Generating Space-Filling Designs Using Meshes

Greta Linse
Department of Mathematical Sciences
Montana State University

April 30, 2008

A writing project submitted in partial fulfillment
of the requirements for the degree

Master of Science in Statistics

1 Introduction

In many scientific fields experimental mixtures are created and tested for usefulness. Often the ingredients for a compound are known but the ideal proportions of the ingredients are not. In some cases obtaining a mixture close to this ideal proportion is good enough. For example, finding the perfect tropical punch recipe does not necessitate having precise proportions of each ingredient. If one particular mixture is tried and found to be unsatisfactory, perhaps more of one ingredient can be added to perfect the taste. In most scientific and engineering applications cost and time are important factors and often you cannot just add a dash more of one ingredient to perfect the mixture. Consider an aluminum alloy made from aluminum, copper, manganese, and magnesium. The process required to make the alloy is such that the casting cannot be later adjusted by adding a little bit more of one ingredient to make it stronger. Instead, the proportions of each ingredient need to be decided upon in advance and then the alloy is made. Many combinations of the components may be necessary in order to determine the mixture which best satisfies the requirements.

Due to the real constraints of time and money every possible combination of ingredients cannot be tested. As a result only a finite number of mixtures can be tested. Here a mixture is defined by the specified proportions for the set of ingredients. Thus the question becomes, "How can the experimental proportions be chosen, so that the best mixture can be found?" Generation of a simple random sample is not the best approach in this situation because it could choose proportions for each component that are too close to each other and not be a good use of resources. Instead, one approach is to find test proportions that are as equally distributed in the experimental region as possible. Doing so would eliminate the over-testing of any given region in the design space and would more efficiently use the available resources. Thus, it remains to find these equally distributed mixture proportions.

Consider the following example. Currently there are two major gasoline/ethanol blends on the market in the United States E10, and E85, where E10 contains 10% ethanol and 90% gasoline, and similarly E85 contains 85% ethanol and 15% gasoline. Are these the best mixtures possible to meet or exceed governmental and environmental criteria? Perhaps a 50-50 blend would be a better blend. Or perhaps splitting the category "ethanol" into interesting:

$$0 \leq a_i \leq x_i \leq b_i \leq 1 \text{ for } i = 1, 2, \ldots, n$$

where a_i and b_i are the lower and upper bounds for x_i, the i^{th} component proportion.

Three: There can be restrictions on combinations of components, and these are called Multiple Component Constraints. These constraints require that some linear combination of the n components satisfy proportion constraints.

$$D_k \leq \sum_{i=1}^{n} C_i x_i \leq E_k \text{ where } D_k,\ C_i, \text{ and } E_k \text{ are all constants.}$$

Multiple component constraints can either be written with both upper and lower bounds or by two inequalities having only an upper bound. For example, the inequality $a \leq x \leq b$ can be rewritten $-x \leq -a$ and $x \leq b$. As a result of splitting the two-sided multiple component constraint inequalities into two one-sided constraint inequalities there will be twice as many inequalities.

$$\sum_{i=1}^{n} C_i x_i \leq E_{jk} \text{ for } j = 1, 2 \tag{1}$$

The single component constraint is a special case of the multiple component constraint. It is better to think of all constraint inequalities as multiple component constraints that may have only one component in the inequality. A matrix inequality is an alternative way to write linear inequalities, and is computationally convenient especially for implementation in Matlab. In matrix form the constraint inequalities in (1) are

$$\mathbf{A}\mathbf{p} \leq \mathbf{b} \tag{2}$$

where the matrix $\mathbf{A}$ contains the constraint coefficients, $\mathbf{p}$ is the vector of proportions $\begin{pmatrix} x_1, & x_2, & \ldots, & x_n \end{pmatrix}'$ representing a point which is in the experimental region, and $\mathbf{b}$ is the vector of constraints.

In the fuel mixture example, let $E_C = x_1$, $E_S = x_2$, and $G = x_3$ for notational convenience. The specified proportions must satisfy the three restrictions, or ways to write the constraint inequalities. The three components must sum to one, each component has an upper and lower bound contained in the interval $[0, 1]$ and there are restrictions that apply to combinations of mixture components.

$$0.10 \le \sum_{i=1}^{3} C_i x_i \le 0.85 \qquad \text{where } C_1 = 1, \quad C_2 = 0, \quad \text{and } C_3 = 0$$

$$0 \le \sum_{i=1}^{3} C_i x_i \le 0.25 \qquad \text{where } C_1 = 0, \quad C_2 = 1, \quad \text{and } C_3 = 0$$

$$0.15 \le \sum_{i=1}^{3} C_i x_i \le 0.9 \qquad \text{where } C_1 = 0, \quad C_2 = 0, \quad \text{and } C_3 = 1$$

$$0.10 \le \sum_{i=1}^{3} C_i x_i \le 0.85 \qquad \text{where } C_1 = 1, \quad C_2 = 1, \quad \text{and } C_3 = 0$$

$$0 \le \sum_{i=1}^{3} C_i x_i \le 0.85 \qquad \text{where } C_1 = 1, \quad C_2 = -1, \text{ and } C_3 = 0$$

$$1 \le \sum_{i=1}^{3} C_i x_i \le 1 \qquad \text{where } C_1 = 1, \quad C_2 = 1, \quad \text{and } C_3 = 1$$

The problem at hand is to transform the constrained region in n-dimensions to a region orthogonal to the n^{th} component axis. Then, using Matlab code, find experimental test mixtures in the $(n-1)$-dimensional transformed constrained region that are as equally spread out as possible in order to find the best mixture with little redundancy and without neglecting an area of the experimental region. These test points will be found using a triangular mesh generating code written by Per-Olof Persson and interfacing code written by the author. There are other approaches to this problem, including number theoretic approaches (Borkowski, 2006), but this proposed approach is unique in the use of existing Matlab code as well as new Matlab code written specifically for this problem. Once the $(n-1)$-dimensional test points are found in the transformed region, it is a simple matter to transform them back to the n-dimensional constrained region and implement them in an experiment as desired.

2 A Mesh Generating Approach

In a mixture experiment a necessary constraint is that the proportion of all of the components sum to one. The fuel example has three components (corn ethanol, switchgrass ethanol, and gasoline), however, there are only two degrees of freedom. Given proportions of two components the third is determined by subtraction of the known proportions from one. In other words, the constraint that $\sum_{i=1}^{n} x_i = 1$ is equivalent to the equation $x_n = 1 - \sum_{i=1}^{n-1} x_i$. Thus, the design region is contained in an $(n-1)$-dimensional simplex which is embedded in n-dimensional space, and can be projected to $(n-1)$-dimensional space without losing any information. The transformation process is taken from chapter 3 of (Cornell, 2002).

2.1 2-dimensional (2-d) Motivating Example

For two components, the 2-d simplex is a line segment between $\mathbf{x}_1 = (1, 0)$ and $\mathbf{x}_2 = (0, 1)$. The center of mass for this simplex is at $\mathbf{c} = (\frac{1}{2}, \frac{1}{2})$, and it will be the new origin $\tilde{\mathbf{c}}$ in the transformed simplex or line segment in the 2 dimensional case. To get $\tilde{\mathbf{c}} = (0, 0)$, subtract $(\frac{1}{2}, \frac{1}{2})$ from $\mathbf{c}$. If this subtraction is performed on the vertices of the 2-dimensional simplex, the resulting vertices are $\mathbf{x}_1^* = (\frac{1}{2}, -\frac{1}{2})$ and $\mathbf{x}_2^* = (-\frac{1}{2}, \frac{1}{2})$. For convenience, however, the coordinates are scaled so that they are either ± 1 to avoid fractions at this point. Thus, define $\tilde{\mathbf{x}}_1 = (1, -1)$, and $\tilde{\mathbf{x}}_2 = (-1, 1)$. In general, to shift x (the centroid or center of mass of the simplex) to the origin the following transformation is used:

$$\tilde{\mathbf{x}}_i = n\left(\mathbf{x}_i - \frac{1}{n}\mathbf{j}\right) = n\mathbf{x}_i - \mathbf{j}$$

where $\mathbf{j}$ is a $n \times 1$ vector of ones.

The next step is to perform a rotation of the axes which will make the axis of the n^{th} component of the mixture orthogonal to the simplex. In the 2-d case, this means rotating the axes 45° clockwise. This makes the $\mathbf{x}_2$ axis perpendicular to the simplex or line segment between $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$. It is easy to visualize in the 2-d case, but is not so simple in higher dimensions. A rotation matrix that results in a rotation of the axes 45° clockwise is given by:

$$\tilde{\boldsymbol{\Theta}} = \begin{bmatrix} \theta_1 & \theta_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

Because the shifted simplex is along the vector θ_1 and $\theta_1 \bullet \theta_2 = 0$, then $\theta_1 \perp \theta_2$. Thus, the axis of the 2^{nd} component is orthogonal to the simplex. By multiplying $\tilde{\mathbf{\Theta}}$ by $\frac{1}{\sqrt{2}}$ the resulting matrix $\mathbf{\Theta}$ is normalized and is called an orthonormal matrix:

$$\mathbf{\Theta} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

Next, let $\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. Note that $\tilde{\mathbf{X}}$ is symmetric so $\tilde{\mathbf{X}}' = \tilde{\mathbf{X}}$. The expression $\tilde{\mathbf{X}}'\mathbf{\Theta}$ is the projection of the two dimensional simplex onto one-dimension.

$$\tilde{\mathbf{X}}'\mathbf{\Theta} = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & 0 \\ -2 & 0 \end{bmatrix} = \sqrt{2} \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$$

For convenience, define the matrix $\mathbf{W}'$ to be $\mathbf{W}' = (n-1)\tilde{\mathbf{X}}'\mathbf{\Theta}$, which means that $\mathbf{W}' = \tilde{\mathbf{X}}'\mathbf{\Theta}$ when $n = 2$. Note that the rank of the matrix $\mathbf{W}$ equals one because there is one linearly independent column in $\mathbf{W}'$. The columns of $\mathbf{W}$ are the transformed vertices of the simplex projected from two dimensions onto only the x-axis.

2.2 In General: n dimensions

In general, any point from a shifted n-dimensional simplex where the centroid of the simplex is the origin $(0, 0, 0, \ldots, 0)'$ can be given by the following transformation:

$$\tilde{\mathbf{x}}_i = n(\mathbf{x}_i - \frac{1}{n}\mathbf{j}) = n\mathbf{x}_i - \mathbf{j} \text{ for } i = 1, 2, \ldots, n.$$

If all of these points are written as the columns of a matrix $\tilde{\mathbf{X}}$ then

$$\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \ldots & \tilde{\mathbf{x}}_n \end{bmatrix}.$$

This means that the matrix $\tilde{\mathbf{X}}$ can be expressed in terms of the matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n \end{bmatrix}$, the $n \times N$ matrix whose columns are the N original points from the simplex, and $\mathbf{J}$ an $n \times N$

matrix of ones:

$$\tilde{\mathbf{X}} = n\left(\mathbf{X} - \frac{1}{n}\mathbf{J}\right)$$

The $n \times n$ rotation matrix Θ is defined for n-dimensions as follows:

$$\Theta = \frac{1}{\sqrt{n(n-1)}} \begin{bmatrix} n-1 & 0 & 0 & \cdots & 0 & s \\ -1 & (n-2)l & 0 & \cdots & 0 & s \\ -1 & -l & (n-3)m & & 0 & s \\ -1 & -l & -m & & 0 & s \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -l & -m & & t & s \\ -1 & -l & -m & & -t & s \end{bmatrix}$$

where $l, m, \ldots, t, s$ are positive constants defined to make the sum of the squares of the entries in each column equal to $n(n-1)$ (Cornell, 2002). For example, take the sum of the squares of the second column: $(n-2)^2l^2 + (n-2)l^2 = n(n-1)$ This equation can be solved for l^2, and results in $l^2 = \frac{n}{n-2}$. If $n = 3$, then $l = \sqrt{3}$.

The following equation results in a transformation making the n^{th} axis orthogonal to the simplex in terms of the rotation and translation matrices:

$$\mathbf{W} = (n-1)\Theta'\tilde{\mathbf{X}}.$$

The matrix $\mathbf{W}$ can also be written in terms of the original points in the simplex by replacing the translation matrix by its definition:

$$\mathbf{W} = (n-1)\Theta'(n\mathbf{X} - \mathbf{J}).$$

Also note that the transposed transformation matrix $\mathbf{W}'$ is given by:

$$\mathbf{W}' = (n-1)\left(n\mathbf{X}' - \mathbf{J}'\right)\Theta.$$

The columns of W are the transformed vertices, and the n^{th} row of W is entirely zeros.

The vertices in $(n-1)$-dimensions are the columns of $W[1:n-1,:]$, the matrix W with the last row removed.

For any matrix $\mathbf{X}$ of points within the simplex or constrained region, if $\mathbf{x}_i$ is the i^{th} test point, or column in the $\mathbf{X}$ matrix, then $\mathbf{w}_i$ is the i^{th} transformed test point, or column in the $\mathbf{W}$ matrix, where $\mathbf{w}_i = (n-1)\Theta'\tilde{\mathbf{x}}_\mathbf{i}$. From (2), $\mathbf{A}$ is an $2k \times n$ matrix of constraint coefficients, and $\mathbf{b}$ is a $2k \times 1$ vector of constraints. It is necessary to find the transformed matrix $\tilde{\mathbf{A}}$ and the transformed vector $\tilde{\mathbf{b}}$ that satisfies the equation $\tilde{\mathbf{A}}\mathbf{w}_i \leq \tilde{\mathbf{b}}$.

The equation $\mathbf{w}_i = (n-1)\Theta'(n\mathbf{x}_i - \mathbf{j})$, can be written in terms of a particular point in the n-d constrained region, $\mathbf{x}_i = \frac{1}{n}\left(\frac{1}{n-1}\Theta\mathbf{w}_i + \mathbf{j}\right)$. Then $\mathbf{p}$ in (2) can be replaced by $\mathbf{x}_i$

$$\mathbf{A}\mathbf{x}_i \leq \mathbf{b} \tag{3}$$

which is equivalent to

$$\mathbf{A}\left[\frac{1}{n}\left(\frac{1}{n-1}\Theta\mathbf{w}_i + \mathbf{j}\right)\right] \leq \mathbf{b}.$$

By distributing the matrix $\mathbf{A}$ of constraint coefficients and by factoring out $\frac{1}{n}$, this inequality is equivalent to

$$\frac{1}{n}\left(\frac{1}{n-1}\mathbf{A}\Theta\mathbf{w}_i + \mathbf{A}\mathbf{j}\right) \leq \mathbf{b}.$$

The goal is to obtain a linear inequality with those terms dependent on $\mathbf{w}_i$ less than or equal to an expression containing all other terms not dependent on $\mathbf{w}_i$. Thus, the above inequality is rewritten by multiplying both sides by n and subtracting $\mathbf{A}\mathbf{j}$ from both sides because it is free of $\mathbf{w}_i$:

$$\frac{1}{n-1}\mathbf{A}\Theta\mathbf{w}_i \leq n\mathbf{b} - \mathbf{A}\mathbf{j}.$$

Finally, by multiplying both sides by $n-1$ the following inequality is obtained.

$$\mathbf{A}\Theta\mathbf{w}_i \leq (n-1)\left[n\mathbf{b} - \mathbf{A}\mathbf{j}\right]$$

Now, define the transformed constraint coefficient matrix $\tilde{\mathbf{A}} = \mathbf{A}\Theta$ which are the constraint coefficients of the transformed point $\mathbf{w}_i$. Also define the transformed constraint vector $\tilde{\mathbf{b}} = (n-1)\left[n\mathbf{b} - \mathbf{A}\mathbf{j}\right]$. Then (2) can be expressed as a linear inequality in terms of transformed points in the $(n-1)$-dimensional projected constraint region.

For mixture designs, the region defined by the constraints is a subset of the n-dimensional simplex. The vertices of the n-d simplex are fixed, with the location of each vertex corresponding to a state where the entire mixture is made up of one component. For example, if there are three components the vertices are $\begin{pmatrix}1 & 0 & 0\end{pmatrix}'$, $\begin{pmatrix}0 & 1 & 0\end{pmatrix}'$, and $\begin{pmatrix}0 & 0 & 1\end{pmatrix}'$. These vertices are also called elementary column vectors. The vertices of the n-dimensional simplex will never change, no matter what the constrained region is as the transformation is a linear one-to-one function. If a transformation is defined using the vertices (the elementary column vectors) then it will hold for any point in the simplex, including any point in the constrained region. Thus, the transformation equation defined as above can be coded in Matlab using only the elementary vectors which are dependent only on the number of components in the mixture. Therefore, the only input for the Matlab code transform.m is n, the number of components. (See Appendix B for transform.m).

2.3 Fuel Example

Although the fuel example involves three components, if a design point were to consist of 35% ethanol from corn and 55% gasoline, it is then known that 10% of ethanol from switchgrass is in the fuel mixture. Thus, this three component mixture can be rewritten as one with only two components. The first step is to shift and stretch the simplex so that the origin is at the center of mass of the simplex with the vertices still having integer components. For this example, there are three components, and $\mathbf{x}_1 = \begin{pmatrix}1 & 0 & 0\end{pmatrix}'$; $\mathbf{x}_2 = \begin{pmatrix}0 & 1 & 0\end{pmatrix}'$; $\mathbf{x}_3 = \begin{pmatrix}0 & 0 & 1\end{pmatrix}'$, the shifted and stretched vertices are $\tilde{\mathbf{x}}_1 = \begin{pmatrix}2 & -1 & -1\end{pmatrix}'$; $\tilde{\mathbf{x}}_2 = \begin{pmatrix}-1 & 2 & -1\end{pmatrix}'$; $\tilde{\mathbf{x}}_3 = \begin{pmatrix}-1 & -1 & 2\end{pmatrix}'$, and the orthonormal rotation matrix is as follows:

$$\Theta = \frac{1}{\sqrt{3(3-1)}}\begin{bmatrix}2 & 0 & \sqrt{2}\\ -1 & \sqrt{3} & \sqrt{2}\\ -1 & -\sqrt{3} & \sqrt{2}\end{bmatrix}. \tag{4}$$

The transformed vertices in 2-dimensions are:
$\mathbf{w}_1 = \begin{pmatrix}\frac{12}{\sqrt{6}} & 0\end{pmatrix}'$; $\mathbf{w}_2 = \begin{pmatrix}-\frac{6}{\sqrt{6}} & \frac{6}{\sqrt{2}}\end{pmatrix}'$; $\mathbf{w}_3 = \begin{pmatrix}-\frac{6}{\sqrt{6}} & -\frac{6}{\sqrt{2}}\end{pmatrix}'$,

For the specific fuel example there are the following constraint inequalities:

Two-sided Inequalities	With Corresponding One-sided Inequalities
$0.10 \leq x_1 \leq 0.85$	$-x_1 \leq -0.10$ and $x_1 \leq 0.85$
$0 \leq x_2 \leq 0.25$	$-x_2 \leq 0$ and $x_2 \leq 0.25$
$0.15 \leq x_3 \leq 0.9$	$-x_3 \leq -0.15$ and $x_3 \leq 0.9$
$0.10 \leq x_1 + x_2 \leq 0.85$	$-x_1 - x_2 \leq -0.10$ and $x_1 + x_2 \leq 0.85$
$x_2 \leq x_1$	$x_2 - x_1 \leq 0$ and $x_1 - x_2 \leq 0.85$

which can be written as (2) in terms of the constraint coefficient matrix $\mathbf{A}$ and the constraint vector $\mathbf{b}$.

$$\mathbf{A} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ -1 & -1 & 0 \\ 1 & 1 & 0 \\ -1 & 1 & 0 \\ 1 & -1 & 0 \end{bmatrix} \quad \text{and } \mathbf{b} = \begin{bmatrix} -.10 \\ 0.85 \\ 0 \\ 0.25 \\ -0.15 \\ 0.9 \\ -0.10 \\ 0.85 \\ 0 \\ 0.85 \end{bmatrix}$$

For this example the transformed constraint coefficient matrix $\tilde{\mathbf{A}} = \mathbf{A}\Theta$, and the transformed constraint vector $\tilde{\mathbf{b}} = (n-1)\,[n\mathbf{b} - \mathbf{A}\mathbf{j}]$ are given below:

$$\tilde{\mathbf{A}} = \begin{bmatrix} -\frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & -\frac{2}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{2}{\sqrt{3}} \\ -\frac{3}{\sqrt{6}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{3}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix} \text{ and } \tilde{\mathbf{b}} = \begin{bmatrix} 1.4 \\ 3.1 \\ 2.0 \\ -0.5 \\ 1.1 \\ 3.4 \\ 3.4 \\ 1.1 \\ 0 \\ 5.1 \end{bmatrix}.$$

Now that the constraint equations have been transformed to an $(n-1)$-dimensional region in n-d space,the test points can be found in the transformed experimental region. Once the test points have been found, the transformation that takes every test point in the grid back into the n-dimensional design region is given by:

$$\mathbf{P}' = \frac{1}{n}\left(\frac{1}{n-1}\Theta\tilde{\mathbf{P}}' + \mathbf{J}\right)$$

The Matlab code findpoints.m (see Appendix A) was written to take advantage of "N-D Mesh Generator using Distance Functions" (see Appendix D) written by Persson (2004) and uses a truss and force-equilibrium, or static approach to achieve nearly equilateral triangles in 2-d and the higher dimension analog for higher dimensional spaces. The triangulation is performed by the Delaunay Triangulation Algorithm (delaunay.m) (Persson and Strang, 2004) in Matlab code.

2.4 Delaunay Triangulation

When given points in a region, Delaunay triangulation connects points to its nearest neighbors in order to form a triangle (in two dimensions) or a simplex (in higher dimensions). The condition on how to form the triangle or simplex is that no other point in the region lies within the circumcircle of the triangle in two dimensions or within the circum-hypersphere of the simplex in higher dimensions. The result is the maximization of the minimum angle for each triangle, or equivalently the maximization of the minimum edge length for each triangle in two dimensions. The n-dimensional analog of the result is that the minimum edge length of each simplex is maximized. Thus the Delaunay triangulation provides an easy way to compute an upper bound for the largest minimum distance between any two points in the region by finding the maximum edge length over the whole region. Distance and length is assumed to be the standard Euclidean metric. There are generalizations to other metrics, but the Euclidean metric is the easiest to work with. The Delaunay triangulation is guaranteed to be unique as long as a circle cannot be formed by for any four points (in 2-dimensions) in the set of points to be triangulated (de Berg et al., 2000; Wik, 2008).

The Delaunay triangularization method in Matlab finds the convex hull of a set of points in $(n-1)$-dimensions and then maps these points back into n-dimensions. The convex hull is defined to be the intersection of all convex sets containing all points in the region. If it is assumed that each side of the convex hull is a simplex, then the convex hull is unique. This guarantees that the triangularization in n-dimensions will be unique (Linse, 2006; Wik, 2006).

2.5 Re-distribution of Points

In the initial point dispersion, see Figure 1, any point that is outside of the region of interest is discarded and the points that remain within the region are re-distributed. The coordinates of the vertices for the design region are given by $\mathbf{w}_i$ above. The design region for Figure 1 is the entire 3 component simplex which has been projected into 2-dimensional space. Given m points which are the vertices of the triangulation, the points can be redistributed by force-equilibrium equations so that the internal and external forces acting at each point along the edges of each triangle are in equilibrium. For a point to be in equilibrium means that there is no spring force acting along a truss (edge) pulling it towards another point or points. Spring forces are always positive which prevents triangles from contracting and ending up with edge lengths shorter than the average.When edge lengths are shorter than average, they are required to be stretched again until they (optimally) reach an equilibrium location. This will effectively ensure that the edges of each triangle (in 2-dimensions)

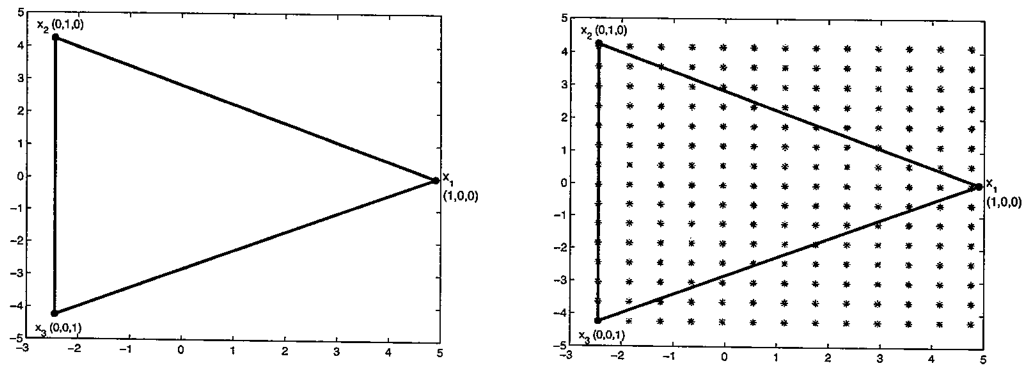

Figure 1: Initial Point Dispersion of 3 Component Simplex

or simplex (in higher dimensions) will be nearly equal. The external reaction forces at the boundaries keep the points from moving out of the region. If the algorithm sends a point out of the region, then these reaction forces move the point back to the boundary along a normal force vector. The "Simple Mesh Generator" code approximately solves the force-equilibrium system by imposing an artificial time-dependence and finds a stationary solution to a system of ordinary differential equations by the forward Euler method (Linse, 2006). Figure 2 shows the initial triangularization for the 3 component simplex. The initial triangularization is not too bad because the initial dispersion of the points are on a grid. However, the triangularization is not very good along the edges of the simplex that are not oriented along the grid. Each initial triangle is also not an equilateral triangle. In the final triangularization nearly every triangle appears to be equilateral or near equilateral and the behavior on the boundaries is much better. In this example, it took only 15 re-triangularizations to reach the final solution, and there are 100 nodes or test points. By observation there are 39 boundary or near boundary points and 61 interior points.

2.6 Finding Design Test Points

Input for the Matlab program findpoints.m requires the matrix $\mathbf{A}$ of constraint coefficients, the vector $\mathbf{b}$ of constraints, and h_0, the initial distance (or edge length) between points in the original box bounding the constrained region. For triangulation, the initial edge length, h_0, needs to be chosen with two things in mind. First, decreasing h_0 will increase the computational time to achieve a mesh solution and will produce a large number of test

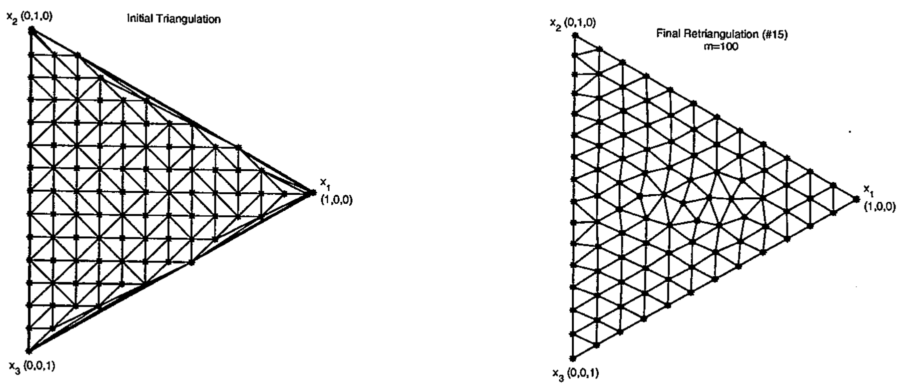

Figure 2: Initial Triangulation and Final Triangulation

points which may not be desirable. Test points are the nodes of the triangles determined by the mesh generator and are called "test" points because they will be used in an experimental situation to test for the best mixture proportions. Second, the larger h_0 is the harder it will be for the distmeshnd.m code to determine an equilibrium solution. These problems are compounded in higher dimensions. The initial test points are laid out on a square grid within the bounding box (Figure 1). While in two dimensions it would be advantageous to distribute points in the bounding box based on a equilateral triangle grid, this approach can not be generalized to higher dimesions. In two dimensions, the space can be tiled with equilateral triangles. The three dimensional analog of an equilateral triangle is a pyramid with four equilateral faces. A three dimensional region cannot be tiled with these pyramids alone, irregular shapes would be needed to fill in the gaps left by the pyramids. If only three component mixtures are to be studied then in the Matlab code findpoints.m, distmeshnd.m can be replaced with distmesh2d.m.

The output from the Matlab code findpoints.m is m, the number of nearly equally dispersed points in the experimental n-dimensional region and $\mathbf{P}$ an $m \times n$ matrix containing the location of the points in the n-dimensional experimental region.

For the fuel example, the following table gives the number of test points in the region for various initial edge lengths, and the computational time. When an initial edge length of 1.2 was tried, Matlab tried to triangulate 10 test points but after 15 re-triangulations the test points were on or near the boundary and Matlab crashed. See figures 4,5 and 6 for the final triangularization of the fuel example for the initial edge lengths of $h_0 = 0.3$, $h_0 = 0.5$,

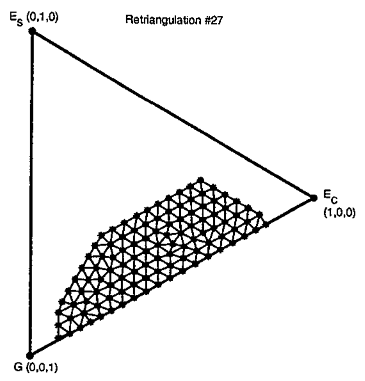

Figure 3: $h_0 = 0.3$ and $m = 108$

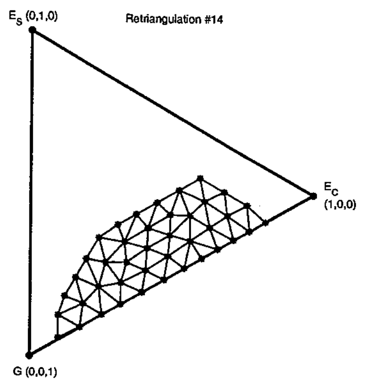

Figure 4: $h_0 = 0.5$ and $m = 41$

and $h_0 = 0.9$ respectively.

h_0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
m	920	237	108	65	41	29	23	19	16	12	11	10
t in sec.	119.8	20.8	13	5.5	3.5	4.9	6.1	3.1	2.2	1.42	7.1	∞?
#of Re-triangulations	35	26	27	18	14	24	25	16	15	10	25	15+

3 Conclusion

This new triangular meshing approach is very effective and relatively quick at determining test mixture proportions especially for three components. The code involved is not very

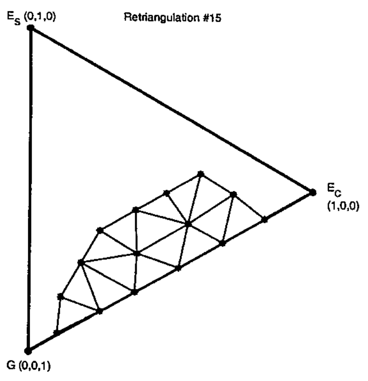

Figure 5: $h_0 = 0.9$ and $m = 16$

complex and can be understood by a wide audience. It is beneficial to have accessible and transparent code in order to allow more disciplines or applications to transform the code and use it without having to write new code from scratch.

Mixture designs have always been important to industry and scientific research. However, there has not been an efficient and easy method to choose design points based specifically on the design region to be studied. Other methods test predetermined proportions (Cornell, 2002) not because they offer a good coverage of the design region, but because they are present in every design region. Another approach, the number-theoretic methods (Borkowski, 2006) are good because you can specify the exact number of test points needed for the mixture design. This approach is as easily automated as the triangular meshing approach which uses Matlab code, and the best coverage of the design region is not guaranteed.

The mesh generating approach presented above does have some disadvantages and there is much room for further work. The main disadvantage is that the final number of test points cannot be specified in advance. In fact after running the code several times for several different edge lengths it is possible to not ever achieve the exact number of desired test points but to only get close. For example, in the fuel mixture problem above, consider the case where the researcher only has funds and resources for 50 tests. The initial edge lengths 0.4 and 0.5 give the final number of test points as 65 and 41 respectively. The researcher must then run the code for different edge lengths between 0.4 and 0.5 to achieve a number of test points as close to 50 as possible. If it is not possible to get exactly 50 then the researcher must be willing to accept a smaller number of tests or try to find resources for a few more tests. The other disadvantage to the current triangular mesh approach is that for higher dimensions it is harder for Matlab to determine if a point is inside the design region or not. Both the forcing of points back to the boundary and the termination criterion rely on the maximum distances being less than a certain amount. This maximum allowance is determined by the specified edge length h, which needs to be larger for higher dimensions in order for the solution to converge. In order to fix this a larger tolerance for being considered a boundary or exterior point was used, but this did not eliminate the problem.

The triangular mesh generating approach could be improved by speeding up the computation time for mixtures with 6 components or more. One way to decrease the computation time would be to write the distance formula (distance.m) in MEX code, which is essentially C++ code compiled in Matlab. The distance algorithm has to be run for every test point for every re-triangulation. Although this algorithm is written using matrices, the quickest and preferred method for Matlab computation, Matlab is still inherently slower at complex computations than C++. For small numbers of components the difference in computation times would be negligible, but for large numbers of components the difference could be quite significant. Other programs besides Matlab were considered, but Matlab is still the best at both computation and graphical output. The code used for this approach distmeshnd.m also has a variant for Cran R, but R is significantly slower than Matlab for these complex computations.

While the triangular mesh approach does give the best coverage of the design region it still places test points on or near the boundaries. Some applications are such that boundary solutions are not useful or desirable. One way to prevent boundary solutions would be to change the matrix inequality into a strict inequality. $\mathbf{Ax} \leq \mathbf{b}$ becomes $\mathbf{Ax} < \mathbf{b}$. This may not be good enough at keeping points from the boundary due to round-off error. Instead an artificial boundary could be chosen within the desired design region. The artificial design region would then be used for calculating the test points. This is a very simple approach using the triangular mesh code, however, defining the artificial boundary would have to deal with lower and upper constraints of the components separately. Instead of using the matrix inequality $\mathbf{Ax} \leq \mathbf{b}$, the inequality $\mathbf{A_1x} \leq p\mathbf{b_1}$ could be used for upper constraints where p is a percent. If $p = 0.95$ then the artificial design region would be 95% of the actual desired design region. For lower constraints, the inequality $\mathbf{A_2x} \leq -p\mathbf{b_2}$ could be used, where p is the same percent for the upper constraints. In the code where $\mathbf{b}$ was required for input, the user would have to separately input matrices for upper and lower constraints and then $p\mathbf{b_1}$ or $-p\mathbf{b_2}$ would be used as appropriate. This will keep all design test points in the inner p percent of the design region and away from the actual design region boundary. The applications and refinements are numerous, this is one of the reasons that makes the triangular mesh generating approach so good. It is hoped that this work will be used and expanded to the benefit of many industries and researchers.

References

Convex hull, December 2006. URL `http://en.wikipedia.org/wiki/Convex_hull`.

Delaunay triangulation, January 2008. URL `http://en.wikipedia.org/wiki/Delaunay_triangulation\#_note--Delaunay1934`.

John Borkowski. Space-filling designs for high-dimensional mixture experiments with multiple component constraints. In *Proceedings of the 3rd Sino-International Symposium on Probability, Statistics, and Quantitative Measurement*, pages 19–29, June 2006.

J. Cornell. *Experiments with Mixtures: Designs, Models, and the Analysis of Mixture Data.* John Wiley & Sons, 2002.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, 2nd, revised edition, 2000.

Greta Linse. Unpublished final paper. Math 581: Numerical Solution of Partial Differential Equations I, December 2006.

Per-Olof Persson and Gilbert Strang. A simple mesh generator in MATLAB. *SIAM Review*, 46(2):329–345, June 2004.

A findpoints.m

```
function [m,P] = findpoints(A,b,h0)
%  findpoints finds nearly uniformly dispersed points in a region to use
%  as optimal test locations. Where Az <= b, for z an n-d point in the
%  region.
% Input:
%  A : a matrix of constraint coefficients, for either Single Component
%      Constraints or Multiple Component Constraints
%  b : a vector of the right hand side of the constraint inequality
%  h0: intial edge length between original point dispersion
% Output:
%  m = the number of nearly uniformly dispersed point in the region
%  P = the location of the points in n-d.
%  t = time to converge on a solution
%
tic;
n=size(A,2); % number of columns of A correspond to number of components

% Calls the function to transform verticies n-d simplex to verticies of
% (n-1)-d transformed region
[W,Theta]=transform(n);
J=ones(n,1);
A_tilde=A*Theta; % transformed constraint coefficients
b_tilde=n*(n-1)*(b-(1/n)*A*J); % transformed constraints

for w = 1:n-1 % Define bounding box around (n-1)-d transformed region
    bbox(w,1) = min(W(:,w));
    bbox(w,2) = max(W(:,w));
end
Wext=[W;W(1,:)]; % Duplicates first set of coordinates to close off bounding box

% Calls the function to calculate the distance between any point and the boundaries of
% the constrained design region
fd=inline('Mdistance(p,A,b)','p','A','b');
pfix=W(:,1:end-1); % Fixes the vertices of the bounding box

% Calls the N-D Mesh Generator using Distance Functions by Persson and Strang,
% huniform creates uniform initial spacing of design points
[p,t]=distmeshnd(fd,@huniform,h0,bbox',pfix,Wext,A_tilde,b_tilde);

for ii=1:size(p,1) % Transform p design points back to n-d design region
    s(ii,:) = 1/(n*(n-1))*Theta*[p(ii,:),0]'+(1/n)*J;
end

% Verify that the points are within the design region, and throw out those that are
% outside a certain tolerance. Relevant to more than 3 components.
jj=1; kk = 1;
for k=1:size(s,1)
    Q = A*P(k,:)' - b <= 1*10^(-2)*h0;
    if sum(Q)/length(Q) ==1
        r(jj,:)=s(k,:);
        jj = jj + 1;
    else
        P(kk,:)=s(k,:);
    end
end
m = size(r,1); % Final number of  points within design region
t=toc % Time it took to converge on final solution and transform points back to n-d
```

B transform.m

```
function [Wprime, Theta] = transform(n)
%   Function to project a simplex that is in n dimensions to an n-1
%   dimensional region imbedded in the n dimensional space. Thus the nth
%   coordinates of the transformed vectors can be disregarded as they will
%   all be zero. This formula was taken from Cornell, 2002.
% Input:
%       X: a vector of vertices in n-d
%       n: the dimension of the space
%
% Output:
%       Wprime: the transformed vector with zeros in nth column
%       Theta: the orthonormal transformation matrix

X=diag(ones(n,1));
J=ones(n,n);
Xtilde =n*X - J;
Theta = zeros(n,n);
% The first n-1 columns of Theta
for i=1:(n-1)
    Theta(i,i)=(n-i)*sqrt((n*(n-1))/((n-i)*(n+1-i)));
    Theta(i+1:n,i)=-sqrt((n*(n-1))/((n-i)*(n+1-i)));
end
Theta(:,n)=sqrt(n-1); % Column n of Theta
scale = 1/sqrt(n*(n-1));
Theta = scale*Theta;
Wprime=(n-1)*Xtilde'*Theta; % Transformed vertices of the simplex
```

C Mdistance.m

```
function d = Mdistance(p,A,b)
%   Function to calculate the signed Euclidean distance between any point and
%   the boundary of the design region. The calling function findpoints.m and
%   distmeshnd.m consider negative distances within the design boundary and
%   positive distances on the outside. Matrix A and vector b define the design
%   region. This function utilizes Matlab's preference for dealing with matrices.
% Input:
%  p: a vector of points
% A: a matrix of constraint coefficients
% b: a matrix of constraints
% Output:
% d: a vector of signed distances for each point

M = size(A,1);
K = size(p,1);
% The last column of A is removed. Since each row of p corresponds to a point in
% (n-1)-dimensions and A has n columns, removing the last column is equivalent to
% adding a nth dimension to point p(i) =p(p(1:n-1),0).
Areduced = A(:,1:end-1);
normAr = sqrt(diag(Areduced*Areduced')); % normalize the Areduced matrix

% Calculate the Signed Euclidean distance to each boundary defined by Ap(i) <= b for
% each point.
for k=1:K
    D=inv(diag(normAr,0))*(b-Areduced*p(k,:)');
    d(k)=-min(D(:)); % Save the distance to the closest boundary
end
```

D distmestnd.m

```
function [p,t]=distmeshnd(fdist,fh,h,box,fix,Wext,varargin)
%DISTMESHND N-D Mesh Generator using Distance Functions.
%   [P,T]=DISTMESHND(FDIST,FH,H,BOX,FIX,FDISTPARAMS)
%
%      P:           Node positions (NxNDIM)
%      T:           Triangle indices (NTx(NDIM+1))
%      FDIST:       Distance function
%      FH:          Edge length function
%      H:           Smallest edge length
%      BOX:         Bounding box [xmin,xmax;ymin,ymax; ...] (NDIMx2)
%      FIX:         Fixed node positions (NFIXxNDIM)
%      FDISTPARAMS: Additional parameters passed to FDIST
%
%   Example: Unit ball
%      dim=3;
%      d=inline('sqrt(sum(p.^2,2))-1','p');
%      [p,t]=distmeshnd(d,@huniform,0.2,[-ones(1,dim);ones(1,dim)],[]);
%
%   See also: DISTMESH2D, DELAUNAYN, TRIMESH, MESHDEMOND.
%   Copyright (C) 2004-2006 Per-Olof Persson. See COPYRIGHT.TXT for details.

dim=size(box,2);
 ttol=.1; L0mult=1+.4/2^(dim-1); deltat=.1; geps=1e-1*h; deps=sqrt(eps)*h;

% 1. Create initial distribution in bounding box
if dim==1
  p=(box(1):h:box(2))';
else
  cbox=cell(1,dim);
  for ii=1:dim
    cbox{ii}=box(1,ii):h:box(2,ii);
  end
  pp=cell(1,dim);
  [pp{:}]=ndgrid(cbox{:});
  p=zeros(prod(size(pp{1})),dim);
  for ii=1:dim
    p(:,ii)=pp{ii}(:);
  end
end

if dim <=3
    ptol = 0.001;
else
    ptol = 0.01;
end

% 2. Remove points outside the region, apply the rejection method
p=p(feval(fdist,p,varargin{:})<geps,:);
r0=feval(fh,p);
p=[fix; p(rand(size(p,1),1)<min(r0)^dim./r0.^dim,:)];
N=size(p,1);

count=0;
p0=inf;
```

```
while 1
  % 3. Retriangulation by Delaunay
  if max(sqrt(sum((p-p0).^2,2)))>ttol*h
    p0=p;
    t=delaunayn(p);
    pmid=zeros(size(t,1),dim);
    for ii=1:dim+1
      pmid=pmid+p(t(:,ii),:)/(dim+1);
    end
    t=t(feval(fdist,pmid,varargin{:})<-geps,:);
    % 4. Describe each edge by a unique pair of nodes
    pair=zeros(0,2);
    localpairs=nchoosek(1:dim+1,2);
    for ii=1:size(localpairs,1)
      pair=[pair;t(:,localpairs(ii,:))];
    end
    pair=unique(sort(pair,2),'rows');
    % 5. Graphical output of the current mesh
    if dim==2
      plot(Wext(:,1),Wext(:,2),'k','Marker','*')
      hold on
      trimesh(t,p(:,1),p(:,2),zeros(N,1))
      view(2),axis equal,axis off
      title(['Retriangulation #',int2str(count)]),drawnow, %pause
      hold off

    elseif dim==3
      if mod(count,5)==0
        simpplot(p,t,'p(:,2)>0');
        title(['Retriangulation #',int2str(count)])
        drawnow
      end
    else
        if mod(count,50)==0
            disp(sprintf('Retriangulation #%d',count))
        end
    end
    count=count+1;
  end

  % 6. Move mesh points based on edge lengths L and forces F
  bars=p(pair(:,1),:)-p(pair(:,2),:);
  L=sqrt(sum(bars.^2,2));
  L0=feval(fh,(p(pair(:,1),:)+p(pair(:,2),:))/2);
  L0=L0*L0mult*(sum(L.^dim)/sum(L0.^dim))^(1/dim);
  F=max(L0-L,0);
  Fbar=[bars,-bars].*repmat(F./L,1,2*dim);
  dp=full(sparse(pair(:,[ones(1,dim),2*ones(1,dim)]), ...
                 ones(size(pair,1),1)*[1:dim,1:dim], ...
                 Fbar,N,dim));
  dp(1:size(fix,1),:)=0;
  p=p+deltat*dp;
  % 7. Bring outside points back to the boundary
  d=feval(fdist,p,varargin{:}); ix=d>0;
  gradd=zeros(sum(ix),dim);
  for ii=1:dim
    a=zeros(1,dim);
    a(ii)=deps;
    d1x=feval(fdist,p(ix,:)+ones(sum(ix),1)*a,varargin{:});
    gradd(:,ii)=(d1x-d(ix))/deps;
  end
  p(ix,:)=p(ix,:)-d(ix)'*ones(1,dim).*gradd;

  % 8. Termination criterion
  maxdp=max(deltat*sqrt(sum(dp(d<-geps,:).^2,2)));
  if maxdp<ptol*h, break; end
end
```