Linear Programming

Justin DeBenedetto, Mingyue Gao,
Lata Kodali, Kelly Kuykendall

April 16, 2013

Definition of Linear Programming

Definition of LP:

A technique for the optimization of a *linear* objective function, subject to linear equality and linear inequality constraints.

Typically,

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

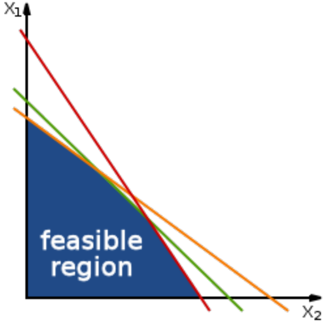
where

- $\mathbf{x}$ is represents the unknowns,
- $f^\top$ and b are *vectors* of known coefficients,
- A is a *matrix* of known coefficients.

Feasible Region

Definition of Feasible Region:

The *feasible region* is the set of all points that satisfies all the constraints and sign restrictions of the linear program. Typically, the feasible region is constructed by the intersection of all the inequality constraints, which forms a convex n-dimensional geometric figure. Also, called a "simplex."



Optimal Solution for a Linear Program

Reasons a Linear Program *fails* to have an optimal solution:

1. Two constraints are inconsistent such as $x \geq 2$ and $x \leq 1$ (an infeasible LP).
2. Feasible Region (polytope) is unbounded (may have a minimum but no maximum).

Optimal Solution for LP:

If the linear objective function is *bounded* and a *feasible solution exists*, then the optimum value is attained on the boundary of the feasible region.
For this reason, the *vertices* of the polytope are called **basic solutions.**

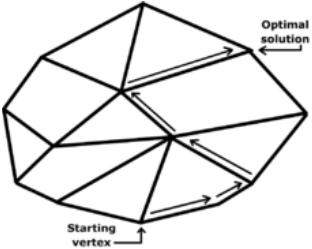
Simplex Method

Goal of LP Algorithm:

The goal of any LP algorithm is to find the optimal solution along the boundary of the feasible region or the "simplex."

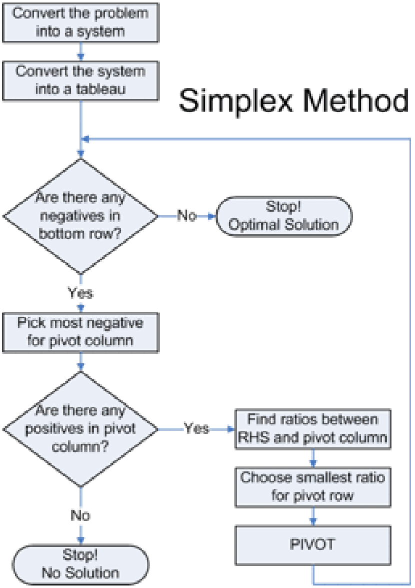
Definition of Simplex Method:

The *Simplex Method* solves the linear program by comparing the function value at each vertex of the feasible region until the smallest is found.



Feasible Region with n variables.

Basic Procedure for Simplex Method



Simplex Method Example

Problem.

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

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

- Standard Problem is for Maximization:

$$\text{Maximize } -z = -2x + 3y$$

1. **Convert Problem in System of Linear Equations:**

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

   - s and t are called *slack* variables.

Steps 2 through 4

2. **Convert the system into the initial tableau.**

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

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

Steps 2 through 4

2. **Convert the system into the initial tableau.**

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

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

   - The *active* variables are s, t, and $-z$.

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.

Steps 2 through 4

2. **Convert the system into the initial tableau.**

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

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

- The *active* variables are s, t, and $-z$.

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> of the pivot column.** If there are no positive entries, stop, there is no solution.

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

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

Steps 5 and 6

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

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

- Since the smallest ratio occurred at the 1 entry, 1 is our pivot.

Steps 5 and 6

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

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

- Since the smallest ratio occurred at the 1 entry, 1 is our pivot.

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

Active	$\mathbf{x}$	$\mathbf{y}$	$\mathbf{s}$	$\mathbf{t}$	$-\mathbf{z}$	**Ans**	
$\mathbf{s}$	1	$\boxed{1}$	1	0	0	4	
$\mathbf{t}$	1	-1	0	1	0	6	$R_2 + R_1$
$-\mathbf{z}$	2	-3	0	0	1	0	$R_3 + 3R_1$

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

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

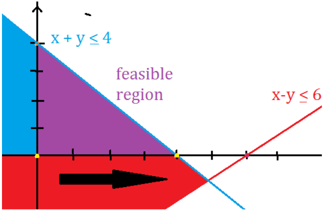
Last Step

(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	$\boxed{1}$	$\boxed{12}$

- Maximum value occurs when $-z = 12/1 = 12$.
- $x = 0$ (since x is inactive) and $y = 4/1 = 4$ or at the point $(0, 4)$.
- Minimum occurs at $z = 12$.

Graphically:



MATLAB: *linprog*

linprog

The program *linprog* solves LPs:

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

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

Basic Syntax

x = linprog(f,A,b) solves $\min_x f^\top x$ such that $Ax \leq b$.

- Default is to minimize f. So, to maximize use $-f$.
- *linprog* uses three different algorithms.
    - Default is 'interior-point' and can use Simplex instead.

linprog Example 1:

Diet 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 **BM** and **N** must be purchased to minimize the cost?
Note: 1 kilogram (Kg) is 2.205 pounds.

linprog Example 1:

1. First, determine the decision variables:
   - X_1: quantity of fodder **M** in Kg.
   - X_2: quantity of fodder **N** in Kg.

linprog Example 1:

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:

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

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

linprog Example 1:

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:

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

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

3. Third, determine the objective function.

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

linprog Example 1:

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

linprog Example 1:

```
>> 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 (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.

linprog Large-Scale Example:

Farming Problem.

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:

- $x_1 =$ acres of corn planted,
- $x_2 =$ acres of wheat planted,
- $x_3 =$ acres of oat planted.

linprog Large-Scale Example:

Farming Problem.

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:

- $x_1 =$ acres of corn planted,
- $x_2 =$ acres of wheat planted,
- $x_3 =$ acres of oat planted.

Next, we identify the constraints:

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

linprog Large-Scale Example:

Farming Problem.

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:

- $x_1 =$ acres of corn planted,
- $x_2 =$ acres of wheat planted,
- $x_3 =$ 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):

$z = 400 \cdot x_1 + 200 \cdot x_2 + 250 \cdot x_3.$

linprog Large-Scale Example:

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: in order to maximize, must use $-f$.

Matlab Output:

Optimization terminated.
x =
128.86
261.57
234.58
fmax = 162500.00

Solution: The total yield is 162500.

linprog Large-Scale Example:

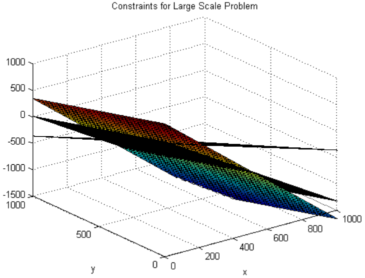
Slack variables (Large Scale Method)

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.

Graphing the inequality constraints:

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)



Nelder-Mead Algorithm

Nelder-Mead Algorithm

- The Nelder-Mead algorithm or simplex search algorithm is a type of multidimensional unconstrained optimization *without derivatives*.
- Ideal for non-smooth functions.

Nelder-Mead Algorithm

Nelder-Mead Algorithm

- The Nelder-Mead algorithm or simplex search algorithm is a type of multidimensional unconstrained optimization *without derivatives*.
- Ideal for non-smooth functions.
- 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.

Nelder-Mead Algorithm

Nelder-Mead Algorithm

- The Nelder-Mead algorithm or simplex search algorithm is a type of multidimensional unconstrained optimization *without derivatives*.
- Ideal for non-smooth functions.
- 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.

Definition of Nelder-Mead

After obtaining the initial simplex, the method then performs a *sequence of transformations* of the working simplex S.
The goal is to *decrease* the function value at the vertices by testing points.

Algorithm Outline

1. Construct the initial working simplex S.

Algorithm Outline

1. Construct the initial working simplex S.
   - The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \ldots, x_n$ around a given input point $x_{in} \in R^n$.

Algorithm Outline

1. Construct the initial working simplex S.
   - The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \ldots, x_n$ around a given input point $x_{in} \in R^n$.
2. Repeat the following steps until the termination test is satisfied:
   - Calculate the termination test information;

Algorithm Outline

1. Construct the initial working simplex S.
   - The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \ldots, x_n$ around a given input point $x_{in} \in R^n$.
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.

The iteration:

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

Algorithm Outline

1. Construct the initial working simplex S.
   - The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \ldots, x_n$ around a given input point $x_{in} \in R^n$.
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.

   The iteration:

   1. **Ordering:** Determine the indices h, s, l of the worst, second worst and the best vertex, respectively for x_h, x_s, x_l, in the current working simplex S.
   2. **Centroid:** Calculate the centroid c of the best side (this is the one opposite the worst vertex x_h).

Algorithm Outline

1. Construct the initial working simplex S.
   - The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \ldots, x_n$ around a given input point $x_{in} \in R^n$.
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.

   The iteration:

   1. **Ordering:** Determine the indices h, s, l of the worst, second worst and the best vertex, respectively for x_h, x_s, x_l, in the current working simplex S.
   2. **Centroid:** Calculate the centroid c of the best side (this is the one opposite the worst vertex x_h).
   3. **Transformation:** Compute the *new* working simplex from the current one (4 types).
      - Reflection(α)
      - Expansion(γ)
      - Contraction(β)
      - Shrinkage (δ)

Algorithm Outline

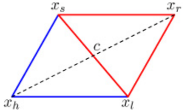
1. Construct the initial working simplex S.
   - The initial simplex S is usually constructed by generating $n+1$ vertices $x_0, \ldots, x_n$ around a given input point $x_{in} \in R^n$.
2. Repeat the following steps until the termination test is satisfied:
   - Calculate the termination test information;
   - If the termination test is not satisfied then **transform** the working simplex.

   The iteration:
   1. **Ordering:** Determine the indices h, s, l of the worst, second worst and the best vertex, respectively for x_h, x_s, x_l, in the current working simplex S.
   2. **Centroid:** Calculate the centroid c of the best side (this is the one opposite the worst vertex x_h).
   3. **Transformation:** Compute the *new* working simplex from the current one (4 types).
      - Reflection(α)
      - Expansion(γ)
      - Contraction(β)
      - Shrinkage (δ)
3. Return the best vertex of the current simplex S and the associated function value.

Graphical Interpretation of Transformations:

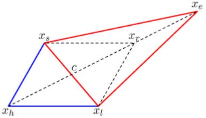
Reflect ($\alpha = 1$)

Compute
$x_r := c + 1 \cdot (c - x_h) = 2c - x_h$ and
$f_r := f(x_r)$.

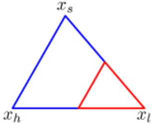


Expand ($\gamma = 2$)

Compute $x_e := c + \gamma(x_r - c) = c + 2(x_r - c) = 2x_r - c$.



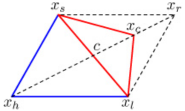
Shrink (δ)



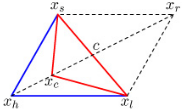
Contract ($\beta = 1/2$)

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
$x_c := c + \frac{1}{2}(x_r - c) = \frac{1}{2}(x_r + c)$.



Inside: If $f_r \geq f_h$, compute
$x_c := c + \frac{1}{2}(x_h - c) = \frac{1}{2}(x_h + c)$ and
$f_c := f(x_c)$.



Example with first two iterations.

Function

Consider

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

and the following vertices:

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

1. Ordering (worst, good, best):

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

$$f_l = \min_{j \neq h} f_j = f(1.2,0) = -3.36$$

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

2. Centroid:

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

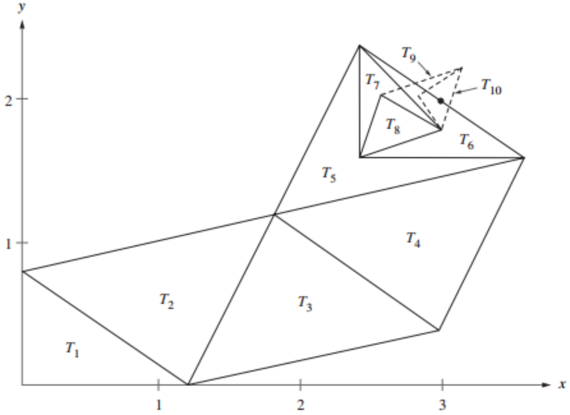
For $f(x,y) = x^2 - 4x + y^2 - y - xy$ with $x_h = (0,0), x_s = (0,0.8), x_l = (1.2,0)$.

- 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 $f(x,y) = x^2 - 4x + y^2 - y - xy$ with $x_h = (0,0), x_s = (0,0.8), x_l = (1.2,0)$.

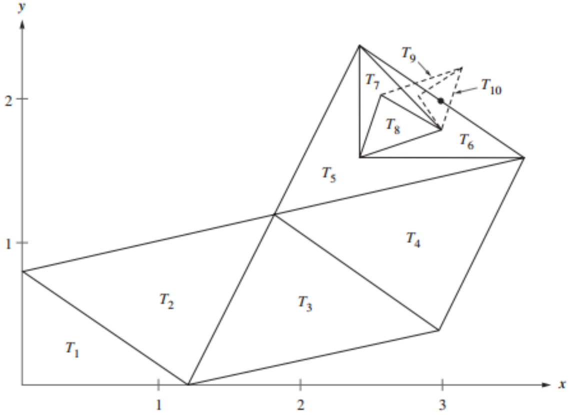
- For the first iteration (creates a new simplex, T_2).
  - $$x_r = 2c - x_h = (1.2, 0.8)$$
    and
    $$f_r = f(1.2, 0.8) = -4.48.$$
  - Since $f_r < f_l$,
    $$x_e := 2x_r - c = (1.8, 1.2)$$
    and $f_e = f(1.8, 1.2) = -5.88$.
  - Since $f_e < f_r$, by expansion, the new triangle (T_2) has vertices
    $$V_1 = (1.8, 1.2), V_2 = (1.2, 0), V_3 = (0, 0.8)$$
- 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
    $$V_1 = (1.8, 1.2), V_2 = (1.2, 0), V_3 = (3, 0.4)$$

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



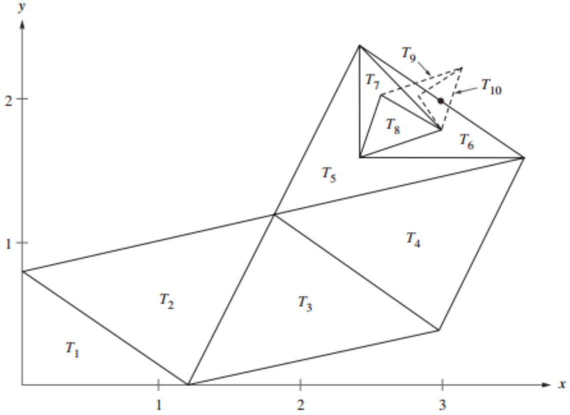
- T_1 is the original simplex.

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



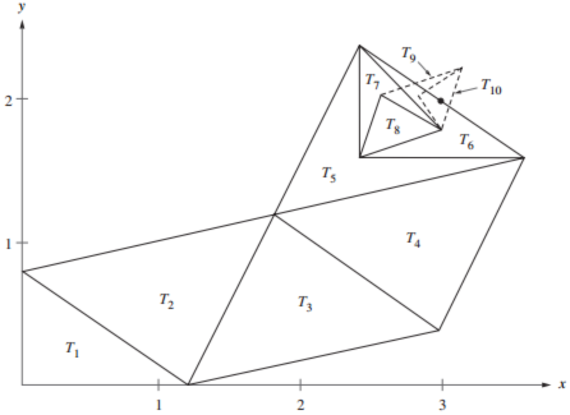
- T_1 is the original simplex.
- T_2 is an expansion.

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



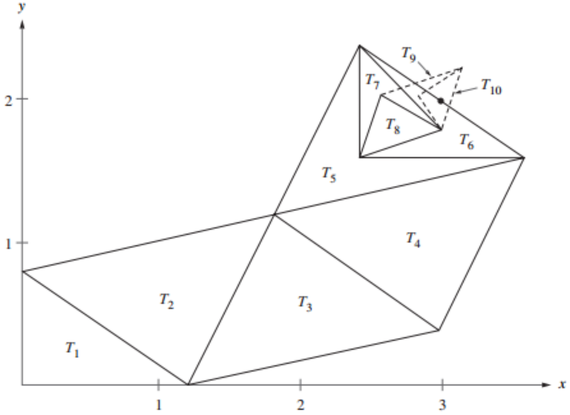
- T_1 is the original simplex.
- T_2 is an expansion.
- T_3, T_4, T_5, are reflections.

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



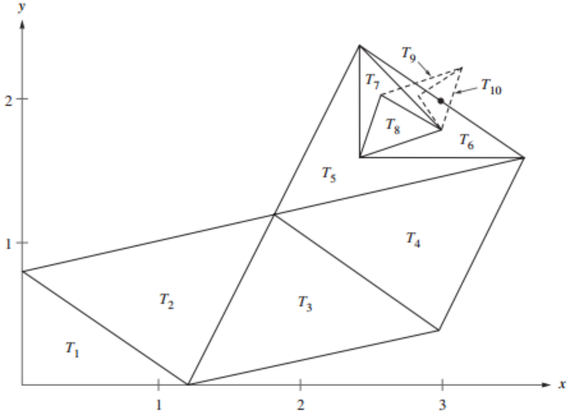
- T_1 is the original simplex.
- T_2 is an expansion.
- $T_3, T_4, T_5,$ are reflections.
- T_6, T_7, T_8 are contractions.

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



- 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.

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



- 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).

Termination and Conclusions

Termination

The algorithm terminates if any of the following is true:

- $term\backslash\_x.$ This is the domain convergence or termination test. It becomes true when the working simplex S is sufficiently small (some or all vertices x_j are close enough).
- $term\backslash\_f.$ This is the function-value convergence test. It becomes true when (some or all) function values f_j are close enough.
- $fail.$ This is the no-convergence test. It becomes true if the number of iterations or function evaluations exceeds some prescribed maximum allowed value.

Termination and Conclusions

Termination

The algorithm terminates if any of the following is true:

- $term\backslash\_x$. This is the domain convergence or termination test. It becomes true when the working simplex S is sufficiently small (some or all vertices x_j are close enough).
- $term\backslash\_f$. This is the function-value convergence test. It becomes true when (some or all) function values f_j are close enough.
- $fail$. This is the no-convergence test. It becomes true if the number of iterations or function evaluations exceeds some prescribed maximum allowed value.

Advantages/Disadvantages

- Advantages:
  - Has significant improvements in the first few iterations and quickly producing satisfactory results.
  - In many cases, only one or two function evaluations per iteration, except in shrink transformations, which is rare in occurrence.
- Disadvantages:
  - Lack of convergence theory.
  - Worst case scenario causes an enormous amount of iterations.

MATLAB: *fminsearch*

fminsearch

Minimizes an unconstrained non-linear function:

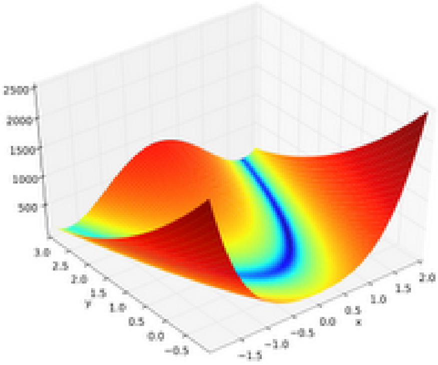
$$\min_x f(x).$$

Process:

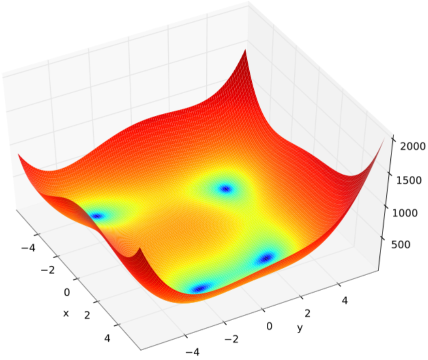
- Finds the minimum of an unconstrained function <u>without</u> derivatives.
- Starts at initial estimate at finds the local minimizer after it either converges to a solution, the maximum number of iterations is reached or the algorithm is terminated by the output function.
- The default algorithm used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

Classic Large Scale Examples.

Examples we are familiar with-
Rosenbrock function:



Himmelblau function:



fminsearch Non-trivial Large Scale Example

Large-Scale Function

We minimize the following non-trivial function:

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

for $c, k \in \mathbb{R}$.

fminsearch Non-trivial Large Scale Example

Matlab Input:

```
>> 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]