Linear Programming

Justin DeBenedetto, Mingue Gao, Lata Kodali, Kelly Kuykendall

April 16, 2013

1 Linear Programming

Linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints.
A typical linear program involves finding a vector **x** that minimizes a linear function $f^\top x$ subject to linear constraints:

$$\min_x f^\top x \text{ such that } \begin{cases} A \cdot x \leq b & \text{(inequality constraint)} \\ A_{eq} \cdot x = b_{eq} & \text{(equality constraint)} \\ lb \leq x \leq ub & \text{(bound constraint)} \end{cases}$$

where **x** is represents the vector of variables (to be determined), $f^\top$ and b are vectors of known coefficients, and A is a matrix of known coefficients.

The inequality constraints create what is known as a *convex polytope*, which is a convex n-dimensional geometric figure. Essentially, this polytope forms the *feasible region*, the set of all points that satisfies all the constraints and sign restrictions of the linear program. This convex polytope is otherwise known as a "simplex." For minimization problem, an *optimal solution*, is a point in the feasible region with the smallest objective function value. Most linear programs have only one optimal solution, but there are cases where a program has no solution or infinitely many.

1.1 Optimal Solution

Some linear programs will not have an optimal solution. There are two main reasons why an optimal solution may not occur. One is when two constraints are inconsistent such as $x \geq 2$ and $x \leq 1$. Hence, the linear program is infeasible. Second is when the polytope is unbounded in the direction of the coefficients of the objective function, i.e. $f^\top$.

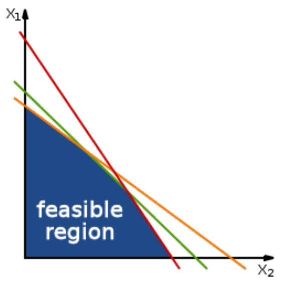
So, if the linear objective function is bounded and a feasible solution exists, then the optimum value is attained on the boundary of the optimal level-set, i.e. on the edges of the polytope. The vertices of the polytope are also called basic feasible solutions.

1.2 Goal of Linear Programming Algorithm

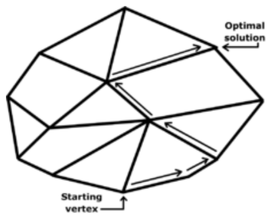
The goal of any linear programming algorithm is the find the optimal solution along the boundary of the feasible region or the "simplex". We discuss one particular algorithm, called the **Simplex Method**, developed by George Dantzig in 1947. By studying the vertices, the simplex algorithm solves the linear program by comparing the function value at each vertex of the feasible region until the smallest is found.

2 Simplex Method

The *simplex method* is a search procedure that sifts through the set of basic feasible solutions, one at a time, until the optimal basic feasible solution (whenever it exists) is identified. The search is done along the sides of the polytope created from the inequality constraints of the linear program.



Feasible Region with 2 Variables



Feasible Region with n variables

We demonstrate the simplex algorithm by considering the following problem:

Minimize $z = 2x - 3y$ subject to

$$\begin{aligned} x + y &\leq 4 \\ x - y &\leq 6 \\ x, y &\geq 0. \end{aligned}$$

We note the standard problem is for maximization and hence must consider maximizing $-z = -2x + 3y$.

2.1 Basic Procedure for a Standard Problem:

1. Convert the problem into a linear system of equations by adding *slack* variables.

$$\begin{array}{rcl} x + y \leq 4 & \Rightarrow & x + y + s = 4. \\ x - y + \leq 6 & \Rightarrow & x - y + t = 6. \\ -z = -2x + 3y & \Rightarrow & 2x - 3y - z = 0. \end{array}$$

2. Convert the system into the initial tableau (essentially an augmented matrix).

Active	**x**	**y**	**s**	**t**	**−z**	**Ans**
s	1	1	1	0	0	4
t	1	-1	0	1	0	6
−z	2	-3	0	0	1	0

$$\Rightarrow \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 \\ 2 & -3 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ s \\ t \\ -z \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 0 \end{bmatrix}$$

We note that the active variables are *slack* variables as well as the quantity $-z$ we are maximizing.

3. The *pivot column* is the column with the <u>most negative value</u> in the objective function. If there are no negatives, stop, you're done.

4. Find the <u>ratios</u> between the non-negative entries in the right hand side and the positive entries in the pivot column. If there are no positive entries, stop, there is no solution.

Active	**x**	**y**	**s**	**t**	**−z**	**Ans**
s	1	$\boxed{1}$	1	0	0	$\boxed{4}$
t	1	$\boxed{-1}$	0	1	0	$\boxed{6}$
−z	2	$\boxed{-3}$	0	0	1	0

Ratio is $4/1 = 4$.
Ratio is $6/(-1) = -6$.

5. The *pivot row* is the row with the smallest *non-negative ratio*. Zero counts as a non-negative ratio.

Active	x	y	s	t	$-$z	Ans
s	1	$\boxed{1}$	1	0	0	4
t	1	-1	0	1	0	6
$-$z	2	-3	0	0	1	0

Since the smallest ratio occurred at the 1 entry, 1 is our pivot. We note if no such non-negative ratio exists, then typically the feasible region is unbounded and no optimal solution occurs.

6. Pivot (perform elementary row operations) where the pivot row and pivot column meet.

Active	x	y	s	t	$-$z	Ans
s	1	$\boxed{1}$	1	0	0	4
t	1	-1	0	1	0	6
$-$z	2	-3	0	0	1	0

$R_2 + R_1$
$R_3 + 3R_1$

Active	x	y	s	t	$-$z	Ans
y	1	$\boxed{1}$	1	0	0	4
t	2	0	1	1	0	10
$-$z	5	0	3	0	1	12

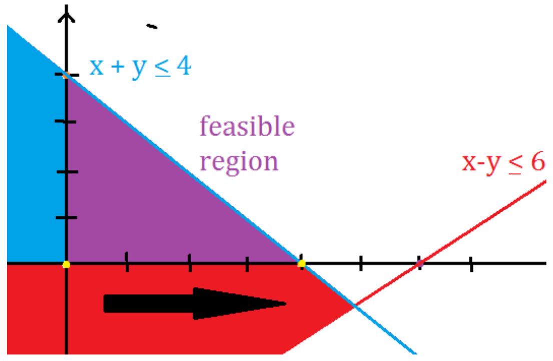
Since the pivot occurred under the column for y, now, y, t, and $-z$ are our active variables.

7. Go back to step 3 until there are no more negatives in the bottom row.
Since there are no more negative in the bottom row, we get the final tableau

Active	x	y	s	t	$-$z	Ans
y	1	$\boxed{1}$	1	0	0	$\boxed{4}$
t	2	0	1	$\boxed{1}$	0	$\boxed{10}$
$-$z	5	0	3	0	$\boxed{1}$	$\boxed{12}$

Hence, our maximum value occurs when $-z = 12/1 = 12$ at $x = 0$ (since x is inactive) and $y = 4/1 = 4$ or at the point $(0, 4)$. Thus, the minimum occurs at $z = 12$.

The demonstration example is involves 2 variables, and so in this case, the boundary of the feasible set of points lies on a polygon in $\mathbb{R}^2$.
In this particular case, the feasible region is



We started at the origin $(0, 0)$ and moved in the horizontal direction (determined by the location of the pivot). Hence, we stop at this vertex since the tableau no longer had negative values.

3 MATLAB: *linprog*

Using MATLAB, the program *linprog* will solve a linear program of the form:

$$\min_x f^\top x \text{ such that } \begin{cases} A \cdot x \leq b \\ A_{eq} \cdot x = b_{eq} \\ lb \leq x \leq ub \end{cases}$$

where f, x, b, b_{eq}, lb, and ub are vectors, and A and A_{eq} are matrices.

The syntax for *linprog*: **x = linprog(f,A,b)** solves $\min_x f^\top x$ such that $Ax \leq b$. By default *linprog* minimizes the objective function, and so to maximize, we use $-f$.

3.1 Diet Problem

This was originally used to figure out rations for armys while maintaining nutrition. Other examples include trying to figure out a balance of different calories and minimizing expenses.

Problem:
We want to feed the cattle in a farm using a diet as *cheap* as possible. Such a diet must contains four types of nutrients that will call **A, B, C,** and **D**. These components can be found in two kind of fodders, **M** and **N**. The amount of every component in grams per kilogram of these fodders is shown in the next table:

	A	**B**	**C**	**D**
M	100	-	100	200
N	-	100	200	100

An animal's daily diet must be mixed at least with 0.4 Kg of **A** component, 0.6 Kg of **B** component, 2 Kg of **C** component and 1.7 Kg of **D** component. The **M** fodder cost 0.2 €/Kg and the **N** fodder 0.08 €/Kg. What quantities of fodders **M** and **N** must be purchased to minimize the cost? Note: 1 kilogram (Kg) is 2.205 pounds.

1. First, determine the decision variables:
   - X_1: quantity of fodder **M** in Kg.
   - X_2: quantity of fodder **N** in Kg.
2. Second, determine the equality and inequality constraints in the components **A**, **B**, **C**, **D**, respectively:

$$\begin{aligned} \mathbf{A}: &\quad 0.1 \cdot X_1 + 0 \cdot X_2 \geq 0.4. \\ \mathbf{B}: &\quad 0 \cdot X_1 + 0.1 \cdot X_2 \geq 0.6. \\ \mathbf{C}: &\quad 0.1 \cdot X_1 + 0.2 \cdot X_2 \geq 2. \\ \mathbf{D}: &\quad 0.2 \cdot X_1 + 0.1 \cdot X_2 \geq 1.7. \end{aligned}$$

   Lastly, $X_1 \geq 0$ and $X_2 \geq 0$.

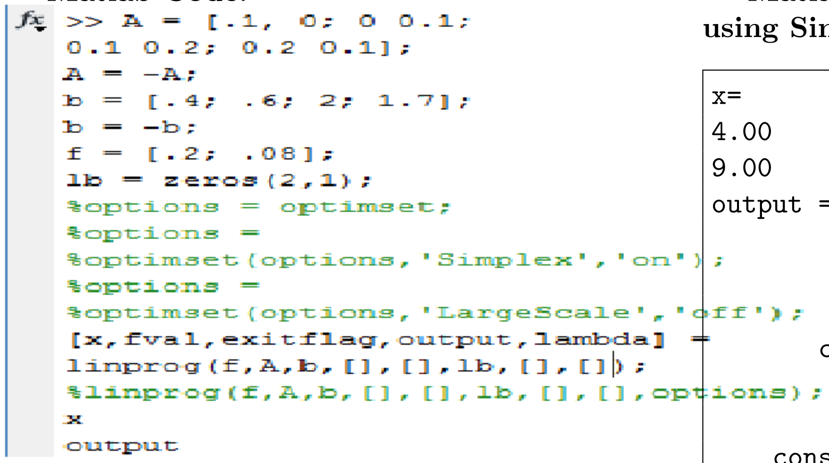
   For Matlab, we need $A \cdot x \geq b$ where A is a matrix and x and b are vectors.

$$A = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \\ 0.1 & 0.2 \\ 0.2 & 0.1 \end{bmatrix} \cdot x = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \geq b = \begin{bmatrix} 0.4 \\ 0.6 \\ 2 \\ 1.7 \end{bmatrix}$$

3. Third, determine the objective function.

$$\text{Minimize } Z = 0.2 \cdot X_1 + 0.08 \cdot X_2.$$

Matlab Code:



```
>> A = [.1, 0; 0 0.1;
0.1 0.2; 0.2 0.1];
A = -A;
b = [.4; .6; 2; 1.7];
b = -b;
f = [.2; .08];
lb = zeros(2,1);
%options = optimset;
%options =
%optimset(options,'Simplex','on');
%options =
%optimset(options,'LargeScale','off');
[x,fval,exitflag,output,lambda] =
linprog(f,A,b,[],[],lb,[],[]);
%linprog(f,A,b,[],[],lb,[],[],options);
x
output
```

Matlab Output (default algorithm of *linprog*):

```
x=
4.00
9.00
```

Matlab Output (replaced with comments using Simplex):

```
x=
4.00
9.00
output =
        iterations: 2
          algorithm:
         'medium scale: simplex'
      cgiterations: []
            message:
         'Optimization terminated.'
   constrviolation: 0
     firstorderopt: 1.7764e-016
```

Solution:
This means purchase 4 Kg = 8.82 lbs of fodder **M** and 9 Kg = 19.845 lbs of fodder **N** in order to maintain the proper diet at a low cost.

3.2 Large-Scale Example:

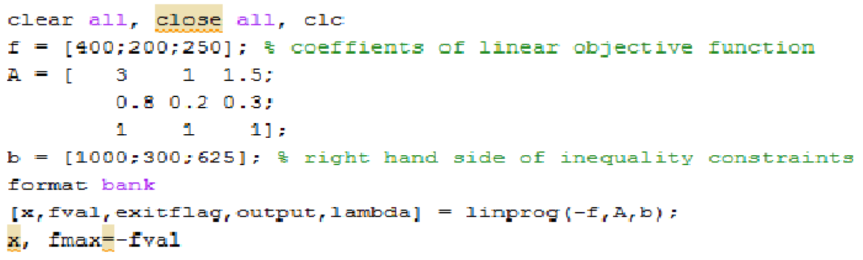
We would like to maximize the area of a farmers land for the best crop yield of corn, wheat and oats with constraints dependent upon acreage, labor and irrigation.
First, we identify the variables:

$$x_1 = \text{acres of corn planted}, \quad x_2 = \text{acres of wheat planted}, \quad x_3 = \text{acres of oat planted}.$$

Next, we identify the constraints:

$$\begin{aligned} 3.0 \cdot x_1 + 1.0 \cdot x_2 + 1.5 \cdot x_3 &\leq 1000. && \text{(irrigation required)} \\ 0.8 \cdot x_1 + 0.2 \cdot x_2 + 0.3 \cdot x_3 &\leq 300. && \text{(labor required)} \\ x_1 + x_2 + x_3 &\leq 625. && \text{(total acreage planted)} \end{aligned}$$

Objective function to be maximized (total yield): $z = 400 \cdot x_1 + 200 \cdot x_2 + 250 \cdot x_3$.
Matlab Input:



```
clear all, close all, clc
f = [400;200;250]; % coeffients of linear objective function
A = [   3     1   1.5;
        0.8 0.2 0.3;
        1     1     1];
b = [1000;300;625]; % right hand side of inequality constraints
format bank
[x,fval,exitflag,output,lambda] = linprog(-f,A,b);
x, fmax=-fval
```

Note: normally we have a positive linprog(f,A,b) if we want to minimize a function, but since in this case we want to find the maximum, we flip the sign.

Matlab Output:

```
Optimization terminated.
x =
128.86
261.57
234.58
fmax = 162500.00
```

Hence, the total yield is 162500.

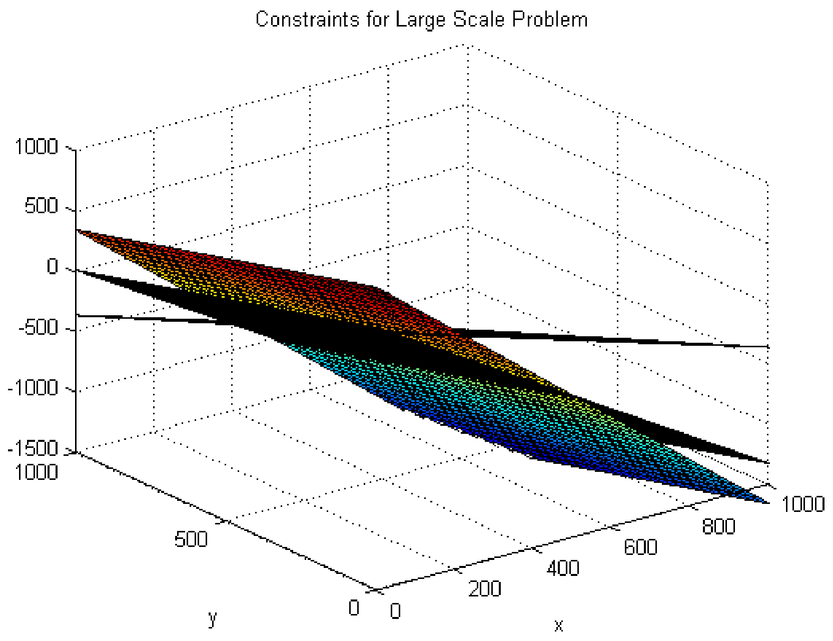
Slack variables (Large Scale Method)

```
b-A*x
ans = 0.00
      74.23
      0
```

Introducing a slack variable replaces an inequality constraint with an equality constraint and a non-negativity constraint.

The following graphs the constraints whose intersections (feasible region) form a polyhedron:

```
ezsurf('625-x-y',[0,1000,0,1000],35)
hold
ezsurf('(1000-3*x-y)/1.5',[0,1000,0,1000],35)
ezsurf('(300-0.8*x-0.2*y)/0.3',[0,1000,0,1000],35)
```



4 Nelder-Mead Method

The Nelder-Mead algorithm or simplex search algorithm, is one of the best known algorithms for multidimensional unconstrained optimization without derivatives.

The method does not require any derivative information, which makes it suitable for problems with non-smooth functions. It can also be used for problems with discontinuous functions, which occur frequently in statistics and experimental mathematics.

This algorithm involves working with an initial simplex. A *simplex* S in $\mathbb{R}^n$ is defined as the convex hull of $n+1$ vertices $x_0, \ldots, x_n$ in $\mathbb{R}^n$. In $\mathbb{R}^2$, the simplex is a triangle, and in $\mathbb{R}^3$, the simplex is a tetrahedron.

The method then performs a sequence of transformations of the working simplex S, aimed at *decreasing* the function values at its vertices. At each step, the transformation is determined by computing one or more test points, together with their function values, and by comparison of these function values with those at the vertices.

4.1 Outline of the algorithm

1. Construct the initial working simplex S.
2. Repeat the following steps until the termination test is satisfied:
   - Calculate the termination test information;
   - If the termination test is not satisfied then transform the working simplex.
3. Return the best vertex of the current simplex S and the associated function value.

We demonstrate the algorithm by minimizing the following example:

$$f(x, y) = x^2 - 4x + y^2 - y - xy.$$

4.2 Construct the initial working simplex S.

The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \dots, x_n$ around a given input point $x_{in} \in R^n$. Typically, the frequent choice is $x_0 = x_{in}$ while the other n vertices are constructed in of two ways:

1. S is right-angled at x_0, based on coordinate axes, or

$$x_j := x_0 + h_j e_j, \qquad j = 1, ..., n$$

   where h_j is a stepsize in the direction of unit vector e_j in R^n.

2. S is a regular simplex, where all edges have the same specified length.

For $f(x, y) = x^2 - 4x + y^2 - y - xy$, we begin with the vertices

$$V_1 = (0, 0), V_2 = (1.2, 0), V_3 = (0, 0.8).$$

4.3 Iteration

One iteration of the Nelder-Mead method consists of the following three steps:

1. **Ordering:** Determine the indices h, s, l of the worst, second worst and the best vertex, respectively, in the current working simplex S

$$f_h = \max_j f_j, f_s = \max_{j \neq h} f_j, f_l = \min_{j \neq h} f_j$$

   For our example:

$$f_h = \max_j f_j = f(0, 0) = 0, f_s = \max_{j \neq h} f_j = f(0, 0.8) = -0.16, f_l = \min_{j \neq h} f_j = f(1.2, 0) = -3.36$$

   Hence, $x_h = (0, 0)$, $x_s = (0, 0.8)$, $x_l = (1.2, 0)$.

2. **Centroid:** Calculate the centroid c of the best side (this is the one opposite the worst vertex x_h)

$$c := \frac{1}{n} \sum_{j \neq h} x_j.$$

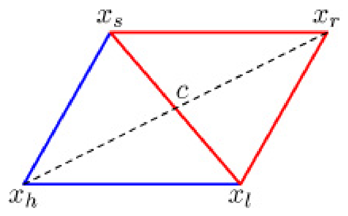
   For our example:

$$c := \frac{1}{n} \sum_{j \neq h} x_j = \frac{1}{2}(x_s + x_l) = \frac{1}{2}[(0, 0.8) + (1.2, 0)] = (0.6, 0.4).$$

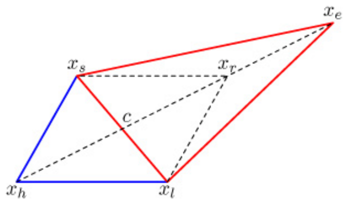
3. **Transformation:** Compute the *new* working simplex from the current one.
   Transformations are controlled by four parameters α for reflection, β for contraction, γ for expansion and δ for shrinkage, which satisfy $\alpha > 0, \quad 0 < \beta < 1, \quad \gamma > 1, \quad \gamma > \alpha, \quad 0 < \delta < 1.$
   Standard values are typically: $\alpha = 1, \beta = \frac{1}{2}, \gamma = 2, \delta = \frac{1}{2}$.

We demonstrate each transformation type graphically:

Reflect (α)
Compute the reflection point
$x_r := c + \alpha(c - x_h)$ and $f_r := f(x_r)$.
If $f_l \leq f_r \leq f_s$,
accept x_r and terminate the iteration.
For 2-dimensions:
$x_r := c + 1 \cdot (c - x_h) = 2c - x_h$.

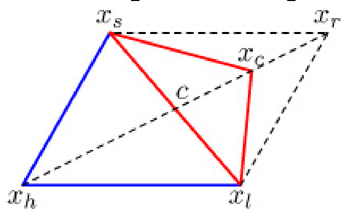


Expand (γ)
If $f_r < f_l$, compute the expansion point
$x_e := c + \gamma(x_r - c)$ and $f_e := f(x_e)$.
If $f_e < f_r$, accept x_e and terminate the iteration.
Otherwise ($f_e \geq f_r$), accept x_r and terminate the iteration.
For 2-dimensions:
$x_e := c + \gamma(x_r - c) = c + 2(x_r - c) = 2x_r - c$.

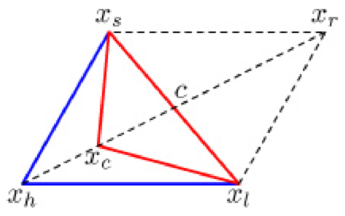


Contract (β)
If $f_r \geq f_s$, compute the contraction point x_c by using the better of the two points x_h and x_r.

Outside: If $f_s \leq f_r < f_h$, compute
$x_c := c + \beta(x_r - c)$ and $f_c := f(x_c)$.
If $f_c \leq f_r$, accept x_c and terminate the iteration.
Otherwise, perform a shrink transformation.
For 2-dimensions:
$x_c := c + \frac{1}{2}(x_r - c) = \frac{1}{2}(x_r + c)$.



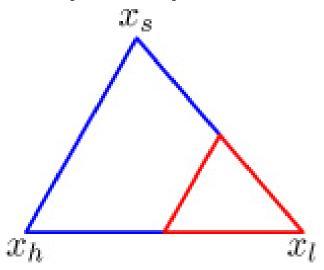
Inside: If $f_r \geq f_h$, compute
$x_c := c + \beta(x_h - c)$ and $f_c := f(x_c)$.
If $f_c < f_h$, accept x_c and terminate the iteration.
Otherwise, perform a shrink transformation.
For 2-dimensions: $x_c := c + \frac{1}{2}(x_h - c) = \frac{1}{2}(x_h + c)$.



Shrink (δ)
Compute n new vertices $x_j := x_l + \delta(x_j x_l)$ and $f_j := f(x_j)$, for $j = 0, ..., n$, with $j \neq l$.
For 2-dimensions: The new vertices are

$$y_1 = x_l + \frac{1}{2}(x_1 x_l), y_2 = x_2 + \frac{1}{2}(x_2 x_l) \text{ and } y_3 = x_l$$

and $f_j := f(y_j)$, for $j = 1, 2, 3$.



We describe the process and 2 iterations for our example $f(x, y) = x^2 - 4x + y^2 - y - xy$.

Process for Transformation:

- First, try to replace only the worst vertex x_h with a better point by using reflection, expansion or contraction with respect to the best side.
- All test points lie on the line defined by x_h and c, and at most two of them are computed in one iteration.
- If this succeeds, the accepted point becomes the new vertex of the working simplex.
- If this fails, shrink the simplex towards the best vertex x_l. In this case, n new vertices are computed.

For our example $f(x, y) = x^2 - 4x + y^2 - y - xy$:

- For the first iteration (creates a new simplex, T_2).
  - $x_r = 2c - x_h = (1.2, 0.8)$ and $f_r = f(1.2, 0.8) = -4.48$.
  - Since $f_r < f_l$, $x_e := 2x_r - c = (1.8, 1.2)$ and $f_e = f(1.8, 1.2) = -5.88$.
  - Since $f_e < f_r$, by expansion, the new triangle (T_2) has vertices

$$V_1 = (1.8, 1.2), V_2 = (1.2, 0), V_3 = (0, 0.8)$$

- For the second iteration (creates a new simplex, T_3).
  - So in T_2,

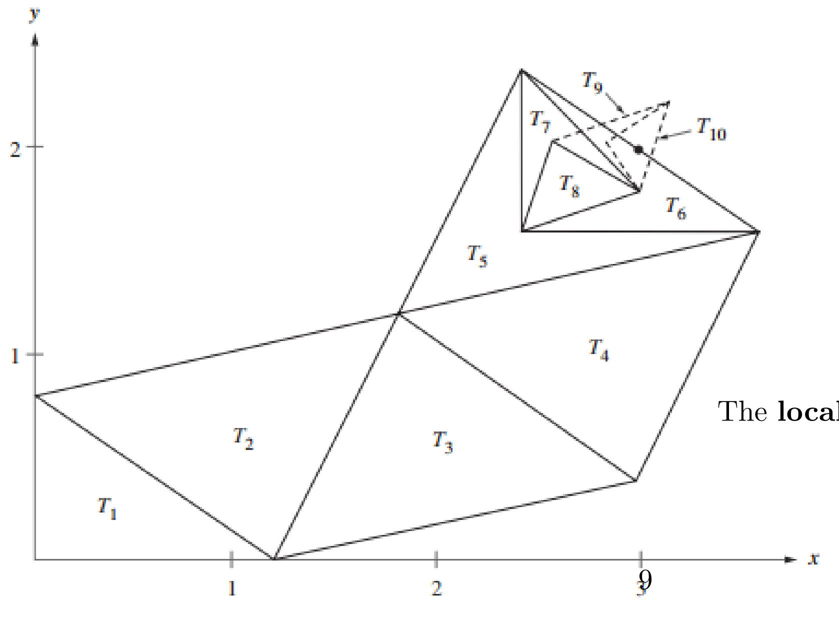
$$f_h = \max_j f_j = f(0, 0.8) = -0.16, f_s = \max_{j \neq h} f_j = f(1.2, 0) = -3.36, f_l = \min_{j \neq h} f_j = f(1.8, 1.2) = -5.88$$

$$c = \frac{1}{2}(x_j + x_l) = (1.5, 0.6)$$

  - $x_r = 2c - x_h = (3, 0.4)$ and $f_r = f(3, 0.4) = -4.44$
  - Since $f_l < f_r < f_s$, by reflection, the new triangle (T_3) has vertices

$$V_1 = (1.8, 1.2), V_2 = (1.2, 0), V_3 = (3, 0.4)$$

The algorithm terminates after the 9 iterations at the point (3,2) and graphically:



T_1 is the original simplex.
T_2 is an expansion.
$T_3, T_4, T_5,$ are reflections.
T_6, T_7, T_8 are contractions.
T_9 is a reflection.
T_{10} is a contraction.

The **local minimum** found is $(3, 2)$.

4.4 Termination

The algorithm terminates if any of the following is true:

term_x. This is the domain convergence or termination test. It becomes true when the working simplex S is sufficiently small in some sense (some or all vertices x_j are close enough).

term_f. This is the function-value convergence test. It becomes true when (some or all) function values f_j are close enough in some sense.

fail. This is the no-convergence test. It becomes true if the number of iterations or function evaluations exceeds some prescribed maximum allowed value.

4.5 Advantages and Disadvantages

Some advantages with using the Nelder-Mead method include significant improvements in the first few iterations and quickly producing satisfactory results. Also, in many cases, the Nelder-Mead algorithm saves time and expense in function evaluation with only one or two function evaluations per iteration, except in shrink transformations, which is rare in occurrence.

However, some disadvantages are the lack of convergence theory and the worst case scenario causes an enormous amount of iterations with negligible improvement in function value. Also, worst case scenarios may cause the algorithm to prematurely terminate.

5 MATLAB: *fminsearch*

Using Matlab, the program *fminsearch* will find the minimum an unconstrained non-linear function: $\min_x f(x)$.
Process:

- Find the minimum of an unconstrained function without derivatives.
- Starts at initial estimate at finds the local minimizer after it either converges to a solution, the maximum number of iterations is reached or the algorithm is terminated by the output function.
- The default algorithm used is the Nelder-Mead Algorithm.

5.1 Example

We minimize the following non-trivial function:

$$f(x,y,z) = -\left(\frac{y}{c}\right)^3 \left(\left(\frac{e^{-(x/c)^k} - e^{-(y/c)^k}}{(y/c)^k - (x/c)^k} \right) - e^{-(z/c)^k} \right)^2 .$$

Matlab Code:

```
>> f = @(x,c,k) -(x(2)/c)^3*
(((exp(-(x(1)/c)^k)-exp(-x(2)/c)^k))/
((x(2)/c)^k-(x(1)/c)^k))
-exp(-(x(3)/c)^k)^2;
 c = 10.1;
 k = 2.3;
[X,fval,exitflag,output] =
fminsearch(@(x) f(x,c,k),[4,10,20])
```

Matlab Output:

```
X = 4.33         27.52           0.00
fval = -2.7709
exitflag = 1
output = iterations: 136
     funcCount: 440
     algorithm:
        'Nelder-Mead simplex direct search'
       message: [1x196 char]
```

References

[1] Introduction to Operations Research. "The Simplex Method." http://www.utdallas.edu/~scniu/OPRE-6201/documents/LP4-Simplex.html.

[2] John H.Mathews and Kurtis K.Fink, *Numerical Methods Using Matlab,* 4th Edition, Prentice-Hall Inc. Upper Saddle River, NJ, 2004.

[3] Math 448/548 - Example 3.4 - Farm Problem. http://www.uccs.edu/~rcascava/Math4480/codes/Farm1.html.

[4] Mathworks. "Linear Programming." http://www.mathworks.com/discovery/linear-programming.html.

[5] Mathworks. "Linear Programming Algorithms." http://www.mathworks.com/help/optim/ug/linear-programming-algorithms.html#f53423.

[6] Mathworks. "Fminserach." http://www.mathworks.com/help/matlab/ref/fminsearch.html.

[7] PHPSimplex. "Simplex Method." http://www.phpsimplex.com/en/simplex_method_theory.htm.

[8] Scholarpedia, "Nelder-Mead algorithm." http://www.scholarpedia.org/article/Nelder-Mead_algorithm.

[9] Stackoverflow. "Maximize Function with Fminsearch." http://stackoverflow.com/questions/10410930/maximize-function-with-fminsearch.

[10] Wayne L. Winston, *Operations Research: Applications and Algorithms,* 4th Edition. Brooks/Cole-Thomson Learning. Belmont, CA. 2004.

[11] Wikipedia. "Linear Programming." http://en.wikipedia.org/wiki/Linear_programming.