Linear Programming

Initial Examples

Obvious Example

A company has two car factories. One in Baltimore and the other in Annapolis. They only have enough raw materials to make one car. The profit in Annapolis is $1000 per car, and the profit in Baltimore is $1200 per car.

The company is going to make the car in Baltimore.

Less Obvious Example

A company has two car factories. One in Baltimore and the other in Annapolis. The profit in Annapolis is $1000 per car, and the profit in Baltimore is $1200 per car.

The factories make different cars. They have 100 crates of raw materials. The cars in Annapolis use up 2 crates, and those in Baltimore use up 3 crates. Both factories can only make at most 35 cars.

Think it through. How many cars will get made in each city?

Less Obvious Example

A company has two factories that make spice. They have to decide how much spice to make. It can be any real number, not just whole numbers like cars.

One factory in Baltimore and the other in Annapolis. The profit in Annapolis is $1000 per pound, and the profit in Baltimore is $1200 per pound.

The factories make spice in different ways that use different amounts of raw materials. They have 100 different crates of different raw materials. There are 1000 combinations of those materials which can make spice. Each combination is a linear equation for making one pound of spice, like 10% of box 1 plus 20% of what is in box 2 can make one pound of spice.

So the company has to decide how much spice to make using each combination.

Some amount of spice has been pre-sold so the factories have to make at least that much. There is a taxation or legal requirement to make a certain amount in each city. Market research indicates you can't sell more than 1000 pounds of spice.

The number of equations has increased dramatically, and so has the number of variables.

Also, the result is no longer in whole numbers.

Generic Linear Programming

This is the generic situation in linear programming. The decisions are real numbers. The number of variables can be large. The number of equations can be large. The equations can be inequalities with equalities. There is a single function to optimize. All equations are linear.

Linear Programming Setup

Definition: The **decision variables** x_i are values that can be chosen. Linear programming involves finding the best choice for those variables under some constraints.

Definition: An **objective function** $\sum c_i x_i$ is a linear function of the decision variables.

The goal of linear programming is to maximize the objective function.

A typical example in business would be the goal of maximizing profit. The objective function would need to be the profit, and it would be necessary to express the profit as a linear function.

Typically one cannot just pick whatever values one wants. There are constraints.

Definition: The **standard form** of a linear program is to maximize an objective function

subject to a system of linear inequalities

$\sum_j a_{ij} x_j \leq b_i$

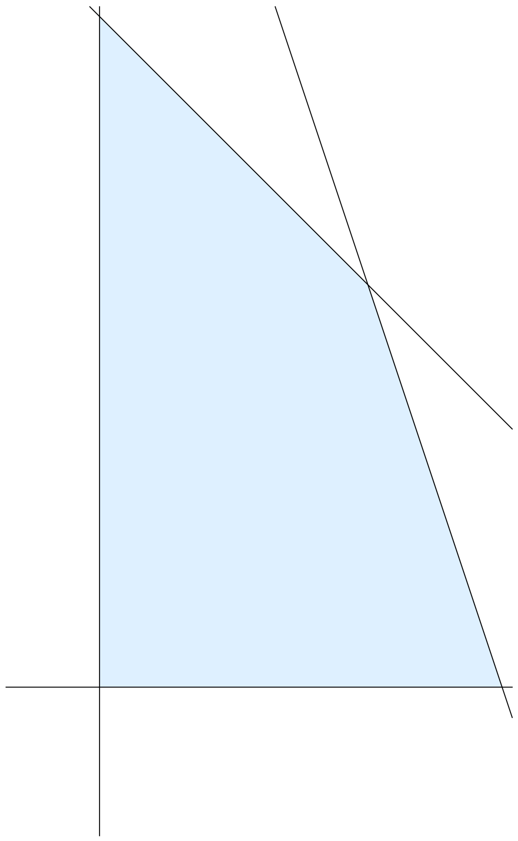
and subject to $x_i \geq 0$

Polytope

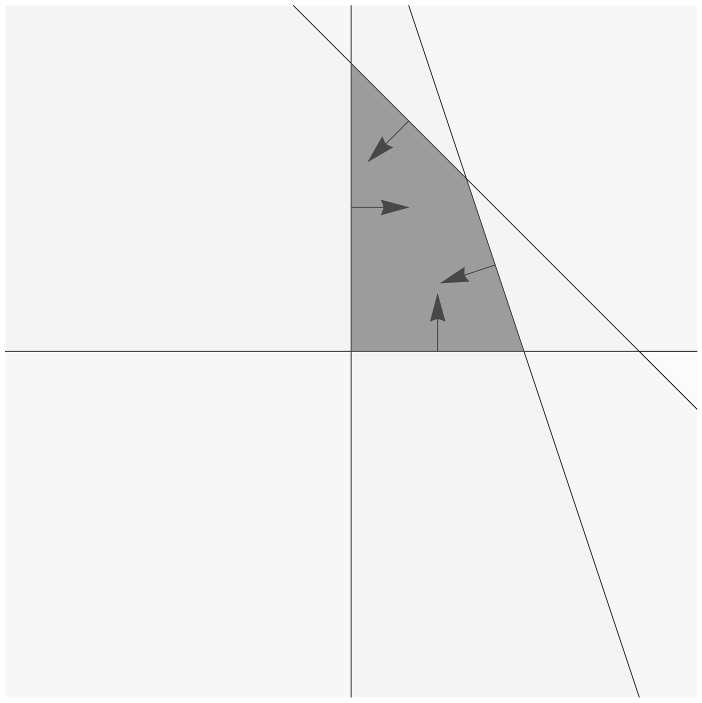
Recall that subject a system linear inequalities
$\sum_j a_{ij} x_j \leq b_i$
is a polytope when it is bounded.

The basic example is in 2D when the edges correspond to the extremal cases $\sum_j a_{ij} x_j = b_i$

Out[ ]=

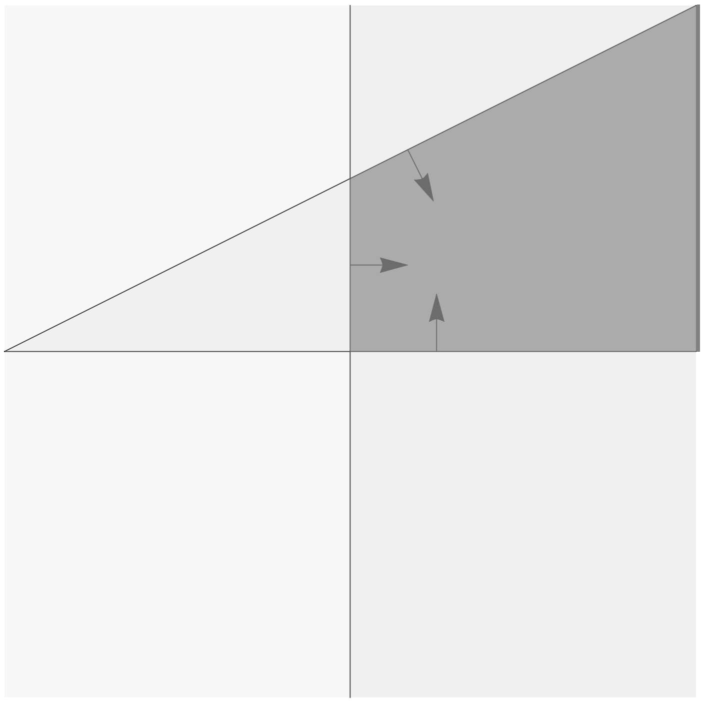


Out[ ]=



Unboundedness

Unless it is unbounded. It could be that there are not enough inequalities to close it in, or it could be that there is a vector v where $\sum_j a_{ij} v \leq 0$ in which case all large multiples of v will satisfy the inequalities.

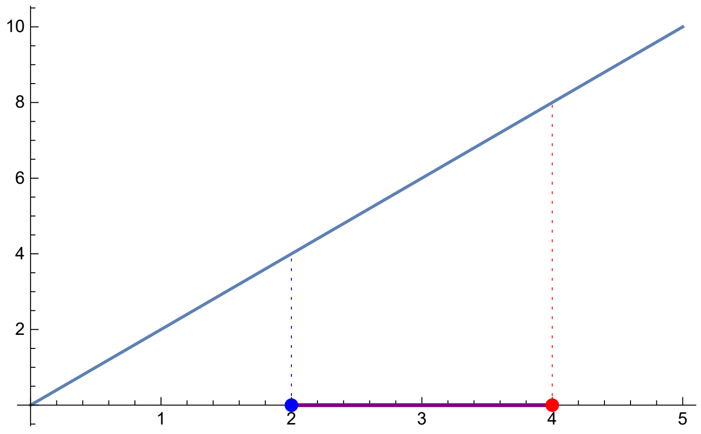
Out[∘]=


Maximum is on a vertex

1D case

If you have a line segment on the number line, then any objective function will have its maximum on an endpoint unless the objective function is a constant. In 1D the only possibility for the objective function to be constant is to be the zero function.

Generically, the objective function is not constant, but it is always linear so the maximum is on an endpoint.

Out[∘]=


For a line segment, it doesn't matter how many dimensions it is embedded in, the maximum of the objective function on a line segment will be on its endpoints. One way to see this is that the objective function is a linear function restricted to the line segment, just like the 1D case.

Maxima in higher dimensions

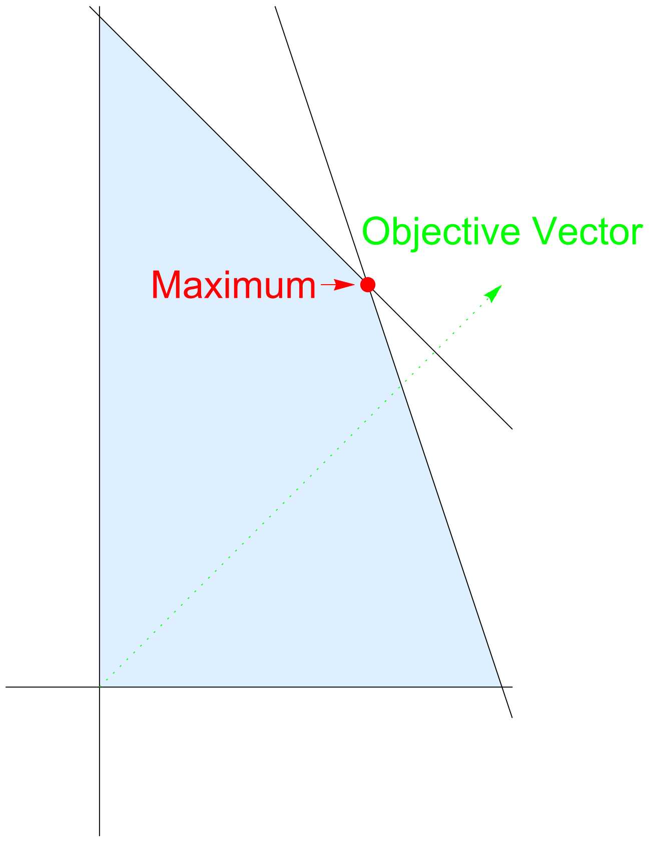
For a bounded 2D face, the maximum of the objective function will be on its vertices. One way to see this is that any interior point lines on a line segment containing edge points so the maximum is achieved on an edge point. Any edge point lies on a line segment between vertices.

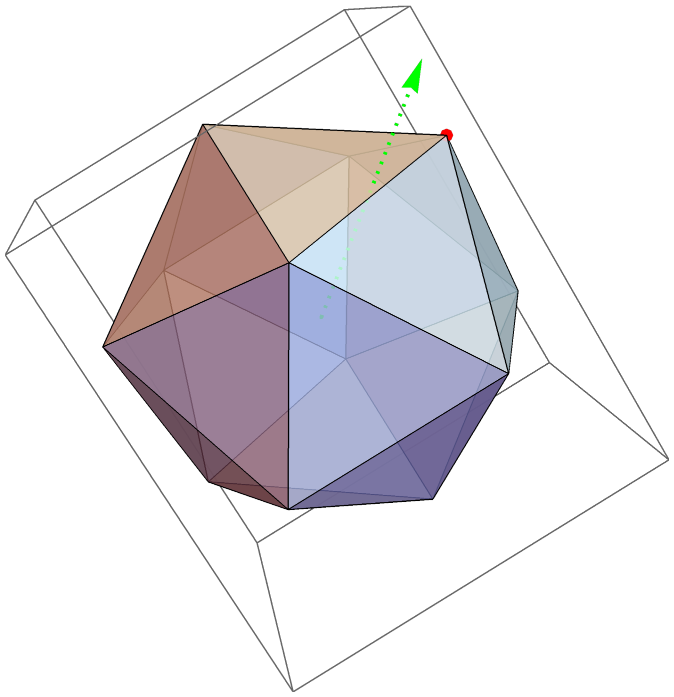
Bounded higher dimensional faces are bounded by faces of one dimensional less, so by the same argument the maximum of the objective function is realized at a vertex. This type of argument is called induction.

Note that in higher dimensions it is possible for the maximum to occur at many points, even when the objective function is not degenerate. But we are always guaranteed that the maximum is achieved at a vertex.

The maximum of the objective function is achieved at a vertex.

Out[ ]=



Out[ ]=


Simplex Method

The purpose of these notes is not be a complete explanation of linear programming. For our purposes, we will see the need for a non-trivial algorithm when the dimension is greater than two.

The goal is to find a vertex where the maximum of the objective function is reached.

The 2D case is misleadingly simple, but also shows the basic idea of a directed search.

One can order the vertices by angle and then proceed around the polygon in the direction that increases the objective function.

In higher dimensions, the vertices don’t come with such a natural ordering and instead one has to choose which vertex to try next.

Also, because it is difficult to calculate vertices in higher dimensions, instead of calculating them all at once, you would like a method that calculates only the next one you will try.

The method is called the Simplex Method.

Using LinearProgramming

The first argument is the objective vector *c*. The objective function is *c* dot *x*.

```
In[ ]:= c = {1, 2, 3};
```

$x_1 + 2 x_2 + 3 x_3$

The second argument is the matrix of coefficients m.

The third argument is the bounds b of the inequalities.

Standard Form

In the standard form, the constraints are given the other way. *m* dot $x \geq b$ instead of less than or equal to *b*

```
m = {{1, 0, 3}, {4, 0, -4}, {6, -1, -9}, {-9, 4, -3}};
```

$$\begin{pmatrix} 1 & 0 & 3 \\ 4 & 0 & -4 \\ 6 & -1 & -9 \\ -9 & 4 & -3 \end{pmatrix}$$

```
b = {9, 1, 6, 7};
```

$$\begin{pmatrix} 9 \\ 1 \\ 6 \\ 7 \end{pmatrix}$$

$$x_1 + 3 x_3 \geq 9$$
$$4 x_1 - 4 x_3 \geq 1$$
$$6 x_1 - x_2 - 9 x_3 \geq 6$$
$$-9 x_1 + 4 x_2 - 3 x_3 \geq 7$$

In Mathematica, the function to use is LinearProgramming.

In Matlab, it is linprog

In Python (scipy) it is scipy.optimize.linprog

In all cases, the basic usage is with three arguments.

```
In[ ]:= solution = LinearProgramming[c, m, b]
```

Out[ ]= $\left\{\frac{37}{7}, \frac{102}{7}, \frac{26}{21}\right\}$

We can see that this solution is bigger or equal to b.

```
In[ ]:= m.solution
```

Out[ ]= $\left\{9, \frac{340}{21}, 6, 7\right\}$

```
In[ ]:= m.solution - b
```

Out[ ]= $\left\{0, \frac{319}{21}, 0, 0\right\}$

This dot product is the maximum of c dot x with those constraints.

```
In[ ]:= c.solution
```

Out[ ]= $\frac{267}{7}$

This function is pretty complicated. Here is the help in Mathematica.

In[∘]:=
```
? LinearProgramming
```

Out[∘]=

Symbol

LinearProgramming[c, m, b] finds a vector x that

minimizes the quantity $c.x$ subject to the constraints $m.x \geq b$ and $x \geq 0$.

LinearProgramming[c, m, {{b_1, s_1}, {b_2, s_2}, …}] finds a vector x that minimizes $c.x$ subject to $x \geq 0$

and linear constraints specified by the matrix m and the pairs {b_i, s_i}. For each row m_i of m, the

corresponding constraint is $m_i.x \geq b_i$ if s_i == 1, or $m_i.x$ == b_i if s_i == 0, or $m_i.x \leq b_i$ if s_i == −1.

LinearProgramming[c, m, b, l] minimizes $c.x$ subject to the constraints specified by m and b and $x \geq l$.

LinearProgramming[c, m, b, {l_1, l_2, …}] minimizes $c.x$ subject to the constraints specified by m and b and $x_i \geq l_i$.

LinearProgramming[c, m, b, {{l_1, u_1}, {l_2, u_2}, …}]

minimizes $c.x$ subject to the constraints specified by m and b and $l_i \leq x_i \leq u_i$.

LinearProgramming[c, m, b, lu, dom] takes the elements of x to be in the domain dom, either Reals or Integers.

LinearProgramming[c, m, b, lu, {dom_1, dom_2, …}] takes x_i to be in the domain dom_i.

Inequalities going the other way plus equals

In Mathematica

Instead of a vector b, the third argument can be given as a list to specify which row has ≥, = , or ≤

```
{{bound1, sign1}, {bound2, sign2}, ...}
```

The value of sign determines ≥, = , or ≤

If sign2 = 1, then row 2 has a ≥

If sign3 = 0, then row 3 has a =

If sign4 = -1, then row 4 has a ≤

```
LinearProgramming[c,
 {{1, 0, 3},
  {4, 0, -4},
  {6, -1, -9},
  {-9, 4, -3}},

 {{9, 1},
  {1, 1},
  {6, 0},
  {7, -1}}
]
```

Means the system of inequalities is

$x_1 + 3\,x_3 \geq 9$
$4\,x_1 - 4\,x_3 \geq 1$
$6\,x_1 - x_2 - 9\,x_3 == 6$
$-9\,x_1 + 4\,x_2 - 3\,x_3 \leq 7$

Note the third and four row are different from before.

In Matlab, the default is m·x ≤ b and the cases where it is equals is an optional fourth and fifth argument to linprog. The fourth argument are the coefficient matrix for the rows that are equal, and the fifth argument is the vector of constants it is equal to.

In Python, apparently one is supposed to use scipy for linear programming. It works in a similar way to Matlab's linprog.

Greater than to Less than

If you want to switch between greater than or equal to less than or equal, just multiply by minus 1.

$\sum_j a_{ij} x_j \leq b_i$
is equivalent to
$\sum_j -a_{ij} x_j \geq -b_i$

Slack variables

Often it is convenient to introduce new variables. They are called slack variables because they can be any non-negative number.

For example, if you want to force $x_1 > x_2$ then you can introduce a new variable w_1 and introduce the linear equality $x_1 = x_2 + w$

Equality

By trichotomy, one can replace a linear equality with two inequalities.

$\sum_j a_{ij} x_j = b_i$
becomes
$\sum_j a_{ij} x_j \leq b_i$
$\sum_j a_{ij} x_j \geq b_i$
or
$\sum_j a_{ij} x_j \leq b_i$
$\sum_j -a_{ij} x_j \leq -b_i$

Boundary values

In the standard problem the decision variables are non-negative $x_i \geq 0$

To allow other boundary constraints, change variables.

For example, to get $x_i \leq 0$ set $y_i = -x_i$ and rewrite the problem in y_i

In Mathematica, one changes the boundary constraints with the optional fourth argument.

In Matlab or Python (scipy), the boundary values are determined by an optional sixth argument.

Integer Programming

While linear programming over real numbers is fast and efficient, requiring the solutions to be integers is slower and often the results are less accurate.

You can ask Mathematica to do it, and you might be pushing the thing to the limit.

Many discrete optimization problems can be phrased as integer programming problems. There is always the possibility of using the real number optimization and finding a nearby integer, although it may not be the theoretical best among the integers.

Examples

Choose five foods. The goal is to minimize cost under the constraint of getting enough calories and vitamins.

Choose five foods. The goal is to maximize vitamins under the constraints of not getting too much unhealthy nutrition.