Summaries of Prerequisite Concepts

Optimization Methods

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. (Δx represents the *descent direction*, which is explained in the next section.) 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 only practical in situations where the cost of computing the minimization problem (in variable) is low compared to actually calculating the search direction.

Backtracking Line Search

Most often in practice we use **inexact line searches**. 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 $f(x+t\Delta x) \leq f(x) + \alpha t \nabla f(x)^T \Delta x$.

Figure 9.1 (taken from page 465 of [1]) demonstrates the Backtracking Line Search visually for a parabola.

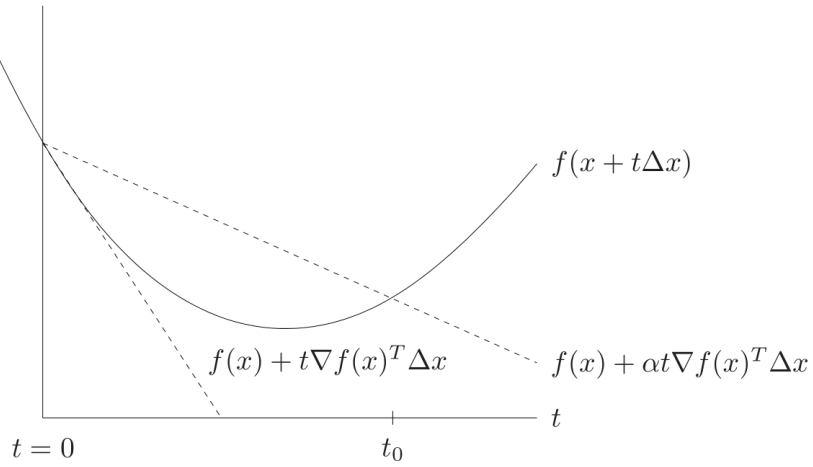

Figure 9.1 *Backtracking line search.* The curve shows f, restricted to the line over which we search. The lower dashed line shows the linear extrapolation of f, and the upper dashed line has a slope a factor of α smaller. The backtracking condition is that f lies below the upper dashed line, *i.e.*, $0 \leq t \leq t_0$.

Notice that the backtracking search will find a step size t such that $0 \leq t \leq t_0$ and that the minimum of the function is in this region. However, the step size we choose may not exactly be the minimum of the function, but we have funneled it down to be closer to the minimum of f.

Julia Code Here is an 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
```

Gradient Descent

Now we proceed to discuss descent algorithms, the first of which will be the **gradient descent method**.

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.

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.

Notice that essentially the algorithm uses a line search to determine a step size and then we update our iterate in the direction of the steepest descent. This is repeated until we meet some sort of stopping criterion, typically something of the form $\|\nabla f(x)\|_2 \leq \eta$, where η is small and positive.

Below is an implementation of the gradient descent algorithm in Julia for a function $f(x, y) = y^3 - y + x^2 - 3x$. This program uses the Backtracking Line Search implementation presented earlier to perform the line search. Notice that we need to input the function, the gradient of the function, and an initial guess.

```
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]

#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)
```

Nonlinear Conjugate Gradient Method

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, we start off by setting

$\Delta x_0 = -\nabla f(x_0)$.

We 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:

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

α_i is found by ensuring that the gradient is orthogonal to the search direction.

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: $x_{i+1} = x_i + \alpha_i d_i$.

The algorithm below uses the Newton-Raphson method to find the values of α_i.

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 $\varepsilon < 1$:

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

$i \Leftarrow 0$
$k \Leftarrow 0$
$r \Leftarrow -f'(x)$
$d \Leftarrow r$
$\delta_{\text{new}} \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta_{\text{new}}$
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**
    $\alpha \Leftarrow -\frac{[f'(x)]^T d}{d^T f''(x) d}$
    $x \Leftarrow x + \alpha d$
    $j \Leftarrow j + 1$
    $j < j_{\max}$ 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

PDE Methods

Finite Volume Method

> "The new idea of the finite volume method is to first integrate the equation over a small so-called control volume, then use the divergence theorem to convert the volume integral into a boundary integral involving fluxes, before finally approximating these fluxes across the boundaries." [3]

The Finite Volume Method takes advantage of the Divergence Theorem, so here are two different statements of the theorem:

Theorem (Divergence Theorem (Version 1) [2]). *Let E be a simple solid region and S the boundary surface of E with positive orientation. Let $\vec{F}$ be a vector field whose components have continuous first order partial derivatives. Then*

$$\iint_S \vec{F} \cdot \mathrm{d}\vec{S} = \iiint_E \operatorname{div} \vec{F} \, \mathrm{d}V.$$

Theorem (Divergence Theorem (Version 2) [6]). *Suppose that V is a subset of $\mathbb{R}^n$ which is compact and has a piecewise smooth boundary S (also indicated with $\partial V = S$). If $\mathbf{F}$ is a continuously differentiable vector field defined on a neighborhood of V. Then*

$$\iiint_V (\nabla \cdot \mathbf{F})\, dV = \oiint_S (\mathbf{F} \cdot \hat{n})\, dS$$

The left side is a volume integral over V and the right side is a surface integral over the boundary of V.

Finite Volumes for a General Two-Dimensional Diffusion Equation

For a scalar function $a(x)$ the general diffusion equation is

$$\nabla \cdot \big(a(\mathbf{x})\nabla u\big) = f \quad \text{in } \Omega.$$

"The finite volume method is designed to discretize such differential operators on arbitrary domains with very general grids." [3]

Given a grid of polygons with vertices $\{x_i\} \subset \Omega$, we create a set of **control volumes** around each x_i which allows the Finite Volume Method to discritize the PDE. One construction of control volumes is Voronoi cells.

Definition (Voronoi Cell [3]). *For a given set of vertices $\{x_i\} \subset \Omega$ of a mesh, we define the corresponding **Voronoi cells** V_i by*

$$V_i = \left\{x \in \Omega : \|x - x_i\| \leq \|x - x_j\|, \forall j \neq i, x_j \text{ neighbor of } x_i\right\}.$$

Then, using the divergence theorem to convert a volume integral into a surface integral, we integrate the PDE over each control volume V_i:

$$\int_{V_i} \nabla \cdot \big(a(\mathbf{x})\nabla u\big)\, d\mathbf{x} = \int_{\partial V_i} a(\mathbf{x})\nabla u \cdot \mathbf{n}\, ds = \int_{V_i} f\, d\mathbf{x}.$$

Then, using some approximation schemes, we obtain a sparce and structured linear system.

"One of the main advantages of the finite volume method is that it can naturally take boundary conditions into account on very general domains." [3]

The Optimization Problem

An **optimization problem** has the form

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

where

- $x = (x_1, \ldots, x_n)$ is the **optimization variable**

- $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the **objective function**
- $f_i : \mathbb{R}^n \to \mathbb{R}$ are the **inequality constraint functions**
- $h_i : \mathbb{R}^n \to \mathbb{R}$ are the **equality constraint functions**

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

A vector $x^\star$ is called **optimal** or a **solution** if it has the smallest objective value among all vectors that satisfy the constraints. That is, $x^\star$ is a solution to the optimization problem if for any z with $f_1(z) \leq 0, \ldots, f_m(z) \leq 0$, then $f_0(z) \geq f_0(x^\star)$.

A point x that is in the domain of all of the functions f_i and h_i is called **feasible** if it satisfies all the constraints. The **optimal value** $p^\star$ of the problem is defined as

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

The optimization problem is **linear** or called a **linear program** if the objective and constraint functions are all linear. That is, for $i = 1, \ldots, m$, $f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y)$ for all $x, y \in \mathbb{R}^n$ and all $\alpha, \beta \in \mathbb{R}$.

If the optimization problem is not linear, we call it a **nonlinear program**.

We can change a minimization problem into a **maximization problem** by minimizing $-f_0$ subject to the same constraints.

Convex Optimization

A set C is **convex** if the line segment between any two points in C lies in C. That is 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 $\mathbf{dom} f$ is a convex set and if for all $x, y \in \mathbf{dom} f$, and θ with $0 \leq \theta \leq 1$, we have

$$f\left(\theta x + (1 - \theta)\, y\right) \leq \theta f(x) + (1 - \theta) f(y). \tag{2}$$

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

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & a_i^T = b_i, \quad i = 1, \ldots, p \end{array} \tag{3}$$

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

There are three additional requirements that differentiate a convex optimization problem versus are regular 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.

In a convex optimization problem, any locally optimal point is also globally optimal.

The Linear Optimization Problem

When the objective and constraint functions are all affine, we have a **linear optimization problem** or **linear program**.

$$\begin{array}{ll} \text{minimize} & c^T x + d \\ \text{subject to} & Gx \preceq h, \\ & Ax = b, \end{array} \tag{4}$$

where $G \in \mathbb{R}^{m \times n}$ and $A \in \mathbb{R}^{p \times n}$. ($\preceq$ represents componentwise inequality.)

Linear optimization problems are convex optimization problems.

The Quadratic Optimization Problem

If the objective function is a convex quadratic and the constraint functions are affine in (3), then we have a **quadratic program**.

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x + r \\ \text{subject to} & Gx \preceq h \\ & Ax = b, \end{array} \tag{5}$$

where P is a symmetric positive semidefinite $n \times n$ matrix, $G \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{p \times n}$, $q \in \mathbb{R}^n$, and $r \in \mathbb{R}$.

Example: Least-Squares

In the **least squares problem** we want to minimize $\|Ax - b\|_2$. Notice

$$\frac{1}{2}\|Ax - b\|_2^2 = \frac{1}{2}x^T A^T A x - b^T A x + \frac{1}{2}b^T b$$

is actually a convex quadratic function. Since $\|Ax - b\|_2 \geq 0$, we know that the minimal solution for $\frac{1}{2}\|Ax - b\|_2^2$ will also be a minimal solution for $\|Ax - b\|_2$. Therefore, if we are able to minimize the corresponding quadratic function, we will have accomplished our goal.

Looking back to (5), we have $P = A^T A$, $q = b^T A$, and $r = \frac{1}{2}b^T b$, so this is really a quadratic optimization problem.

The Method of Moving Asymptotes (MMA)

> "Ideally, a method for structural optimization should be flexible and general. It should be able to handle not only element sizes as design variables, but also, for instance, shape variables and material orientation angles."[5]

General Method Description

Consider a structural optimization problem of the following general form

$$
\begin{array}{lllll}
P\text{:} & \text{minimize} & f_0(\mathbf{x}) & & (\mathbf{x} \in \mathbb{R}^n) \\
 & \text{subject to} & f_i(\mathbf{x}) \leq \hat{f}_i, & \text{for } i = 1, \ldots, m & \\
 & & \underline{x}_j \leq x_j \leq \overline{x}_j, & \text{for } j = 1, \ldots, n &
\end{array} \tag{6}
$$

where

- $\mathbf{x} = (x_1, \ldots, x_n)^T$ is the vector of design variables
- $f_0(\mathbf{x})$ is the objective function (typically structural weight)
- $f_i(\mathbf{x}) \leq \hat{f}_i$ are behavior constraints (typically limitations on stresses and displacements)
- $\underline{x}_j$ and $\overline{x}_j$ are given lower and upper bounds ("technological constraints") on the design 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 iteration point $\mathbf{x}^{(k)}$, 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 (6), the (usually implicit) functions f_i by approximating explicit 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

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

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

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

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 becomes the approximation of the original problem. 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 $P^{(k)}$:

$$\begin{aligned} P^{(k)}: \quad \text{minimize} \quad & \sum_{j=1}^{n} \left(\frac{p_{oj}^{(k)}}{U_j^{(k)} - x_j} + \frac{q_{oj}^{(k)}}{x_j - L_j^{(k)}} \right) + r_o^{(k)} \\ \text{subject to} \quad & \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{U_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - L_j^{(k)}} \right) + r_i^{(k)} \leq \hat{f}_i, \quad \text{for } i = 1, \ldots, m \\ \text{and} \quad & \max\{\underline{x}_j, \alpha_j^{(k)}\} \leq x_j \leq \min\{\overline{x}_j, \beta_j^{(k)}\}, \quad \text{for } j = 1, \ldots, n \end{aligned} \tag{7}$$

(The parameters $\alpha_j^{(k)}$ and $\beta_j^{(k)}$ are called "move limits" and are not that important.)

$\alpha_j^{(k)}$ 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.

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:

$$\begin{array}{lll} D\text{:} & \text{maximize} & W(\mathbf{y}) \\ & \text{subject to} & \mathbf{y} \geq 0 \end{array} \tag{8}$$

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

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

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

The SIMP Method

References

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

[2] P. Dawkins, *Calculus III - Divergence Theorem*. https://tutorial.math.lamar.edu/classes/calciii/DivergenceTheorem.aspx, Apr 2018. [Online; accessed 15-March-2021].

[3] M. J. Gander and F. Kwok, *Numerical Analysis of Partial Differential Equations Using Maple and MATLAB*, Society for Industrial and Applied Mathematics, Aug 2018.

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

[5] 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.

[6] Wikipedia contributors, *Divergence theorem — Wikipedia, the free encyclopedia*. https://en.wikipedia.org/w/index.php?title=Divergence_theorem&oldid=1011407395, 2021. [Online; accessed 15-March-2021].