in our work to simplify the resulting calculations.

Given a mesh of polygons on a domain Ω with sample points at $\{x_i\} \subset \Omega$, we create a set of *control volumes* around each x_i. The resulting set of control volumes will be used to discretize the partial differential equation. The finite volume method has us integrate our PDE over each control volume and then use the Divergence Theorem (Theorem 1) to convert volume integrals into surface integrals involving the fluxes across the boundaries of the control volumes. We then approximate those fluxes across the boundaries to calculate approximate solutions to the PDE of interest, such as (1.1).

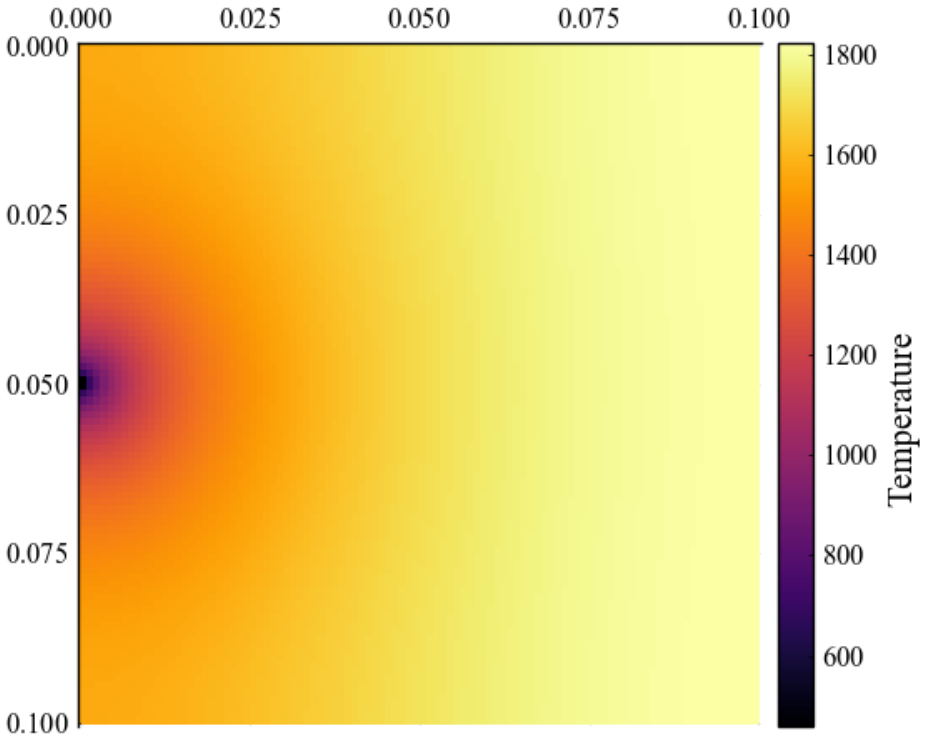


Figure 1.1: Heatmap for a 0.1 m $\times$ 0.1 m object with uniform heat generation and a heat-sink at the center of its west boundary. This map was produced via the Finite Volume Method using 100×100 uniform control volumes.

Theorem 1 (The Divergence Theorem)**.** *Suppose that $\mathcal{V}$ is a compact subset of $\mathbb{R}^n$ that has a piecewise smooth boundary $\mathcal{S}$ (i.e. $\partial\mathcal{V} = \mathcal{S}$) with outward pointing normal vectors. If $\mathbf{F}$ is a continously differentiable vector field defined on a neighborhood of $\mathcal{V}$, then*

$$\iiint_{\mathcal{V}} (\nabla \cdot \mathbf{F}) \, \mathrm{d}\mathcal{V} = \oiint_{\mathcal{S}} (\mathbf{F} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\mathcal{S} \tag{1.2}$$

where $\hat{\mathbf{n}}$ is the outwards pointing normal vector to the boundary.

The divergence theorem is the key component in the finite volume method because it allows us to look at fluxes across the boundaries of each control volume, rather than the control volume itself.

Let us look at the finite volume method applied to the heat equation in two dimensions. Suppose we have discretized our space by dividing it up into a mesh of control volumes $\{V_i\}$. We integrate (1.1) over each control volume, using the divergence theorem to convert the volume integral into a surface integral:

$$\int_{V_i} \frac{\partial T}{\partial t} \, \mathrm{d}\mathbf{x} = \int_{V_i} \nabla \cdot (k(\mathbf{x}) \nabla T) \, \mathrm{d}\mathbf{x} \underset{(1.2)}{=} \int_{\partial V_i} k(\mathbf{x}) \nabla T \cdot \hat{\mathbf{n}} \, \mathrm{d}s, \tag{1.3}$$

where s represents the curves that form the boundary of the control volume. Then, applying an approximation scheme to this result, we obtain a sparse and structured linear system. For example, one could apply what is called a "two-point flux approximation" scheme which uses finite differences of function values from neighboring cells to the control volume to approximate the flux through the control volume faces [7].

In a square grid there are only four neighboring cells which we can label as North, South, East, West. For a control volume V_i we'll label the North boundary as ∂V_N, the South boundary as ∂V_S, the East boundary as ∂V_E, and the West boundary as ∂V_W. Additionally, let Δx be the length of the North and South boundaries, and Δy the length of the East and

West boundaries. We can discretize (1.3) as

$$\begin{aligned}
\int_{\partial V_i} k(\mathbf{x})\nabla T\cdot\hat{\mathbf{n}}\,\mathrm{d}s &= \int_{\partial V_N} k(\mathbf{x})\nabla T\cdot\hat{\mathbf{n}}_N\,\mathrm{d}s+\int_{\partial V_S} k(\mathbf{x})\nabla T\cdot\hat{\mathbf{n}}_S\,\mathrm{d}s \\
&\quad+\int_{\partial V_E} k(\mathbf{x})\nabla T\cdot\hat{\mathbf{n}}_E\,\mathrm{d}s+\int_{\partial V_W} k(\mathbf{x})\nabla T\cdot\hat{\mathbf{n}}_W\,\mathrm{d}s \\
&\approx k_N\frac{T_N-T_i}{\|\mathbf{x}_N-\mathbf{x}_i\|}\Delta x+k_S\frac{T_S-T_i}{\|\mathbf{x}_S-\mathbf{x}_i\|}\Delta x \\
&\quad+k_E\frac{T_E-T_i}{\|\mathbf{x}_E-\mathbf{x}_i\|}\Delta y+k_W\frac{T_W-T_i}{\|\mathbf{x}_W-\mathbf{x}_i\|}\Delta y \qquad (1.4)\\
\implies \Delta x\Delta y\frac{\mathrm{d}T_i}{\mathrm{d}t} &= \left(k_N\frac{T_N-T_i}{\|\mathbf{x}_N-\mathbf{x}_i\|}+k_S\frac{T_S-T_i}{\|\mathbf{x}_S-\mathbf{x}_i\|}\right)\Delta x \\
&\quad+\left(k_E\frac{T_E-T_i}{\|\mathbf{x}_E-\mathbf{x}_i\|}+k_W\frac{T_W-T_i}{\|\mathbf{x}_W-\mathbf{x}_i\|}\right)\Delta y
\end{aligned}$$

The process in (1.4) is repeated for all control volumes i to produce a system of ordinary differential equations which is used to solve for the values of T_i.

One other major advantage of the finite volume method is that boundary conditions can easily be taken into account on general domains. For example, adding a heat-sink by applying a Dirichlet boundary condition can be thought of as zeroing out our algebraic equations by introducing a ghost cell that, when interpolated with the boundary cell, causes the temperature across the boundary to be zero.

1.2 The Optimization Problem

We now move on to an overview of mathematical optimization and a survey of optimization methods.

1.2.1 Optimization Problem Definition

Definition 1. *An optimization problem (in standard form) has the form*

$$\begin{array}{ll} \textit{minimize} & f_0(\mathbf{x}) \\ \textit{subject to} & f_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0, \quad i = 1, \ldots, p \end{array} \tag{1.5}$$

where

- $\mathbf{x} = (x_1, \ldots, x_n)$ *are the optimization variables,*
- $f_0 : \mathbb{R}^n \to \mathbb{R}$ *is the objective function,*
- $f_i : \mathbb{R}^n \to \mathbb{R}$ *are the inequality constraint functions, and*
- $h_i : \mathbb{R}^n \to \mathbb{R}$ *are the equality constraint functions.*

If there are no constraints ($m = p = 0$), then the problem is called *unconstrained*. [1, p. 127]

We call a vector $\mathbf{x}^\star$ *globally optimal* if it has the smallest objective value among all vectors that satisfy the constraints. That is, for any $\mathbf{z}$ with $f_1(\mathbf{z}) \leq 0, \ldots, f_m(\mathbf{z}) \leq 0$, then $f_0(\mathbf{z}) \geq f_0(\mathbf{x}^\star)$. A vector $\mathbf{x}$ that is in the domains of each function f_i and h_i is called *feasible* if it satisfies all the constraints. Finally, the *optimal value* $p^\star$ of the problem is defined as

$$p^\star = \{f_0(\mathbf{x}) \mid f_i(\mathbf{x}) \leq 0, i = 1, \ldots, m, h_i(\mathbf{x}) = 0, i = 1, \ldots, p\}\,.$$

Therefore, $p^\star = f_0(\mathbf{x}^\star)$, the objective function value at a feasible and globally optimal vector $\mathbf{x}^\star$.

A vector is *locally optimal* if it has the smallest objective value among all feasible candidates within a neighboring set. The global optimum is also a local optimum, but a local optimum need not be the global optimum.

Notice that the optimization problem in standard form is a minimization problem. We can easily change it into a maximization problem by minimizing the objective function $-f_0$ subject to the same constraints.

The optimization problem is *linear* or called a *linear program* if the objective and constraint functions are all linear. An optimization problem involving a quadratic objective function and linear constraints is *quadratic* or a *quadratic program*. If the optimization problem is not linear or quadratic, it is referred to as a *nonlinear program*.

There are exists efficient methods for solving linear programming and many quadratic programming problems.

1.2.2 Convex Optimization

A set C is *convex* if the line segment between any two points in C lies within C. That is, C is convex if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have $\theta x_1 + (1-\theta)x_2 \in C$.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if the domain of f is a convex set and if for all x, y in the domain of f, and θ with $0 \leq \theta \leq 1$, we have

$$f\left(\theta x + (1-\theta)\, y\right) \leq \theta f(x) + (1-\theta) f(y). \tag{1.6}$$

A *convex optimization problem*, therefore, is an optimization problem of the form

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i = 1, \ldots, m \\ & a_i^T = b_i, & i = 1, \ldots, p \end{array} \tag{1.7}$$

where $f_0, \ldots, f_m$ are convex functions.

Notice that there are three requirements that differentiate a convex optimization problem from a general optimization problem:

- The objective function must be convex.

- The inequality constraint functions must be convex.
- The equality constraint functions must be affine.

In a convex optimization problem we minimize a convex objective function over a convex set and any locally optimal point is also globally optimal. This is a *very* useful fact!

Why might local optimality implying global optimality be useful? Consider a situation where we have a convex objective function. If we are able to find any minimum value for the objective function, then we know this value is not just a local minimum, but indeed a global minimum. If we do not have a convex function, we cannot be as assured that we have found the global optimal value. For example, consider a function such as the one shown in Figure 1.2. An optimization strategy may find the local minimum near $x = 1$, but since the function is not convex everywhere on its domain, we cannot conclude that this value is the globally optimal value. In fact, we see that the global minimum, and hence the actual optimal value, is between $x = -1$ and $x = -0.5$. On the other hand, if we have a function that is everywhere convex, as soon as we find a local minima, we can be assured that it is also the global minima!

So we can see that convexity is a very powerful and useful property in terms of optimization problems. As a result, any time we can take advantage of convexity or approximate functions using convex functions, we often do so.

1.3 Optimization Methods

In this section I will present a few optimization algorithms for unconstrained optimization.

These algorithms follow a general blueprint:

1. Choose an initial "guess" for the optimal value x.
2. Find a descent direction Δx.

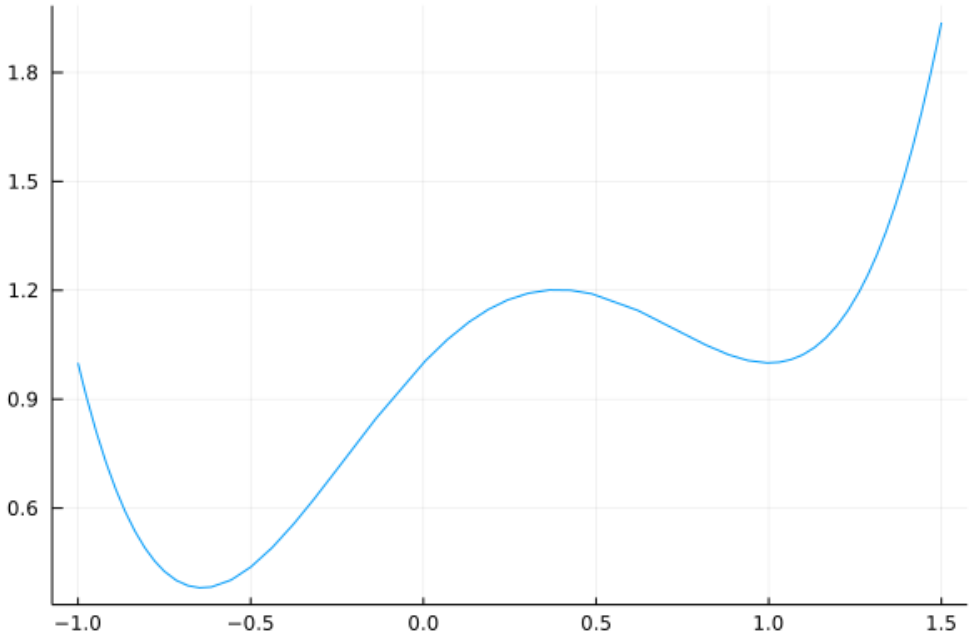


Figure 1.2: A graph of the non-convex function $f(x) = x^4 - x^3 - x^2 + x + 1$. Notice it has two local minima and that the right local minima is not equal to the global minimum.

3. Use a line search method to determine an appropriate step-size (t) in the descent direction.

4. Compute new approximation value $x + t\Delta x$.

Each method uses a line search, but how the descent direction is chosen is the main differentiating factor in each method.

1.3.1 Line Search Methods

The *line search* is a strategy that selects the step size (commonly represented by t) that determines where along the line $\{x + t\Delta x \mid t \in \mathbb{R}_+\}$ the next iterate in the descent method will be. Line search strategies can either be *exact* or *inexact*.

Exact Line Search

An *exact line search* chooses the value t along the ray $\{x + t\Delta x \mid t \in \mathbb{R}_+\}$ that exactly minimizes the function of interest f:

$$t = \arg\min_{s \geq 0} f(x + s\Delta x)$$

An exact line search is almost never practical. In very special cases, such as some quadratic optimization problems where an explicit formula is available, one might employ an exact line search.

Backtracking Line Search

Most often in practice *inexact* line searches are used. In an inexact line search, we choose t such that f is *approximately* minimized or reduced "enough" along $\{x + t\Delta x \mid t \in \mathbb{R}_+\}$.

One inexact line search strategy is the *Backtracking Line Search*.

Algorithm 1 Backtracking Line Search [1]

given a descent direction Δx for f at $x \in \mathbf{dom} f, \alpha \in (0, 0.5), \beta \in (0, 1)$.

$t := 1$.

while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$ **do**

  $t := \beta t$

end while

"Backtracking" in the name is refers to the fact that the method starts with a unit step size ($t = 1$) and then reduces the step size ("backtracks") by the factor β until we meet the stopping criterion $f(x + t\Delta x) \leq f(x) + \alpha t \nabla f(x)^T \Delta x$.

Notice that the backtracking search will find a step size t such that $f(x + t\Delta x)$ is smaller relative to $f(x)$. The step size chosen may not exactly be the minimum of the function, but we have funneled it down closer to the minimum of f.

1.3.2 Gradient Descent

The gradient descent method chooses the search direction to be the negative gradient. That is, in this method we set $\Delta x = -\nabla f(x)$, where f is the function we seek to optimize. Since the gradient of a function gives the direction of greatest increase, naturally the negative gradient will give the direction of the most rapid decline.

Algorithm 2 Gradient Descent Method [1]

given a starting point $x \in \mathbf{dom} f$.
repeat

1. $\Delta x := -\nabla f(x)$.
2. *Line search.* Choose step size t via exact or backtracking line search.
3. *Update.* $x := x + t\Delta x$.

until stopping criterion is satisfied.

Notice that Algorithm 2 essentially employs a line search to determine a step size and then updates the iterate in the direction of the steepest descent. This is repeated until some sort of stopping criterion is met, typically something of the form $\|\nabla f(x)\|_2 \leq \eta$, where η is small and positive. Another common stopping criterion is to stop the algorithm when no significant progress is made between iterates by stopping the algorithm once $\|f_{k+1} - f_k\| < \eta$.

An implementation of the gradient descent algorithm in the Julia language can be found in **??**.

1.3.3 Nonlinear Conjugate Gradient

The Nonlinear Conjugate Gradient method works similarly to the gradient descent algorithm, but adds the additional requirement that in each iteration the descent direction is conjugate to those of each previous iteration.

Suppose we have a function $f(x)$ of N variables. Let x_0 be an initial guess value for the minimum. The opposite of the gradient will give the direction of steepest descent. Therefore, start off by setting $\Delta x_0 = -\nabla f(x_0)$.

Set an adjustable step length α and perform a line search in the direction $d_0 = \Delta x_0$ until a minimum of f is reached:

$$\alpha_0 := \arg\min_{\alpha} f(x_0 + \alpha \Delta x_0),$$
$$x_1 = x_0 + \alpha_0 \Delta x_0.$$

Suppose we were to simply iterate this process and for each step i the following was repeated:

1. Set $\Delta x_i = -\nabla f(x_i)$.
2. Calculate $\alpha_i = \arg\min_{\alpha} f(x_i + \alpha \Delta x_i)$.
3. Compute $x_{i+1} = x_i + \alpha_i \Delta x_i$.

However, there is an issue with this proposed iterative scheme: We have moved α_i in direction Δx_i to find the minimum value in that direction, but by moving α_{i+1} in direction Δx_{i+1} we may have accidentally *undone* the progress made in the previous iteration so that we no longer have a minimum value in direction Δx_i. This problem can be fixed by making sure that successive direction vectors have no influence in the directions of previous iterations. That is, we require our directions in each iteration to be conjugate (with respect to the matrix of coefficient for our system) to one another. Therefore, rather than taking Δx_{i+1} to be $-\nabla f(x_i)$, we compute a direction conjugate to all previous directions by some pre-chosen methodology. This suggests the following iterative scheme:

After the first iteration, the following steps constitute one iteration along a conjugate direction:

1. Calculate the new steepest descent direction: $\Delta x_i = -\nabla f(x_i)$,
2. Compute β_i using some formulation. Two options are below:
   - Fletcher–Reeves: $\beta_i^{FR} = \frac{\Delta x_i^T \Delta x_i}{\Delta x_{i-1}^T \Delta x_{i-1}}$

- Polak–Ribière: $\beta_i^{PR} = \frac{\Delta x_i^T(\Delta x_i - \Delta x_{i-1})}{\Delta x_{i-1}^T \Delta x_{i-1}}$

3. Update the conjugate direction: $d_i = \Delta x_i + \beta_i d_{i-1}$,

4. Line search: Optimize $\alpha_i = \arg\min_{\alpha} f(x_i + \alpha d_i)$,

5. Update iterate value: $x_{i+1} = x_i + \alpha_i d_i$.

Algorithm 3 uses the Newton-Raphson method to find the values of α_i.

Algorithm 3 Nonlinear Conjugate Gradient Using Newton-Raphson [4]

Given a function f, a starting value x, a maximum number of CG iterations $i_{\max}$, a CG error tolerance $\epsilon < 1$, a maximum number of Newton-Raphson iterations $j_{\max}$, and a Newton-Raphson error tolerance $\varepsilon < 1$:

$i \Leftarrow 0$
$k \Leftarrow 0$
$r \Leftarrow -f'(x)$
$d \Leftarrow r$
$\delta_{\text{new}} \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta_{\text{new}}$
while $i < i_{\max}$ and $\delta_{\text{new}} > \epsilon^2 \delta_0$ **do**
  $j \Leftarrow 0$
  $\delta_d \Leftarrow d^T d$
  while true **do**
    $\alpha \Leftarrow -\frac{\left[f'(x)\right]^T d}{d^T f''(x) d}$
    $x \Leftarrow x + \alpha d$
    $j \Leftarrow j + 1$
    $j < j_{\max}$ and $\alpha^2 \delta_d > \varepsilon^2$ OR **Break**
  end while
  $r \Leftarrow -f'(x)$
  $\delta_{\text{old}} \Leftarrow \delta_{\text{new}}$
  $\delta_{\text{new}} \Leftarrow r^T r$
  $\beta \Leftarrow \frac{\delta_{\text{new}}}{\delta_{\text{old}}}$
  $d \Leftarrow r + \beta d$
  $k \Leftarrow k + 1$
  if $k = n$ or $r^T d \leq 0$ **then**
    $d \Leftarrow r$
    $k \Leftarrow 0$
  end if
  $i \Leftarrow i + 1$
end while

1.4 The Method of Moving Asymptotes (MMA)

The Method of Moving Asymptotes (MMA) is a method of nonlinear programming, originally developed for structural optimization [6]. In contrast to the methods presented above, MMA is designed for constrained optimization problems. The method uses an iterative process which creates a convex subproblem that is solved in each iteration. Each of these subproblems is an approximation of the original problem with parameters that change the curvature of the approximation. These parameters act as asymptotes for the subproblem and moving the asymptotes between iterations stabilizes the convergence of the entire process.

1.4.1 General Method Description

Consider an optimization problem of the following general form

$$\begin{array}{llll} P\text{:} & \text{minimize} & f_0(\mathbf{x}) & (\mathbf{x} \in \mathbb{R}^n) \\ & \text{subject to} & f_i(\mathbf{x}) \leq \hat{f}_i, & \text{for } i = 1, \ldots, m \\ & & \underline{x}_j \leq x_j \leq \overline{x}_j, & \text{for } j = 1, \ldots, n \end{array} \tag{1.8}$$

where

- $\mathbf{x} = (x_1, \ldots, x_n)^T$ is the vector of variables
- $f_0(\mathbf{x})$ is the objective function
- $f_i(\mathbf{x}) \leq \hat{f}_i$ are behavior constraints
- $\underline{x}_j$ and $\overline{x}_j$ are given lower and upper bounds on the variables

The general approach for solving such optimization problems is to split it up and solve a sequence of subproblems using the following iteration:

Step 0: Choose a starting point $\mathbf{x}^{(0)}$, and let the iteration index $k = 0$.

Step 1: Given an iterate $\mathbf{x}^{(k)}$, calculate $f_i(\mathbf{x}^{(k)})$ and the gradients $\nabla f_i(\mathbf{x}^{(k)})$ for $i = 0, 1, \ldots, m$.

Step 2: Generate a subproblem $P^{(k)}$ by replacing, in (1.8), the functions f_i by approximating functions $f_i^{(k)}$, based on calculations from Step 1.

Step 3: Solve $P^{(k)}$ and let the optimal solution of this subproblem be the next iteration point $\mathbf{x}^{(k+1)}$. Let $k = k + 1$ and go to Step 1.

In MMA, each $f_i^{(k)}$ is obtained by a linearization of f_i in variables of the type

$$\frac{1}{x_j - L_j} \quad \text{or} \quad \frac{1}{U_j - x_j}$$

dependent on the signs of the derivatives of f_i at $\mathbf{x}^{(k)}$. The values of L_j and U_j are normally changed between iterations and are referred to as moving asymptotes.

Defining The Functions $f_i^{(k)}$

Given the iteration point $\mathbf{x}^{(k)}$ at an iteration k, values of the parameters $L_j^{(k)}$ and $U_j^{(k)}$ are chosen, for $j = 1, \ldots, n$, such that $L_j^{(k)} < x_j^{(k)} < U_j^{(k)}$.

For each $i = 0, 1, \ldots, m$, $f_i^{(k)}$ is defined by

$$f_i^{(k)}(\mathbf{x}) = r_i^{(k)} + \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{U_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - L_j^{(k)}} \right)$$

where

$$p_{ij}^{(k)} = \begin{cases} \left(U_j^{(k)} - x_j^{(k)} \right)^2 & \text{if } \frac{\partial f_i}{\partial x_j} > 0 \\ 0 & \text{if } \frac{\partial f_i}{\partial x_j} \leq 0 \end{cases}$$

$$q_{ij}^{(k)} = \begin{cases} 0 & \text{if } \frac{\partial f_i}{\partial x_j} \geq 0 \\ -\left(x_j^{(k)} - L_j^{(k)} \right)^2 \frac{\partial f_i}{\partial x_j} & \text{if } \frac{\partial f_i}{\partial x_j} < 0 \end{cases}$$

$$r_i^{(k)} = f_i(\mathbf{x}^{(k)}) - \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{U_j^{(k)} - x_j^{(k)}} + \frac{q_{ij}^{(k)}}{x_j^{(k)} - L_j^{(k)}} \right)$$

and where all $\frac{\partial f_i}{\partial x_j}$ are evaluated at $\mathbf{x} = \mathbf{x}^{(k)}$.

Notice that $f_i^{(k)}$ is a first-order approximation of f_i at $\mathbf{x}^{(k)}$. Additionally, by construction, $f_i^{(k)}$ is a convex function.

Looking at the second derivatives, the closer $L_j^{(k)}$ and $U_j^{(k)}$ are chosen to $x_j^{(k)}$, the larger the second derivatives become and hence the more curvature is given to the approximating function $f_i^{(k)}$. This means that the closer $L_j^{(k)}$ and $U_j^{(k)}$ are chosen to $x_j^{(k)}$, the more conservative the approximation of the original problem becomes. If $L^{(k)}$ and $U^{(k)}$ are chosen "'far away"' from $\mathbf{x}^{(k)}$, then the approximation $f_i^{(k)}$ becomes close to linear.

We always choose the values of $L_j^{(k)}$ and $U_j^{(k)}$ to be finite. As a result each $f_i^{(k)}$ becomes strictly convex except when $\frac{\partial f_i}{\partial x_j} = 0$ at $\mathbf{x} = x^{(k)}$.

Now, with the approximating functions $f_i^{(k)}$ as defined earlier, we have the following subproblem $P^{(k)}$:

$$
\begin{array}{lll}
P^{(k)}: & \text{minimize} & \displaystyle\sum_{j=1}^{n} \left(\frac{p_{oj}^{(k)}}{U_j^{(k)} - x_j} + \frac{q_{oj}^{(k)}}{x_j - L_j^{(k)}} \right) + r_o^{(k)} \\
& \text{subject to} & \displaystyle\sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{U_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - L_j^{(k)}} \right) + r_i^{(k)} \leq \hat{f}_i, \quad \text{for } i = 1, \ldots, m \\
& \text{and} & \max\{\underline{x}_j, \alpha_j^{(k)}\} \leq x_j \leq \min\{\overline{x}_j, \beta_j^{(k)}\}, \quad \text{for } j = 1, \ldots, n
\end{array}
\tag{1.9}
$$

(The parameters $\alpha_j^{(k)}$ and $\beta_j^{(k)}$ are called move limits.)

$\alpha_j^{(k)}$ and $\beta_j^{(k)}$ should at least be chosen such that $L_j^{(k)} < \alpha_j^{(k)} < x_j^{(k)} < \beta_j^{(k)} < U_j^{(k)}$.

General Rule for how to choose $L_j^{(k)}$ and $U_j^{(k)}$:

(a) If the process tends to oscillate, then it needs to be stabilized and this can be accomplished by moving the asymptotes closer to the current iteration point.

(b) If, instead, the process is monotone and slow, it needs to be "relaxed". This can be accomplished by moving the asymptotes away from the current iteration point.

1.4.2 The Dual Problem

$P^{(k)}$ is a convex, separable problem, so we can create a dual problem using a Lagrangian function. The Lagrangian function corresponding to $P^{(k)}$ is given by

$$\ell(x, y) = f_0^{(k)}(\mathbf{x}) + \sum_{i=1}^{m} y_i f_i^{(k)}(\mathbf{x})$$

Letting $\mathbf{y}$ be the vector of Lagrange multipliers (or "dual variables") and doing some derivations, we get the dual objective function W defined (for $\mathbf{y} \geq 0$), as below:

$$\begin{aligned} W(\mathbf{y}) &= \min_x \{\ell(\mathbf{x}, \mathbf{y}); \alpha_j \leq x_j \leq \beta_j \text{ for all } j\} \\ &= r_0 - \mathbf{y}^T \mathbf{b} + \sum_{j=1}^{n} W_j(\mathbf{y}) \end{aligned}$$

where $W_j(\mathbf{y}) = \min_{x_j} \{l_j(x_j, \mathbf{y}); \alpha_j \leq x_j \leq \beta_j\}$

This formulation is beneficial since it "eliminates" $\mathbf{x}$.

The dual problem corresponding to $P^{(k)}$ is given as follows:

$$\begin{aligned} D\text{:} \quad & \text{maximize} \quad W(\mathbf{y}) \\ & \text{subject to} \quad \mathbf{y} \geq 0 \end{aligned} \tag{1.10}$$

D is a "nice" problem which may be solved by an traditional gradient method.

Once the dual problem has been solved the optimal solution of the primal subproblem $P^{(k)}$ is directly obtained by substituting the optimal dual solution $\mathbf{y}$ into the following expression:

$$x_j(\mathbf{y}) = \frac{\left(p_{0j} + \mathbf{y}^T \mathbf{p}_j\right)^{1/2} L_j + \left(q_{0j} + \mathbf{y}^T \mathbf{q}_j\right)^{1/2} U_j}{\left(p_{0j} + \mathbf{y}^T \mathbf{p}_j\right)^{1/2} + \left(q_{0j} + \mathbf{y}^T \mathbf{q}_j\right)^{1/2}}.$$

For our implementation of the SIMP method, the MMA algorithm was employed within the NLopt optimization package in Julia.

Chapter 2

SIMP Optimization

2.1 Solid Isotropic Material with Penalization (SIMP)

Volume-to-Point (VP) Heat Conduction Problem

Consider a finite-size volume in which heat is being generated at *every* point, and which is cooled through a small patch (the heat sink) located on its boundary. Suppose that we have a finite amount of high-conductivity (k_+) material available. Our goal is to determine the optimal distribution of the k_+ material through the given volume such that *the average temperature is minimized*.

Solid Isotropic Material with Penalization (SIMP) is a method based on topology optimization that can be used to solve the VP Heat Conduction Problem. In each step of the SIMP method we increase or decrease the proportion of high-conductivity material by a small quantity. This allows us to apply methods designed for continuous optimization problems to the discrete VP problem as it transforms the binary 1—0 problem into a sequence of continuous problems [3].

2.1.1 Preliminary Parameters

Assumptions

In order to develop the method, we need to make a couple of assumptions.

First of all, the energy differential equation driving the heat-flux inside the finite-volume requires:

1. All calculations are run under steady-state conditions. That is, we seek a stable solution where quantities are independent of time.

2. All heat produced in the volume is evacuated through the heat-sink.

3. Low-conductivity materials (k_0) and high-conductivity materials (k_+) are treated as homogeneous and isotropic on their respective conductivities.

We also set the following conditions:

- Thermal conductivities depend only on the material, and therefore are constant:

$$k_0 = 1\frac{\mathrm{W}}{\mathrm{m^2\,K}} \qquad \text{and} \qquad k_+ = 100\frac{\mathrm{W}}{\mathrm{m^2\,K}}.$$

- All structures have a square aspect ratio with $L = H = 0.1\mathrm{m}$.

- The heat-sink is located on the middle of the west side of the structure.

- The heat-sink has Dirichlet boundary conditions: $T_S = 0°\mathrm{C}$.

- All other boundaries are adiabatic (Neumann boundary conditions): $\nabla T \cdot \mathbf{n} = 0$.

Notation

We use the following notation to describe the sets involved in the VP-problem:

- $\mathbf{x} \in \Omega$ = two-dimensional spatial field.

  We set $\Omega = \Omega_0 \cup \Omega_+$ where

  - Ω_0 = portion of Ω that has conductivity k_0.
  - Ω_+ = portion of Ω with conductivity k_+. This is the portion of the space with high-conductivity material.

2.1.2 The Optimization Problem

Using the above established notation, we develop the following optimization problem:

$$\begin{aligned} &\text{minimize} && f(T) && \text{for } \Omega_+ \\ &\text{subject to} && \nabla \cdot (k\nabla T) + q = 0 \\ & && \mathbf{x} \in \Omega \end{aligned} \tag{2.1}$$

The objective function $f(T)$ varies depending on desired design outcomes. Some possible objective functions include average temperature (used in the implementation in this paper), temperature variance, and maximum temperature.

Additionally, we create a constraint upon this problem to limit the quantity of k_+ material available.

$$\int_{\Omega_+} \mathrm{d}\mathbf{x} = \int_{\Omega} \delta_+ \, \mathrm{d}\mathbf{x} \leq V \quad \text{where} \begin{cases} \delta_+ = 0 & \text{if } \mathbf{x} \in \Omega_0 \\ \delta_+ = 1 & \text{if } \mathbf{x} \in \Omega_+ \end{cases} \tag{2.2}$$

Inequality (2.2) imposes a cap on the maximum volume (V) of Ω_+ and hence limits the available amount of high-conductivity material that can be applied to the domain. If we did not have this constraint, the optimal solution would be to set $\Omega_+ = \Omega$, making the entire domain have high conductivity material. However, this is not an interesting or realistic problem and applying the new constraint turns the problem into optimization of the distribution of Ω_0 and Ω_+ regions to minimize the chosen objective function.

Penalization Process

The problem of whether to place high conductivity material in a particular location or not is discrete in nature. This is unfortunate as continuous optimization problems are generally easier to solve. In particular, we cannot apply some of the optimization methods described earlier, such as gradient descent, to a discrete optimization problem as they require the

optimization variables to be continuous.

The SIMP method has a clever way of getting around this particular issue of discrete variables: create a continuous function that allows for a "mix" of the two conductive materials. This function turns our discrete variables into a continuous one, allowing us to apply the methods used in continuous optimization problems. However, in reality, we cannot actually mix the two conductive materials and therefore need a solution that produces a binary 1—0 structure. That is, our final result needs to either have conductivity k_0 or k_+ at each point, not some fraction of each. Therefore, in each iteration of the SIMP process, we *penalize* the mixing of the material. Keeping this in mind we introduce a design parameter $\eta \in [0, 1]$ that controls the amount of mixing of the two materials:

$$k(\eta) = k_0 + (k_+ - k_0)\,\eta^p \qquad \text{with} \qquad 0 \leq \eta \leq 1 \qquad \text{and} \qquad p \geq 1. \tag{2.3}$$

An added bonus of this formulation of $k(\eta)$ is that it is of the form (1.6), and hence a convex function! Notice that when $\eta = 0$, $k(0) = k_0$ and when $\eta = 1$, $k(1) = k_+$. The value p in (2.3) is the *penalization parameter*. p aids in the convergence process; without p the SIMP method converges to a structure that is not 1—0: a composite structure where finite-volumes are made up of different proportions of k_0 and k_+ materials.

To converge to a binary 1—0 structure, we gradually increase p beginning from $p = 1$. Increasing p larger than 1 puts the objective function in (2.1) at a disadvantage if $\eta \neq 1$. Once p gets much larger than 1, the second term in $k(\eta)$ of (2.3) becomes much smaller than k_0 and hence $k(\eta) \approx k_0$ for values of $\eta \neq 1$. As a result, when trying to optimize $f(T)$, values of η in $(0, 1)$ are penalized which leads to design parameters taking on values of 0 or 1, creating a 1—0 structure.

We can think of increasing p as increasing the cost of adding k_+ material to the design. The volume constraint (2.2) caps the amount of k_+ material we may add, and as p increases, the optimizer needs to choose between placing ever more expensive k_+ material or opting for

k_0 material, which caries with it no cost. Therefore, as the process continues, conductivity of the control volumes is either firmly k_+ (1 in the 1—0 terminology) or decreased to k_0 (the 0 in 1—0) and intermediate values are phased out, producing a binary design.

2.1.3 Finite Volume Method Discretization

Many discretization methods could be used to numerically solve the heat equation (1.1). In our implementation of the SIMP algorithm, the Finite Volume Method (described earlier in §1.1.2) was used. FVM is used to discretize the formation of the heat equation in (2.1):

$$\nabla \cdot (k \nabla T) + q = 0. \tag{2.4}$$

We create a rectangular grid of $N_T = m \times n$ temperature control volumes of size $\Delta x \times \Delta y$ (solid squares in Figure 2.1). Each element is indexed by $1 \leq i \leq m$ and $1 \leq j \leq n$, where i refers to the row and j the column of the control volume. The volume indexed $(i, j) = (1, 1)$ is located in the upper-left and $(i, j) = (m, n)$ in the bottom-right corner of the object. The temperature over the area of the temperature control volume (i, j) is considered to be $T^{i,j}$.

Around the upper left corner of each temperature control volume we create a corresponding design element (dashed squares in Figure 2.1). (Staggering the control volume and design grids helps avoid checkerboard solutions, discussed in §3.1.1.) The area within each (i, j) design element has conductivity $k^{i,j} = k(\eta^{i,j})$ as evaluated by (2.3).

In this staggered grid scheme, one of the grids contains the information related to the temperature scalars and the other stores information related to the design parameters, η.

In order to employ the finite volume method, it is necessary to be able to calculate the temperature fluxes along the boundaries of each control volume. To do this, we need to have a value for the conductivity along the faces of each control volume. Notice that the faces of the temperature control volumes lie within two adjacent design regions, which implies that there are two different conductivites along that face. To create a consistent conductivity

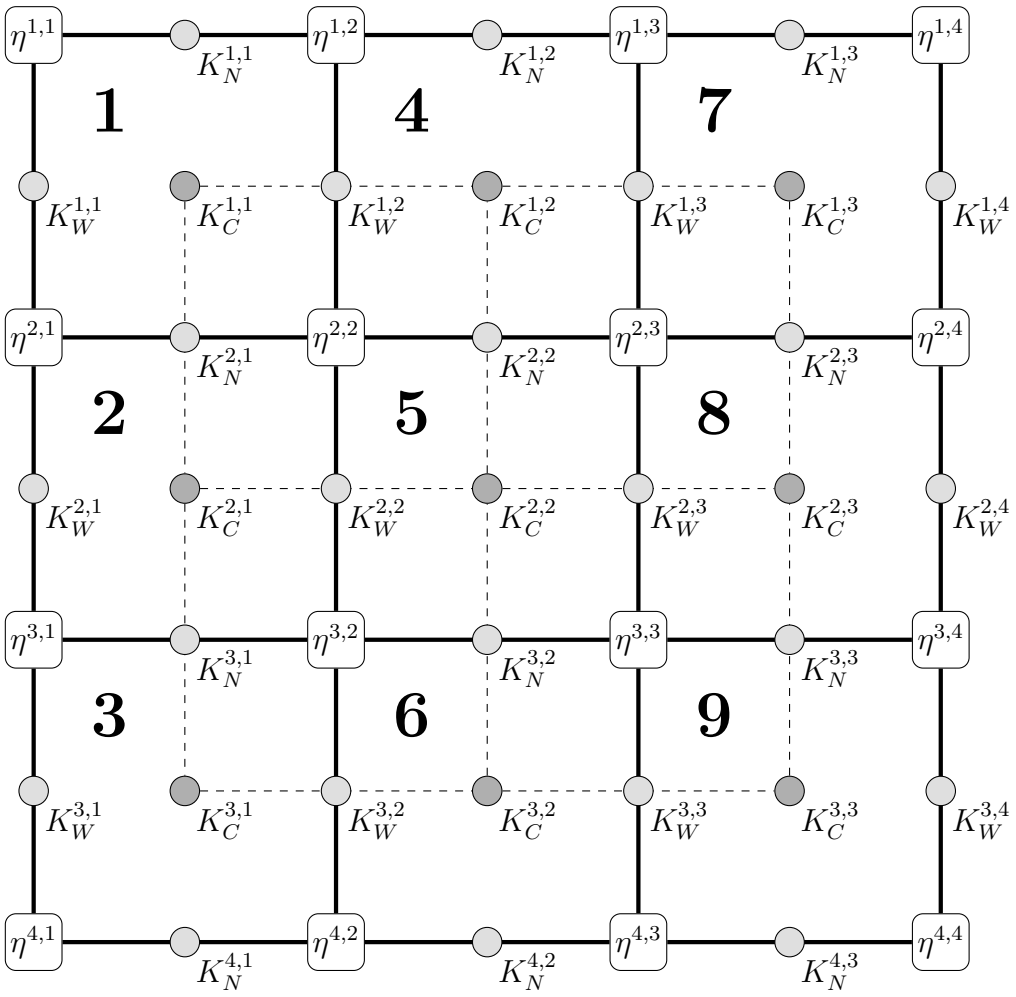


Figure 2.1: Overlayed Temperature (—) and Design (– – –) grids with 4×4 Design Element ($\eta^{i,j}$) and 3×3 Temperature Control Volume ($K_C^{i,j}$) nodes. $K_N^{i,j}$ and $K_W^{i,j}$ indicate nodes at the North and West boundaries, respectively, of each Temperature Control Volume. Each Temperature control volume is numbered beginning in the upper left and continuing column-by-column, left-to-right.

along the control volume wall, we average (using either an arithmetic or harmonic mean) the conductivity of the two adjacent design nodes. Hence, the conductivity along the West face of control volume (i, j), denoted by $k_W^{i,j}$, is given by

$$\text{Arithmetic Mean: } k_W^{i,j} = \frac{k^{i,j} + k^{i+1,j}}{2} \qquad \text{or} \qquad \text{Harmonic Mean: } k_W^{i,j} = 2\left(\frac{1}{k^{i,j}} + \frac{1}{k^{i+1,j}}\right)^{-1}. \tag{2.5}$$

Similarly, the conductivity along the North face of control volume (i, j), denoted by $k_N^{i,j}$, is given by

$$\text{Arithmetic Mean: } k_N^{i,j} = \frac{k^{i,j} + k^{i,j+1}}{2} \qquad \text{or} \qquad \text{Harmonic Mean: } k_N^{i,j} = 2\left(\frac{1}{k^{i,j}} + \frac{1}{k^{i,j+1}}\right)^{-1}. \tag{2.6}$$

For temperature control volume (i, j), the finite volume method discretizes (2.4) into the following linear equation

$$K_C^{i,j} T^{i,j} = K_W^{i,j} T^{i,j-1} + K_W^{i,j+1} T^{i,j+1} + K_N^{i,j} T^{i-1,j} + K_N^{i+1,j} T^{i+1,j} + \Delta x \Delta y Q^{i,j}, \tag{2.7}$$

where the $K^{i,j}$ terms represent the diffusive flux coefficients, $T^{i,j}$ the temperature of control volume (i, j), and $Q^{i,j}$ the heat generation of volume (i, j).

The value of the flux at the center node of the control volume is equal to the total flux through the volume faces, so we have an additional equation to pair with (2.7):

$$K_C^{i,j} = K_W^{i,j} + K_W^{i,j+1} + K_N^{i,j} + K_N^{i+1,j} \tag{2.8}$$

The K_W and K_N coefficients are dependent on the thermal conductivity and cross-sectional area of their corresponding faces:

$$K_W^{i,j} = \frac{k_W^{i,j} \Delta y}{\Delta x} \qquad \text{and} \qquad K_N^{i,j} = \frac{k_N^{i,j} \Delta x}{\Delta y} \tag{2.9}$$

The domain has Neumann boundary conditions everywhere except for the heat sink. This is very easy to implement with the finite volume method: there is no flux through these boundaries, so the flux coefficients are 0. To account for the heat-sink in the middle of the left boundary of the domain, which has Dirichlet boundary conditions, we add a "ghost cell" to the other side of the boundary that has temperature opposite the cell along the heat-sink in the domain. Adding this ghost cell in this way averages out the temperature on the boundary to zero.

Putting together (2.5), (2.6), (2.7), (2.8), (2.9), and considering boundary conditions for all N_T control volumes gives us a system of equations that discretize (2.4). Collecting the coefficients K into a matrix, representing the T and Q values as vectors, and doing a little reorganizing, we can represent the system of equations as a matrix equation:

$$\mathbf{KT} = \Delta x \Delta y \mathbf{Q}. \tag{2.10}$$

Let us take a moment to analyze the structure of the matrix $\mathbf{K}$, as it might not be immediately evident to the reader what the elements of this matrix represent. (It took the author some time to interpret the meaning of this matrix.) $\mathbf{K}$ is a sparse, symmetric, and pentadiagonal $mn \times mn$ matrix.

The entries in the matrix $\mathbf{K}$ indicate the coefficient of diffusive flux between numbered temperature control volumes. Notice in Figure 2.1 how the temperature control volumes are numbered down the columns. We can convert between volume index (i, j) and control volume number, #, using a simple function:

$$\#(i, j) = i + m(j - 1). \tag{2.11}$$

Entry $\mathbf{K}[\alpha, \beta]$ is the flux coefficient between volumes number α and β. Since the flux coefficient between volumes α and β is the same as that between β and α, $\mathbf{K}[\alpha, \beta] = \mathbf{K}[\beta, \alpha]$, producing the symmetry of matrix $\mathbf{K}$. Since a particular control volume only interfaces with

adjacent cells (and itself), each row/column will have (up to) five non-zero entries (all other entries are zero since there is no flux between cells that are not in contact with one another), which produces the pentadiagonality and sparsity of $\mathbf{K}$. The i^{th} elements of the size mn vectors $\mathbf{T}$ and $\mathbf{Q}$ are the values of the temperature and heat generation of control volume number i.

Equation (2.10) is solved for $\mathbf{T}$ using any appropriate method, such as LU factorization. In our implementation we used the standard "\" operator in Julia: $\mathbf{T} = \mathbf{K}\backslash\mathbf{Q}$.

2.1.4 Discretized Optimization Problem

Now that we have been able to discretize (2.4), we can update the optimization problem (2.1):

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{T}) \\ \text{subject to} \quad & \mathbf{KT} = \Delta x \Delta y \mathbf{Q} \\ & \frac{1}{N_T}\mathbf{1}^T\boldsymbol{\eta} \leq \overline{\phi} \\ & \mathbf{k} = k_0\mathbf{1} + (k_+ - k_0)\boldsymbol{\eta}^p \\ & 0 \leq \boldsymbol{\eta} \leq 1 \text{ and } p \geq 1 \end{aligned} \tag{2.12}$$

Here $\boldsymbol{\eta}$ represents the vector of design parameter values for each control volume. Additionally, we introduce the variable $\overline{\phi}$ which represents the maximum porosity, the maximal fraction of high-conductivity material allowed within the domain. N_T equals the total number of temperature control volumes.

For our implementation of the SIMP method for this problem, the average temperature was chosen as the objective function:

$$f_{av}(\mathbf{T}) = \frac{1}{N_T}\mathbf{1}^T\mathbf{T}. \tag{2.13}$$

Additionally, we opted to have constant and uniform heat generation for each control

volume. Specifically, in our implementation we set the heat generation for each volume to be 1:

$$\mathbf{Q} = \mathbf{1}. \tag{2.14}$$

We use the Method of Moving Asymptotes (MMA) to update the design parameters $\boldsymbol{\eta}$ throughout the optimization process. To create a local and convex approximation of the problem (see §1.4), MMA requires both function and constraint evaluations, as well as evaluations of the respective gradients. Hence, we need to calculate the gradients of the average temperature function and porosity constraint. The gradients of the functions with respect to the design parameters indicate the *sensitivity* of those functions to changes in $\boldsymbol{\eta}$, and hence analysis of these derivatives is called *sensistivity analysis*.

Sensitivity Analysis

We seek to find the partial derivatives of f_{av} (the objective function) and ϕ (the constraint) with respect to an arbitrary design element η_ℓ so that we can form the gradients. That is, we are looking to find expressions for

$$\frac{\partial f_{av}}{\partial \eta_\ell} \quad \text{and} \quad \frac{\partial \phi}{\partial \eta_\ell}.$$

The adjoint method is employed to make the calculation of the partial derivative of f_{av} easier to find. By assuming (2.10) is true, we add a clever form of $\mathbf{0}$ to the objective function f_{av}:

$$f_{av}(\mathbf{T}) = \frac{1}{N_T}\mathbf{1}^T\mathbf{T} + \boldsymbol{\lambda}^T \underbrace{(\mathbf{K}\mathbf{T} - \mathbf{Q})}_{=0}. \tag{2.15}$$

The new variable, $\boldsymbol{\lambda} = \left(\lambda_1, \lambda_2, \ldots, \lambda_{N_T}\right)$, is called the adjoint vector and behaves like a Lagrange multiplier [2]. Differentiating (2.15) with respect to an arbitrary design variable η_ℓ gives

$$\frac{\partial f_{av}}{\partial \eta_\ell} = \frac{\mathbf{1}^T}{N_T}\frac{\partial \mathbf{T}_i}{\partial \eta_\ell} + \boldsymbol{\lambda}^T\left(\frac{\partial \mathbf{K}}{\partial \eta_\ell}\mathbf{T} + \mathbf{K}\frac{\partial \mathbf{T}}{\partial \eta_\ell} - \frac{\partial \mathbf{Q}}{\partial \eta_\ell}\right). \tag{2.16}$$

The heat-generation rate is assumed to be homogeneous over design volumes and independent of the conductive material, so it does not depend on the design parameters:

$$\frac{\partial \mathbf{Q}}{\partial \eta_\ell} = 0. \tag{2.17}$$

Factoring terms including the partial derivative of $\mathbf{T}$ and taking into account (2.17), (2.16) becomes

$$\frac{\partial f_{av}}{\partial \eta_\ell} = \underbrace{\left(\boldsymbol{\lambda}^T \mathbf{K} + \frac{\mathbf{1}^T}{N_T}\right)}_{(\star)} \frac{\partial \mathbf{T}}{\partial \eta_\ell} + \boldsymbol{\lambda}^T \frac{\partial \mathbf{K}}{\partial \eta_\ell} \mathbf{T}. \tag{2.18}$$

Hence $\frac{\partial \mathbf{T}}{\partial \eta_\ell}$ is eliminated from the expression if $\boldsymbol{\lambda}$ is taken such that $(\star)$ in (2.18) equals 0:

$$\boldsymbol{\lambda}^T \mathbf{K} + \frac{\mathbf{1}^T}{N_T} = 0 \iff \mathbf{K}\boldsymbol{\lambda} = -\frac{\mathbf{1}}{N_T}\mathbf{1}^T. \tag{2.19}$$

(2.19) is solved the same way as (2.10). Thus, using the values for $\boldsymbol{\lambda}$ from solving (2.19),

$$\frac{\partial f_{av}}{\partial \eta_\ell} = \boldsymbol{\lambda}^T \frac{\partial \mathbf{K}}{\partial \eta_\ell} \mathbf{T}. \tag{2.20}$$

$\boldsymbol{\lambda}$ and $\mathbf{T}$ are found by solving (2.19) and (2.10), respectively. Thus, it remains to find $\frac{\partial \mathbf{K}}{\partial \eta_\ell}$. Recall that matrix $\mathbf{K}$ contains the coefficients of control volume boundary conductivities, which depend on the penalization equation $\mathbf{k}(\eta)$ (2.3). Hence, using the chain rule,

$$\frac{\partial \mathbf{K}}{\partial \eta_\ell} = \frac{\partial \mathbf{K}}{\partial \mathbf{k}} \frac{\partial \mathbf{k}}{\partial \eta_\ell}. \tag{2.21}$$

By (2.3)

$$\frac{\partial \mathbf{k}}{\partial \eta_\ell} = \begin{cases} p\,(k_+ - k_0)\,\eta_\ell^{p-1} & \text{for the } \ell\text{th element,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

Using (2.5), (2.6), (2.8), and (2.9) one is able to find $\frac{\partial \mathbf{K}}{\partial \mathbf{k}}$, the form of which will vary slightly

based on whether an arithmetic or harmonic mean is chosen. For the arithmetic mean

$$\frac{\partial \mathbf{K}}{\partial \mathbf{k}} = \begin{cases} -\frac{1}{2}\frac{\Delta y}{\Delta x} & \text{if } k \text{ corresponds to a } k_W \text{ face,} \\ -\frac{1}{2}\frac{\Delta x}{\Delta y} & \text{if } k \text{ corresponds to a } k_N \text{ face,} \\ -\left(\frac{\Delta y}{\Delta x} + \frac{\Delta x}{\Delta y}\right) & \text{if } k \text{ corresponds to a } k_C \text{ node.} \end{cases} \tag{2.23}$$

It is now possible to compute $\dfrac{\partial f_{av}}{\partial \eta_\ell}$ using the above. We still require $\dfrac{\partial \phi}{\partial \eta_\ell}$.

The porosity function is

$$\phi(\boldsymbol{\eta}) = \frac{1}{N_\eta}\mathbf{1}^T\boldsymbol{\eta} \tag{2.24}$$

where N_η indicates the number of design elements. The optimizer used in the implementation (NLopt) requires constraints to be in the form of a function less than or equal to zero, so we reorganize the form of (2.24) in (2.12):

$$\mathbf{1}^T\boldsymbol{\eta} - N_T\overline{\phi} \leq 0 \tag{2.25}$$

Therefore, we set the left-hand side of this inequality to be our porosity function:

$$\tilde{\phi}\,(\boldsymbol{\eta}) = \mathbf{1}^T\boldsymbol{\eta} - N_T\overline{\phi} \tag{2.26}$$

Therefore,

$$\frac{\partial \tilde{\phi}}{\partial \eta_\ell} = 1 \tag{2.27}$$

Now, we update the optimization problem one last time with the results from (2.13), (2.14), and (2.25).

$$
\begin{aligned}
\text{minimize} \quad & f_{av}(\mathbf{T}) = \tfrac{1}{N_T}\mathbf{1}^T\mathbf{T} \\
\text{subject to} \quad & \mathbf{KT} = \Delta x \Delta y \mathbf{1} \\
& \mathbf{1}^T\boldsymbol{\eta} - N_T\overline{\phi} \leq 0 \\
& \mathbf{k} = k_0\mathbf{1} + (k_+ - k_0)\boldsymbol{\eta}^p \\
& 0 \leq \boldsymbol{\eta} \leq 1 \text{ and } p \geq 1
\end{aligned}
\tag{2.28}
$$

(2.28) is finally in a form that we can input into the NLopt optimizer using the MMA algorithm and produce possible design solutions to the VP heat conduction problem.

Chapter 3

Implementation and Results

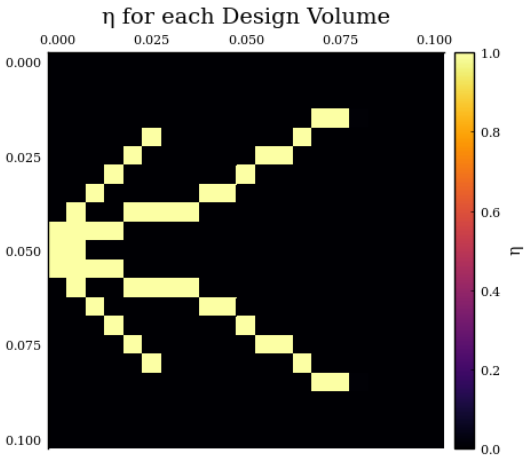
3.1 SIMP Implementation in Julia

The SIMP algorithm consists of two main loops. The inner-loop consists of the optimization process facilitated by the Method of Moving Asymptotes while the outer-loop increases the penalization parameter p after each inner-loop has completed. The whole process is repeated until a desired threshold is met, such as a small enough norm between successive minimal objective function evaluations. The algorithm implementation in the Julia language can be found in A.4 and is available for download on GitHub.

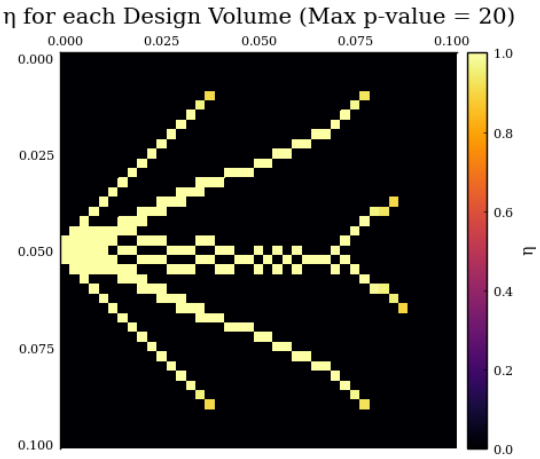
Figure 3.1 shows design results for domains of varying numbers of control volumes. Notice in Figures 3.1b and 3.1c that the ends of the "tentacles" contain some control volumes with $\eta \neq 1$. This is due to the cap put on p in the algorithm. It became necessary to impose a maximum value for p as the number of control volumes increased in order to complete the algorithm within a reasonable amount of time. With higher resolutions (i.e. more control volumes), it appears that the algorithm continuously is trading high conductivity material between the ends of the symmetric branches above and below the horizontal midline.

It is interesting to note the symmetry in the designs. In [3] the authors chose to take advantage of this symmetry and compute only the upper half of the domain and then reflect that design across the central horizontal axis. While 3.1a is certainly symmetric across the central horizontal axis, we notice some variations in 3.1b and 3.1c in the central branches.

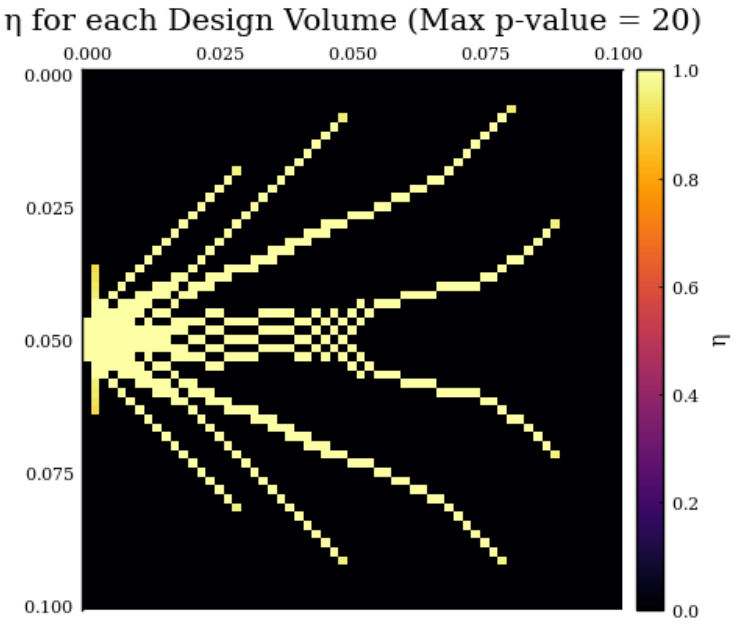
As we move away from square grids, we see a bit less symmetry. Figures 3.2 and 3.4 are designs over rectangular control volumes and have asymmetric central branches.



(a) 20×20 control volumes



(b) 40×40 control volumes (Max $p = 20$)



(c) 60×60 control volumes (Max $p = 20$)

Figure 3.1: Final design outputs of SIMP algorithm for domains with varying numbers of control volumes.

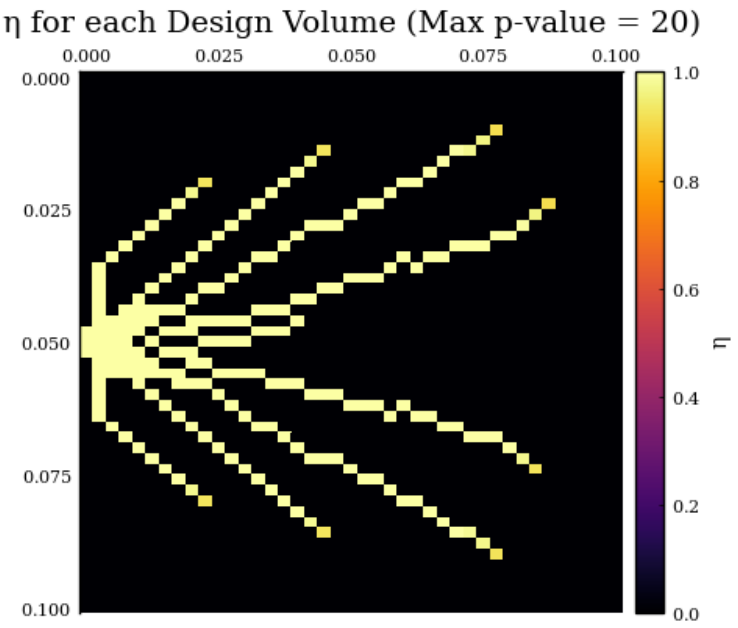


Figure 3.2: 50×40 control volumes (Max $p = 20$)

Notice the alternating pattern in the conductivities of control volumes at the center of Figure 3.1c. This phenomenon is known as "checkerboarding" and is a consequence of our discretization scheme.

3.1.1 "King Me": Avoiding the Checkerboard Problem

If one were to optimize conductive material placement to increase heat transfer on a standard rectangular grid, a simplistic solution would be to create a grid of alternating material types in each adjacent rectangle, like a checkerboard. This way, the heat is always flowing to adjacent cells. While this may indeed increase heat transfer, it doesn't necessarily decrease the average temperature in our object (or transfer the heat towards the heat-sink). Hence, avoiding this non-physical checkerboard solution is of concern.

In order to solve the heat equation (1.1), we employ the Finite Volume Method (described earlier). This involves splitting up our space into a finite number of control volumes. The checkerboard pattern emerges when the solution of our optimization process converges to a 1—0 structure which has some meshes that successively belong to the Ω_0 and Ω_+ sets (i.e.:

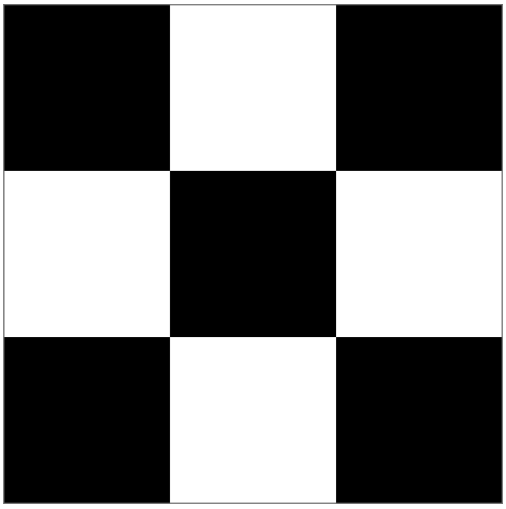


Figure 3.3: A checkerboard pattern result on a 3-by-3 control volume grid. The black spaces represent areas where $\eta = 1$ and the white spaces represent areas where $\eta = 0$ in (2.3). This design results in adjacent regions of alternating thermal conductivites k_+ and k_0, artificially maximizing heat transfer between control volumes.

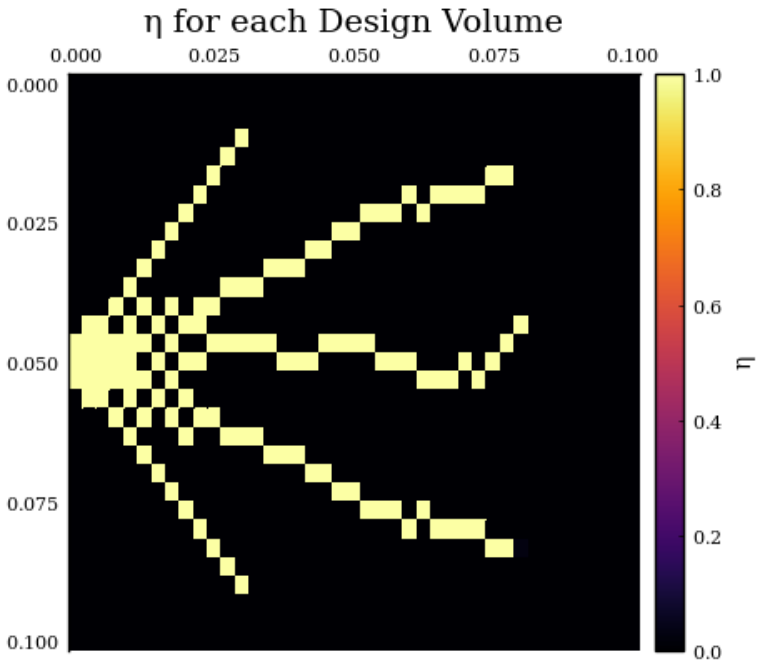


Figure 3.4: Output of SIMP algorithm for 30×40 temperature control volumes. Notice the local checkerboarding around $(0.05, 0.015)$.

adjacent grid volumes have alternating thermal conductivites). As a result, the heat transfer within the structure between k_+ and k_0 regions is maximized, artificially increasing the impact of adding k_+ material on the temperature T, which in turn minimizes the objective function in (2.1) [7]. Typically, this pattern occurs locally but then spreads throughout the entire structure through successive iterations of the optimization process. However, in the real world, these checkerboard placements of our conductive materials do not actually have the effect of lowering the average temperature in structures. In fact, the checkerboard example in Figure 3.3 doesn't even direct heat towards the heat-sink on the left wall of the structure.

In order to avoid obtaining checkerboard solutions from our optimization process, we employ two separate staggered grids for our temperature and design variables (see Figure 2.1). We need to employ some extra equations in order to translate between design and temperature variables, but this strategy helps solve the issue of convergence to checkerboard solutions.

Available literature on topology optimization propose other solutions as well, such as using a weighted average over neighboring nodes when finding design sensitivities and introducing constraints on the norm of the design gradients [5]. These are methods that would be of interest to implement in future work.

3.1.2 Algorithm Results

In our implementation, the algorithm is initialized with $\eta = 0.05$ for each control volume, the maximal porosity $\overline{\phi}$ was 10%, and p was increased in steps of 0.05 beginning at $p = 1$ for each outer-loop iteration. We can see in Figure 3.5 that after one outer-loop iteration, the algorithm has already zeroed the design parameters for many of the control volumes.

With each outer-loop iteration and increasing p, the objective function values for the average temperature increases quite rapidly until it begins to level off. For a 60×60 control volume design, the temperature values begin to converge around $p = 5$ (Figure 3.6). The

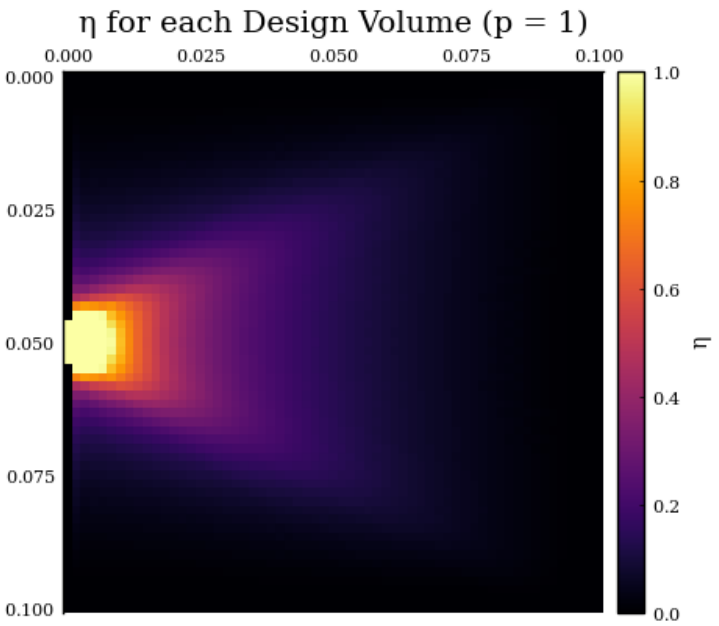


Figure 3.5: Design after one outer-loop iteration for a 60×60 control volumes.

increase in temperature is due to concentration of the k_+ material and penalizing fractional portions of η.

In general, the number of inner-loop iterations seems to quickly decrease as p increases, but there are fluctuations that occur occasionally (Figure 3.7). It is not clear as to why the inner-loop converges after a small number of iterations (often 2 or 3) for one p-value and then will require over 100 iterations for a following p-value. Other literature on the VP heat conduction problem did not have an explanation for this behavior either [3].

It is interesting to note that the fluctuations in the number of inner-loop iterations seem to be of greater magnitude for the rectangular control volume examples tested. (See Figure 3.7a versus Figure 3.7b.)

The algorithm was also run with high initial p-values. Rather than starting with $p = 1$, Figure 3.8a shows the resulting design for initializing the algorithm with $p = 5$. In this case, the algorithm converged very quickly: one outer-loop iteration. Here, the algorithm determines that it is best to densely place the expensive k_+ material around the heat-sink. If the initial penalization parameter is too high, the algorithm appears to stall at the initial

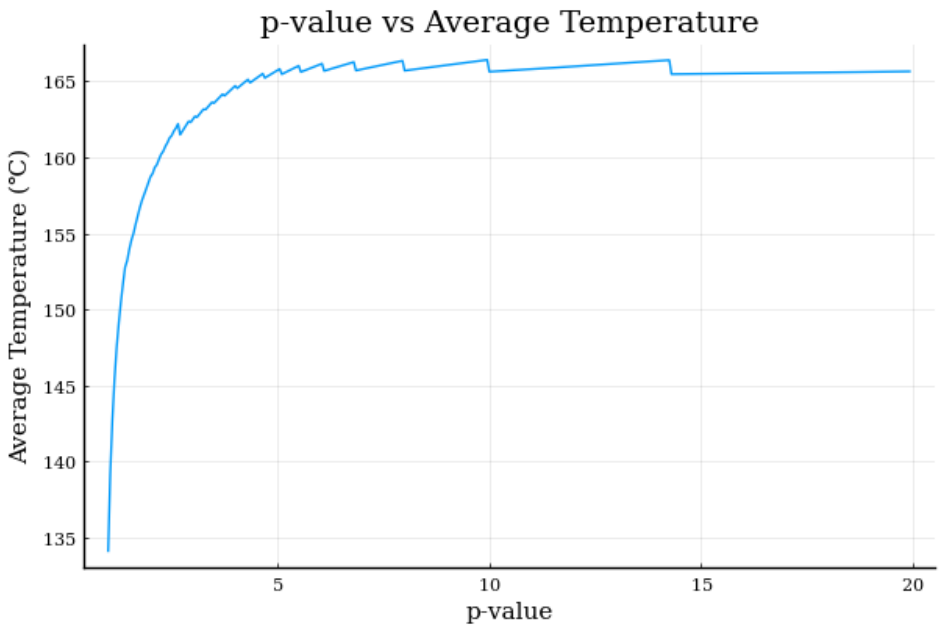
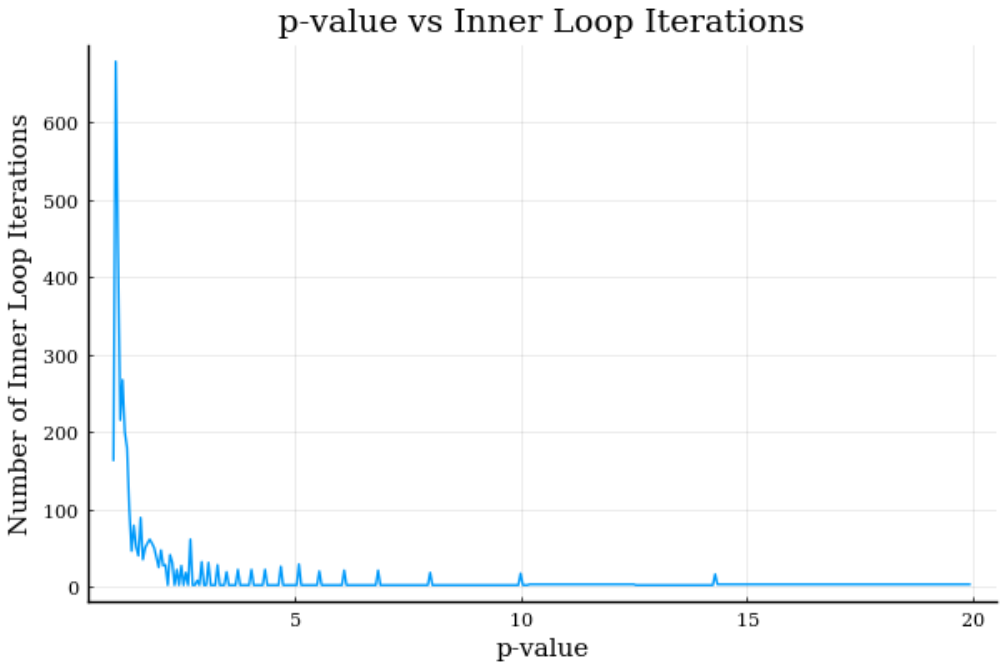


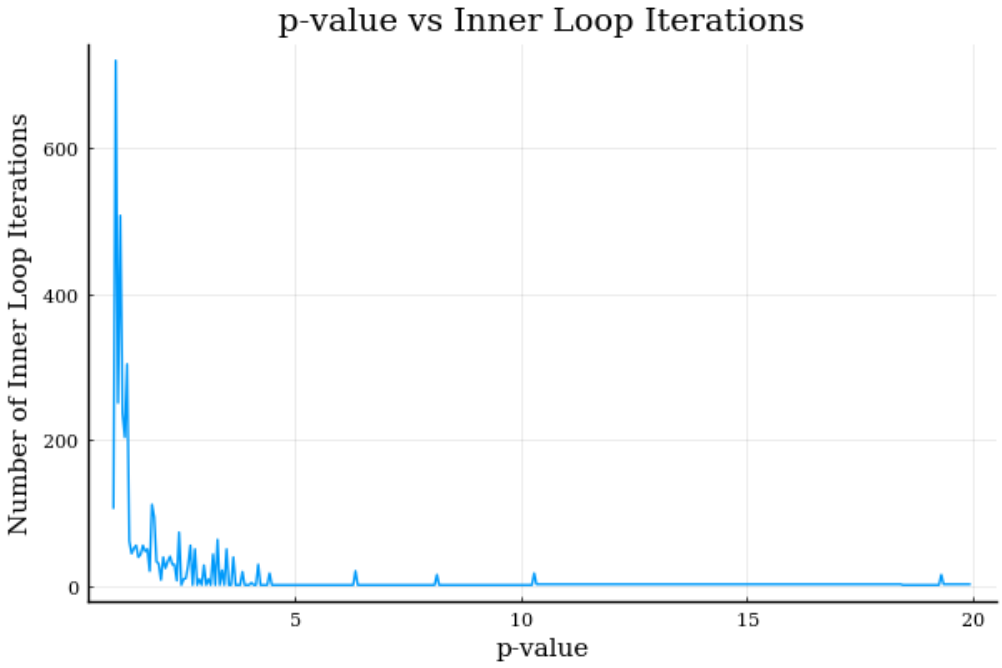
Figure 3.6: p-value plotted against Average Temperature Evaluation for 60×60 control volumes with a maximum p-value of 20.

guess. In Figure 3.8b the algorithm was initialized with $p = 19$. The algorithm ran 3 outer-loop iterations, but the final design is the same as the initial input design. It appears that with such a high penalization parameter it costs too much to change the conductivity at any control volume and the solution the optimizer outputs is the initial guess.

For the designs tested, it appears that the total algorithm runtime increases quadratically with the total number of control volumes in the design (Figure 3.9).

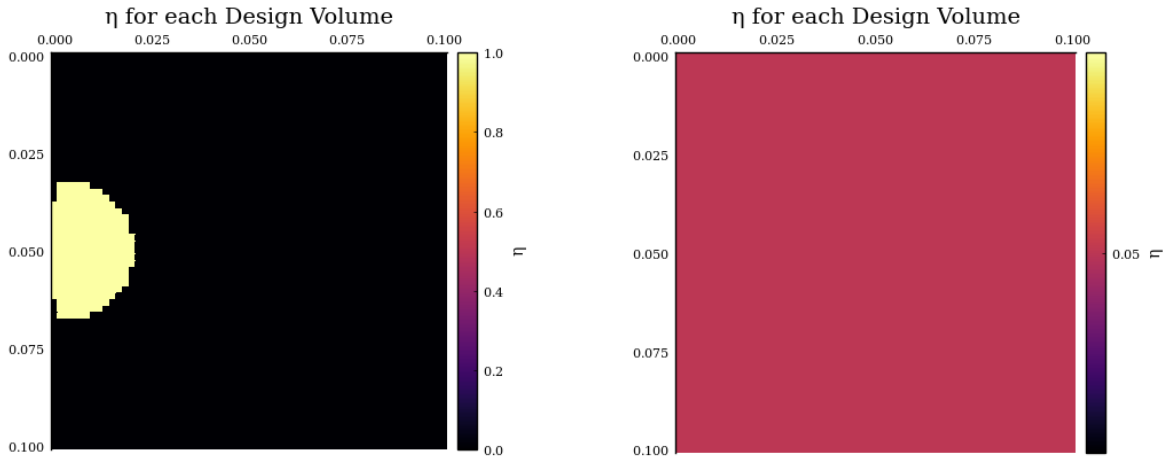


(a) p-value plotted against the number of inner loop iterations for 60×60 control volumes with a maximum p-value of 20.



(b) p-value plotted against the number of inner loop iterations for 50×40 control volumes with a maximum p-value of 20.

Figure 3.7: p-value vs. Number of Inner Loop Iterations for Square and Rectangular Control Volumes



(a) 60×60 Control Volumes with Initial $p = 5$. (b) 60×60 Control Volumes with Initial $p = 19$.

Figure 3.8: SIMP Algorithm design outputs for initial p-values greater than 1.

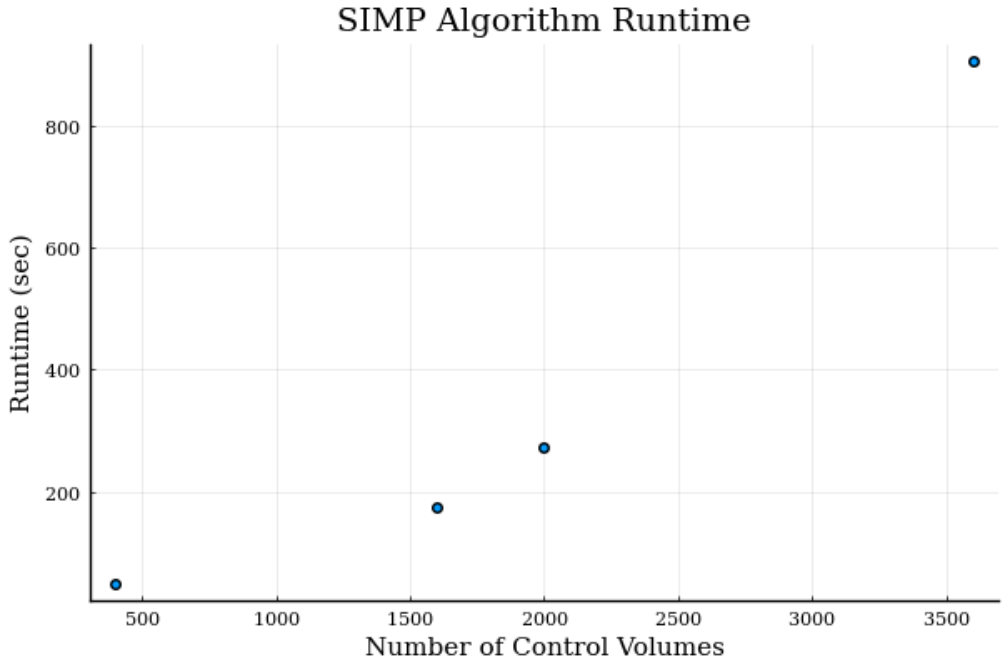


Figure 3.9: Plot of runtime for SIMP algorithm for various numbers of control volumes.

Conclusion

The Volume-to-Point (VP) heat conduction problem seeks to find the optimal placement of high-conductivity material to minimize a property of the design, such as average temperature. In order to understand this problem and its solutions, we needed to study the partial differential heat equation (1.1) and its discretization via the Finite Volume Method. Furthermore, knowledge of optimization problems and the optimization algorithm of the Method of Moving Asymptotes was needed to implement the Solid Isotropic Material with Penalization (SIMP) algorithm to find solutions to the VP problem. We managed to write a working implementation of the SIMP method in the Julia language that can be easily modified to different sized rectangular design domains and is able to be run on a simple desktop computer. Areas of future investigation include implementing the harmonic average filtering scheme in (2.5) and (2.6) into the program. Additionally, we began investigations concerning the use of the SIMP method to maximize the eigenvalue band-gap in acoustic materials, but there was not enough time to produce results. I am also interested in investigating the use of the SIMP method for designing efficient packaging materials.

References

[1] S. P. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge Univ. Pr., 2004.

[2] S. G. Johnson, *Notes on adjoint methods for 18.335*. WebPage, Apr. 2021.

[3] G. Marck, M. Nemer, J.-L. Harion, S. Russeil, and D. Bougeard, *Topology optimization using the SIMP method for multiobjective conductive problems*, Numerical Heat Transfer, Part B: Fundamentals, 61 (2012), pp. 439–470.

[4] J. R. Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, Quake Project, (1994).

[5] O. Sigmund and J. Petersson, *Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima*, Structural Optimization, 16 (1998), pp. 68–75.

[6] K. Svanberg, *The method of moving asymptotes—a new method for structural optimization*, International Journal for Numerical Methods in Engineering, 24 (1987), pp. 359–373.

[7] H. K. Versteeg, *An Introduction to Computational Fluid Dynamics : The Finite Volume Method*, Pearson Education Ltd, Harlow, England New York, 2007.

Appendix A

Julia Codes

A.1 Backtracking Line Search

Implementation of the Backtracking Line Search in Julia with default values for the parameters being $\alpha = 0.25$ and $\beta = 0.5$.

```
function ln_srch(d_dir,x,f,fx,dfx;alpha=0.25,beta=0.5)
    t = 1
    x1 = x+t*d_dir
    y1 = f(x1)
    y2 = fx+alpha*t*(dfx)'*d_dir
    while y1 > y2
        t = beta*t
        x1 = x+t*d_dir
        y1 = f(x1)
        y2 = fx+alpha*t*(dfx)'*d_dir
    end
    return t
end
```

A.2 Gradient Descent

```
using LinearAlgebra


#Function to Optimize
```

```
f(x)=(x[2])^3-x[2]+(x[1])^2-3x[1]

#Gradient of Function
df(x)=[2x[1]-3,3x[2]^2-1]

#Initial Point
x=[0,0]

#Line Search Algorithm
function ln_srch(d_dir,x,f,fx,dfx;alpha=0.25,beta=0.5)
    t = 1
    x1 = x+t*d_dir
    y1 = f(x1)
    y2 = fx+alpha*t*(dfx)'*d_dir
    while y1 > y2
        t = beta*t
        x1 = x+t*d_dir
        y1 = f(x1)
        y2 = fx+alpha*t*(dfx)'*d_dir
    end
    return t
end

#Gradient Descent Algorithm
function grad_d(f,df,x)
    d_dir = -df(x)
    t = ln_srch(d_dir,x,f,f(x),df(x))
    x = x + t*d_dir
    return x
end

#Compute Minimum for Defined Tolerance
while norm(df(x))>0.00001
    global x = grad_d(f,df,x)
end

display(x)
```

A.3 Nonlinear Conjugate Gradient

```
using LinearAlgebra

i = 0
k = 0

#Function to Optimize
f(x)=(x[2])^3-x[2]+(x[1])^2-3x[1]

#Gradient of Function
df(x)=[2x[1]-3,3x[2]^2-1]

#Hessian of Function
hf(x)=[2 0; 0 6x[2]]

#Initial Point
x = [0,0]

n = size(x)[1]

r = -df(x)

d = r

delta_new = (r')*r

delta_0 = delta_new

#Choose Max Iterations
i_max = 100

#Choose Max Newton-Raphson Iterations
j_max = 10

#Set CG Error Tolerance
epsilon_CG = 0.5

#Set Newton-Raphson Error Tolerance
```

```
epsilon_NR = 0.5

while (i < i_max) && (delta_new > (((epsilon_CG)^2)*(delta_0)))
    global j = 0
    global delta_d = (d')*d
    while true
        global alpha = -((df(x))'*d)/((d')*hf(x)*d)
        global x = x + alpha*d
        global j = j + 1
        (j < j_max) && ((alpha)^2*(delta_d) > (epsilon_NR)) || break
    end
    global r = -df(x)
    global delta_old = delta_new
    global delta_new = (r')*r
    global beta = (delta_new)/(delta_old)
    global d = r + beta*d
    global k = k + 1
    if (k == n) || (((r')*d) <= 0)
        global d = r
        global k = 0
    end
    global i = i + 1
end

display(x)
```

A.4 SIMP Method for Volume-to-Point Heat Conduction Problem on 60x60 Control Volume Grid

```
using NLopt, SparseArrays, LinearAlgebra, LaTeXStrings, Plots
pyplot()

##########################
## Fixed Variable Input ##
##########################
```

```
p = 1

p_max = 20

p₊ = 0.05

m = 60

n = 60

k₀ = 1.0

k₊ = 100.0

xlen = 0.1

ylen = 0.1

ε₀ = 1e-3 # Outer loop error tolerance

εᵢ = 1e-4 # Inner Loop error tolerance

q = 1e4

##########################
## Compute size of each ##
##   control volume     ##
##########################

Δx = xlen / n
Δy = ylen / m

########################################
## Create Optimization Problem Structure ##
## Using MMA with dimentions (m+1)*(n+1) ##
########################################

opt = Opt(:LD_MMA, (m + 1) * (n + 1))

########################
```

```
## Average Temperature ##
## Objective Function  ##
#########################

function av_temp(
    η::Vector,
    grad::Vector,
    p,
    m,
    n,
    xlen = 0.1,
    ylen = 0.1,
    k₀ = 1.0,
    k₊ = 100.0,
)

    ########################
    ## Assemble K Matrix ##
    ########################

    ############################
    ## Compute size of each ##
    ##   control volume     ##
    ############################

    Δx = xlen / n
    Δy = ylen / m

    η = reshape(η, m + 1, n + 1)

    # Define Conductivity Penalization Function for design parameters eta
    k = k₀ .+ (k₊ - k₀) .* η .^ p

    # Control Volumes are designated based on matrix-type coordinates, so that
       volume [i,j] is the control volume in the i-th row and j-th column from
       the upper left.

    # Compute conductivites of temperature control volume boundaries
    # k_W[i,j] = conductivity of "West" boundary of [i,j] control volume

        k_W = 0.5 * (k[1:end-1, :] + k[2:end, :])
```

```
# k_N[i,j] = conductivity of "North" boundary of [i,j] control volume

k_N = 0.5 * (k[:, 1:end-1] + k[:, 2:end])

# Initialize K matrix
K = spzeros((m * n), (m * n))

# Number control volumes based on node coordinates, going column-by-column,
   for m rows and n columns
function cord2num(i, j, m)
cv_num = i + (j - 1) * m
return cv_num
end

# Construct K matrix
# K[x,y] tells the heat flux from temperature volume number x to volume
   number y
for i = 1:m, j = 1:(n-1)
K[cord2num(i, j, m), cord2num(i, j + 1, m)] = -k_W[i, j+1] * (Δy / Δx)
K[cord2num(i, j + 1, m), cord2num(i, j, m)] = -k_W[i, j+1] * (Δy / Δx)
end

for i = 1:(m-1), j = 1:n
K[cord2num(i, j, m), cord2num(i + 1, j, m)] = -k_N[i+1, j] * (Δx / Δy)
K[cord2num(i + 1, j, m), cord2num(i, j, m)] = -k_N[i+1, j] * (Δx / Δy)
end

# Diagonal elements of K balance out column sums
for j = 1:(m*n)
K[j, j] = -sum(K[:, j])
end

######################
## Add in effect of ##
##    Heat Sink     ##
######################

# Add heat sink in middle of left side of material by adding conductivity
   to diagonal element of K in corresponding row
if iseven(m)
```

```
    # Nearest half integer
    hm = m ÷ 2
    K[cord2num(hm, 1, m), cord2num(hm, 1, m)] += k_W[hm, 1] ⋆ (Δy / Δx)
    K[cord2num(hm + 1, 1, m), cord2num(hm + 1, 1, m)] += k_W[hm+1, 1] ⋆ (Δy / Δ
        x)
    else
    hm = m ÷ 2 + 1
    K[cord2num(hm, 1, m), cord2num(hm, 1, m)] += k_W[hm, 1] ⋆ (Δy / Δx)
    end

    ########################
    ## Assemble Q Matrix ##
    ########################

    # Input vector of Heat-Generation rates
    Q = Δx⋆Δy⋆q⋆ones(m, n)

    ########################
    ## Compute T Vector ##
    ########################

    # Solve KT = Q
    T = K \ vec(Q)

    ############################
    ## Gradient Computations ##
    ############################

    if length(grad) > 0

        grad = reshape(grad, m + 1, n + 1)

        ############################
        ##    Compute λ vector    ##
        ## (Dual Vector for f_av) ##
        ############################

        λ = K \ (-ones((m ⋆ n), 1) ⋆ (1 / (m ⋆ n)))

        ########################
        ## Create ∂k/∂η Matrix ##
```

```
        ##########################

        dk = (p * (k₊ - k₀)) .* η .^ (p - 1)

        ############################
        ## Assemble ∂K/∂η Matrix ##
        ############################

        for i = 1:m+1, j = 1:n+1

            ############################
            ## Assemble ∂K/∂k Matrix ##
            ##     for each (i,j)     ##
            ############################

            dK = spzeros((m * n), (m * n))

            if 2 <= j <= n
                for a = max(1, i - 1):min(i, m)
                    dK[cord2num(a, j, m), cord2num(a, j - 1, m)] = -0.5 * (Δy /
                        Δx)
                    dK[cord2num(a, j - 1, m), cord2num(a, j, m)] = -0.5 * (Δy /
                        Δx)
                end
            end
            if 2 <= i <= m
                for b = max(1, j - 1):min(j, n)
                    dK[cord2num(i, b, m), cord2num(i - 1, b, m)] = -0.5 * (Δx /
                        Δy)
                    dK[cord2num(i - 1, b, m), cord2num(i, b, m)] = -0.5 * (Δx /
                        Δy)
                end
            end
            for a = max(1, i - 1):min(i, m), b = max(1, j - 1):min(j, n)
                dK[cord2num(a, b, m), cord2num(a, b, m)] = -sum(dK[cord2num(a,
                    b, m), :])
            end

            ######################
            ## Add in effect of ##
            ##    Heat Sink     ##
```

```
            ######################

            if iseven(m)
                hm = m ÷ 2
                if j == 1 && (hm ≤ i ≤ hm + 1)
                    dK[cord2num(hm, 1, m), cord2num(hm, 1, m)] += 0.5 * (Δy / Δ
                        x)
                end
                if j == 1 && (hm + 1 ≤ i ≤ hm + 2)
                    dK[cord2num(hm + 1, 1, m), cord2num(hm + 1, 1, m)] += 0.5 *
                        (Δy / Δx)
                end
            else
                hm = m ÷ 2 + 1
                if j == 1 && (hm ≤ i ≤ hm + 1)
                    dK[cord2num(hm, 1, m), cord2num(hm, 1, m)] += 0.5 * (Δy / Δ
                        x)
                end
            end

            ############################
            ## Find Nonzero elements ##
            ##  of ∂K/∂η_{i,j} and   ##
            ## Assemble ∂f_av Matrix ##
            ############################
            grad[i, j] = 0.0
            A, B, Va = findnz(dK)
            for k = 1:nnz(dK)
                a = A[k]
                b = B[k]
                v = Va[k]
                grad[i, j] += λ[a] * v * T[b]
            end

            ##############################
            ## (∂K/∂k)*(∂k/∂η) = ∂K/∂η ##
            ##############################

            grad[i, j] *= dk[i, j]
        end
    end
```

```
    ##########################
    ## Compute f(T) = T_avg ##
    ##########################

    f_avg = sum(T) / (m * n)

    ########################
    ## Debugging Messages ##
    ########################

    #println("fav = $f_avg\n", "grad = $grad\n") #Output for Debugging Purposes

    return f_avg
end

#################################
## Porosity Constraint Function ##
#################################

function por(x::Vector, grad::Vector, m, n)
    if length(grad) > 0
        grad .= 1.0
    end
    con = sum(x) - 0.1 * (m + 1) * (n + 1)
    #println("con = $con\n", "grad = $grad\n") #Output for Debugging Purposes
    return con
end

#################################
## Add Objective and Constraints ##
##       to opt structure        ##
#################################

min_objective!(opt, (x, g) -> av_temp(x, g, p, m, n))

inequality_constraint!(opt, (x, g) -> por(x, g, m, n), 1e-8)

opt.lower_bounds = 0
opt.upper_bounds = 1
```

```
opt.xtol_rel = ε_i

η = 0.05 .* ones((m + 1) * (n + 1))

f_0 = 10.0 * av_temp(η, [], p, m, n)

p_vec = []
iter_vec = []
f_av_vec = []
total_iterations = 0
total_iter_vec = []

while true
    (minf, minx, ret) = optimize!(opt, η)
    numevals = opt.numevals # the number of function evaluations
    println("$p: $minf for $numevals iterations (returned $ret)")
    global total_iterations += numevals
    global total_iter_vec = push!(total_iter_vec, total_iterations)
    global p_vec = push!(p_vec, p)
    global f_av_vec = push!(f_av_vec, minf)
    global iter_vec = push!(iter_vec, numevals)
    global p += p_+
    err = norm(minf - f_0)
    global f_0 = minf
    ((err <= ε_0) || (p > p_max)) && break
end

η = reshape(η, m + 1, n + 1)

η_map = heatmap(
    0:Δx:xlen,
    0:Δy:ylen,
    η,
    yflip = true,
    xmirror = true,
    aspect_ratio = :equal,
    fontfamily = "serif",
    font = "Computer Modern Roman",
    colorbar_title = "η",
    title = "η for each Design Volume",
)
```