Ch. 5 - Basic Optimization

As engineers, we try to find ways to make things work,

When we can, we try to find the **best** way to make things work.

How do you determine what is "best"?

Typical approach:

- Identify **control variables**: parameters you can tune:

$$\boldsymbol{x} = x_0, x_1, x_2, \ldots, x_{n-1}$$

- Choose an **objective function**: f
  - Function of control variables
  - Measures the scalar "quality" or "cost" of a design:

$$f(\boldsymbol{r}) : \mathbb{R}^n \rightarrow \mathbb{R}$$

- Identify **constraints** (relationships to be satisfied by control variables) and **feasible set** (allowed choices of control variables satisfying constraints)
  - Equality and inequality constraints:

$$g_j(\boldsymbol{x}) = 0; \quad g_k(\boldsymbol{x}) \leq 0$$

Problem statement:

- Find the combination of control variable values that gives the maximum quality or minimum cost:

$f(\boldsymbol{x_0})$ has a minimum at $\boldsymbol{x_0}$ in region R (discuss global vs. local minimum)

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x_0}) \qquad \forall \boldsymbol{x} \in R$$

Sign convention:

- Maximizing f is equivalent to minimizing $-f$
- We can choose either sign.
- Typically choose sign leading to <u>minimization</u>

Note: We are also motivated by physics laws formulated as minimum principles:

- Stable equilibrium corresponds to minimum energy
- Soap bubbles minimize surface area
- Hamilton's principle: dynamics minimizes a time integral involving energy
- Snell's law describes refraction in terms of minimum travel time

Initial **classification of optimization problems**:

- **Unconstrained vs. Constrained**
- **Derivative-free vs. derivative methods**

Start simple with unconstrained, derivative-free optimization with a single control variable and work up from there...

Section 5.1 - Unconstrained Optimization (Derivative-Free Methods)

If you think back to calculus class, optimization may seem quite straightforward:

- Compute the derivative: $f'(x) = \frac{df}{dx}$
- Solve $f'(x_0) = 0$ to find extrema.
- Check the sign of $f''(x)$ to distinguish maxima, minima, and inflection points:

$$f''(x_0) > 0 \implies \text{min}$$

Note on necessary and sufficient conditions

$f'(x_0) = 0 \text{ and } f''(x_0) \geq 0$ are <u>necessary</u> conditions for a minimum.

$f'(x_0) = 0 \text{ and } f''(x_0) > 0$ are <u>sufficient</u> to ensure a <u>local</u> minimum. The inequality rules out saddle points and behavior set by by higher order terms.

Global minimum: - Not ensured by local conditions. - Need to compare values at relative minima (and endpoints of search interval).

Multiple control variables

- Vector of first derivatives (gradient) of the objective functions must vanish.
- First derivative condition becomes:

$$f'(x) = 0 \rightarrow \nabla f(\boldsymbol{x}) = \frac{\partial f}{\partial x_i} = 0$$

- Dropped the subscript from x_0 for simplicity
- Remember that these conditions are evaluated at the candidate minimum point.
- Second derivative condition involves how each component of the gradient changes with respect to each control variable (Hessian): $f''(x) > 0 \rightarrow \text{ the Hessian, } H(\boldsymbol{x}) = [\frac{\partial^2 f}{\partial x_i \partial x_j}]$, is positive (semi-)definite.

Origin of multivariable condition

Once again, our friend Brook Taylor (of Taylor series fame) helps us out. Call the local minimum x^* and consider points nearby given by $x = x^* + \alpha d$. Nearby values expressed by expanding objective function in a Taylor series about x^*:

$$f(x) = f(x^*) + \alpha \nabla f(x^*)^T d + \frac{\alpha^2}{2} d^T H(x^*) d + O(\alpha^3)$$

Applying the first condition criterion, ∇f vanishes, so the <u>second order necessary condition</u> is:

$$d^T H(x^*) d \geq 0 \; \forall \; d$$

In other words, the Hessian must be positive semi-definite (necessary condition). Positive definite Hessian is sufficient to ensure a local minimum at x^*.

The basic idea of checking derivatives sounds good, but may not be practical.

In particular, evaluating the derivative may not be feasible:

- Analytic expression not available
- Prohibitively expensive to compute

Alternative: **Derivative-free optimization**

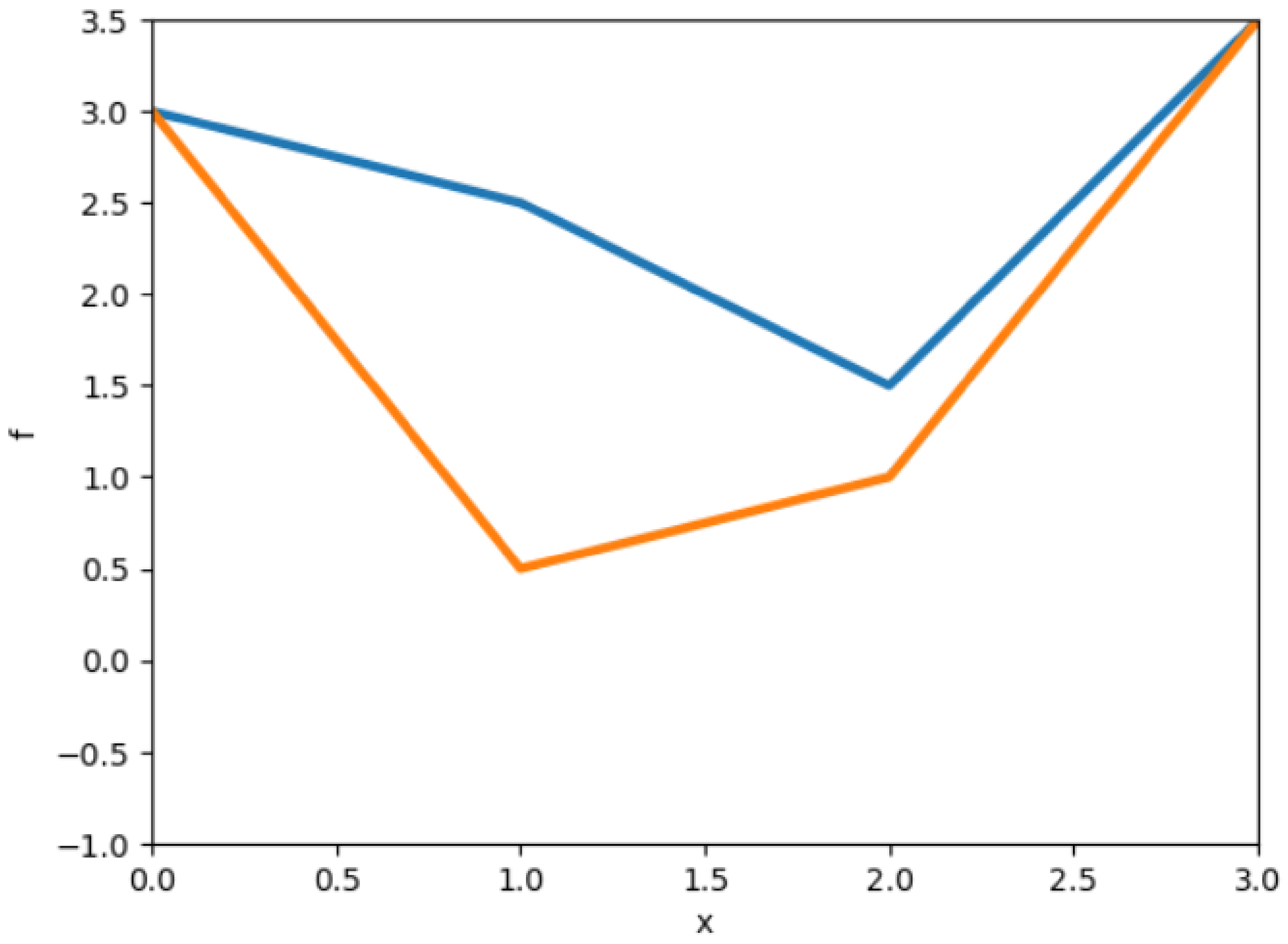
- Evaluate the function at desired points, but do NOT evaluate derivative.
- Similar to basic root-finding methods.
- Assume that a search interval is given in which to find the minimum.

Golden section search

- Assumes that the function is:
    - continuous
    - **unimodal** (monotonic on either side of a single minimum).
- Systematically shrink the interval where that single minimum must reside by computing only values of the function without computing derivatives.
- Basic approach:
    - Compute the values at the 2 endpoints and at 2 internal points.
    - Number the points increasing order x_0, x_1, x_2, x_3 with corresponding function values f_0, f_1, f_2, f_3.
    - Exclude part of the interval based on f_1 and f_2.
    - How exactly should that be done?

Consider examples:

```
# execute initial notebook cell for imports including `from simple_plot import *`
indexplot(np.array([0,1,2,3]), np.array([[3,2.5,1.5,3.5],[3,.5,1,3.5]]).T, labels=['x
```



How do we narrow the search interval? Internal point with larger value becomes an endpoint, and repeat...

Exclude a portion of the interval, adjust one of the interval bounds, compute at 2 new internal points, and continue refining.

To avoid computing function value at 2 new internal points, use the previously computed internal point and just compute at 1 new point

This is where the golden ratio $\phi = (1+\sqrt{5})/2 \approx 1.6$ comes into play.

- Choose evaluation points 38% and 62% of the way along the interval ($62/38 \approx 1.6$).
- Next subdivision will coincide with the previous sample point.
- Optimal convergence rate (among such subdivision schemes).

See code in notebook...

Successive Parabolic Interpolation

Alternative method for 1D derivative-free optimization using interpolants. Allows for contribution from points not specfied a priori.

- Start with an interval to search for the optimum.
- Evaluate the function at the endpoints x_1, x_3.
- Evaluate the function at an interior point starting with $x_2 = (x_1 + x_3)/2$.
- Construct parabola through 3 points by Lagrange interpolation (low degree $\rightarrow$OK):

$$p(x) = f(x_1)\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + f(x_2)\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + f(x_3)\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

- Compute the minimum of the parabola as a proxy for the minimum of the function:

$$x_0 = \frac{x_1+x_2}{2} - \frac{(f_2-f_1)(x_3-x_1)(x_3-x_2)}{2[(x_2-x_1)(f_3-f_2)-(f_2-f_1)(x_3-x_2)]}$$

- Trim interval based on comparison of new interior value $f(x_0)$ and old interior value $f(x_2)$:

$$x_0 < x_2 \implies : x_1^{new} = x_1^{old}, x_2^{new} = x_0, x_3^{new} = x_2^{old}$$

$$x_0 > x_2 \implies : x_1^{new} = x_2^{old}, x_2^{new} = x_0, x_3^{new} = x_3^{old}$$

Keep the portion of the interval that includes the minimum, and use it as the new interior point.

Matlab code for successive parabolic interpolation is on p. 98, and you are encouraged to code it up in python as an exercise. Matlab also has a library function `fminbnd` described on p.99. You might want to find a python library function for 1D derivative-free optimization and test it out.

Section 5.2 - Unconstrained Optimization (Derivative Methods)

Having see derivative methods for 1D optimization before (calculus), let's go directly to multidimensional case.

Start with $n = 2$ so we can more readily create plots to show results.

On p. 99 an elementary (but useful) example is introduced: Find the minimum of the function

$$f(x,y) = x^2 + 3y^2$$

Here we will not avoid use of derivatives (or gradient in the multi-variable case):

$$\nabla f(\boldsymbol{x}) = \frac{\partial f}{\partial x}\hat{\boldsymbol{x}} + \frac{\partial f}{\partial y}\hat{\boldsymbol{y}} = 2x\,\hat{\boldsymbol{x}} + 6y\,\hat{\boldsymbol{y}}$$

After making initial guess to serve as the starting point, how can we progressively move toward the minimum?

- Think of the level sets of the function (which typically form "rings" about the minimum in its neighborhood).
- Move toward the minimum by progressively moving "downhill" to level sets associated with smaller function values.

Having evaluated the function at our initial location, what direction should we NOT move when we want to decrease the function value? In waht direction should we move?

- Function value does not change if we move along (parallel to) level set we start on.
- Need to move normal/perpendicular to the current level set.

The function value changes (at its maximum rate) if we move:

- Normal to the level set.
- Along (or opposite) the direction of the gradient.

Method of Steepest Descent a.k.a. **Dethod of Gradient Descent**:

- Evaluate the gradient at the starting point.
- Take a step in the direction opposite the gradient (since gradient is the direction of increasing function values).
- Continue until we are sufficiently close to a minimum (when $\left|\nabla f\right| < tol$).

Switch to notebook to see code for gradient descent and see how it works:

Section 5.3 - Linear Programming

- Add constraints.
- Control variables cannot take on arbitrary values, but must satisfy relationships called **constraints**.
- Control values that satisfy the costraints are called the **feasible set**.
- Seek the minimum function value within the feasible set.

The discussion starts (naturally) with a linear function and linear **equality constraints**:

$$\begin{aligned}
\text{minimize} \quad & c_0x_0 + c_1x_1 + \ldots + c_{(n-1)}x_{n-1} \\
\text{subject to} \quad & a_{00}x_0 + a_{01}x_1 + \ldots + a_{0(n-1)}x_{n-1} = b_0 \\
& a_{10}x_0 + a_{11}x_1 + \ldots + a_{1(n-1)}x_{n-1} = b_1 \\
& \vdots \\
& a_{(m-1)0}x_0 + a_{(m-1)1}x_1 + \ldots + a_{(m-1)(n-1)}x_{n-1} = b_{m-1} \\
\text{and} \quad & x_0 \geq 0,\, x_1 \geq 0,\, \ldots, x_{n-1} \geq 0
\end{aligned} \tag{1}$$

In vector form this becomes:

$$\begin{aligned}
\text{minimize} \quad & c^T x \\
\text{subject to} \quad & Ax = b \text{ and } x \geq 0
\end{aligned} \tag{2}$$

A system with **inequality constraints** has the form:

$$\begin{aligned}
\text{minimize} \quad & c_0x_0 + c_1x_1 + \ldots + c_{0(n-1)}x_{n-1} \\
\text{subject to} \quad & a_{00}x_0 + a_{01}x_1 + \ldots + a_{0(n-1)}x_{n-1} \leq b_0 \\
& a_{10}x_0 + a_{11}x_1 + \ldots + a_{1(n-1)}x_{n-1} \leq b_1 \\
& \vdots \\
& a_{(m-1)0}x_0 + a_{(m-1)1}x_1 + \ldots + a_{(m-1)(n-1)}x_{n-1} \leq b_{m-1} \\
\text{and} \quad & x_0 \geq 0,\, x_1 \geq 0,\, \ldots, x_{n-1} \geq 0
\end{aligned} \tag{3}$$

Inequality constraints can be converted to the standard form with **equality constraints** by introducing **slack variables**:

$$
\begin{array}{rl}
\text{minimize} & c_0x_0 + c_1x_1 + \ldots + c_{0(n-1)}x_{n-1} \\
\text{subject to} & a_{00}x_0 + a_{01}x_1 + \ldots + a_{0(n-1)}x_{n-1} + y_0 = b_0 \\
& a_{10}x_0 + a_{11}x_1 + \ldots + a_{1(n-1)}x_{n-1} + y_1 = b_1 \\
& \vdots \\
& a_{(m-1)0}x_0 + a_{(m-1)1}x_1 + \ldots + a_{(m-1)(n-1)}x_{n-1} + y_{m-1} = b_{m-1} \\
\text{and} & x_0 \geq 0,\ x_1 \geq 0,\ \ldots, \ldots, x_{n-1} \geq 0 \\
\text{and} & y_0 \geq 0,\ y_1 \geq 0,\ \ldots, y_{m-1} \geq 0
\end{array}
$$

In matrix-vector notation this becomes:

$$
\begin{array}{rl}
\text{minimize} & c^T x \\
\text{subject to} & \bar{A}\bar{x} = b \text{ and } \bar{x} \geq 0 \\
\text{where} & \bar{A} = [A, I] \ \text{ and } \ \bar{x} = [x, y]^T
\end{array}
\tag{4}
$$

Before we start solving, some relevant definitions:

- Any vector x that satisfies the constraints is a **feasible solution**.
- If, in addition, the value of the objective function is minimal (with respect to other feasible solutions), then it is an **optimal solution**.
- For a **basic feasible solution** the number of non-zero solution variables is equal to the number of constraints; i.e. the number of zero entries in $\bar{x}$ matches the original number of variables n.

These definitions set us up for the following essential result:

Fundamental Theorem of Linear Programming:

Given a linear program in the standard form (with equality constraints) where A is an $m \times n$ matrix of rank m:

(i) if there exists a feasible solution, there is a basic feasible solution.

(ii) if there is an optimal feasible solution, there is an optimal basic feasible solution.

> **The shorthand version:** For solutions of linear programming problems:
>
> - feasible $\implies$ $\exists$ basic feasible
> - optimal feasible $\implies$ $\exists$ optimal basic feasible

On this basis, our typical goal is to **compute an optimal basic feasible solution**.

An example: The Brewer's Problem

(Thanks to Robert Sedgewick and Kevin Wayne of Princeton)

- Small brewery produces ale and beer.
- Resources are limited.
- Ale and beer need different inputs and produce different profits.

Inputs and profits

	corn	hops	malt	profit
available	480	160	1190	
ale(per barrel)	5	4	35	13
beer(per barrel)	15	4	20	23

Brewer's problem:

- Choose product mix to maximize profit

Our goals:

- Consider some feasible options.
- Find an optimal solution to increase profit!?

Some example allocations of resources with profit

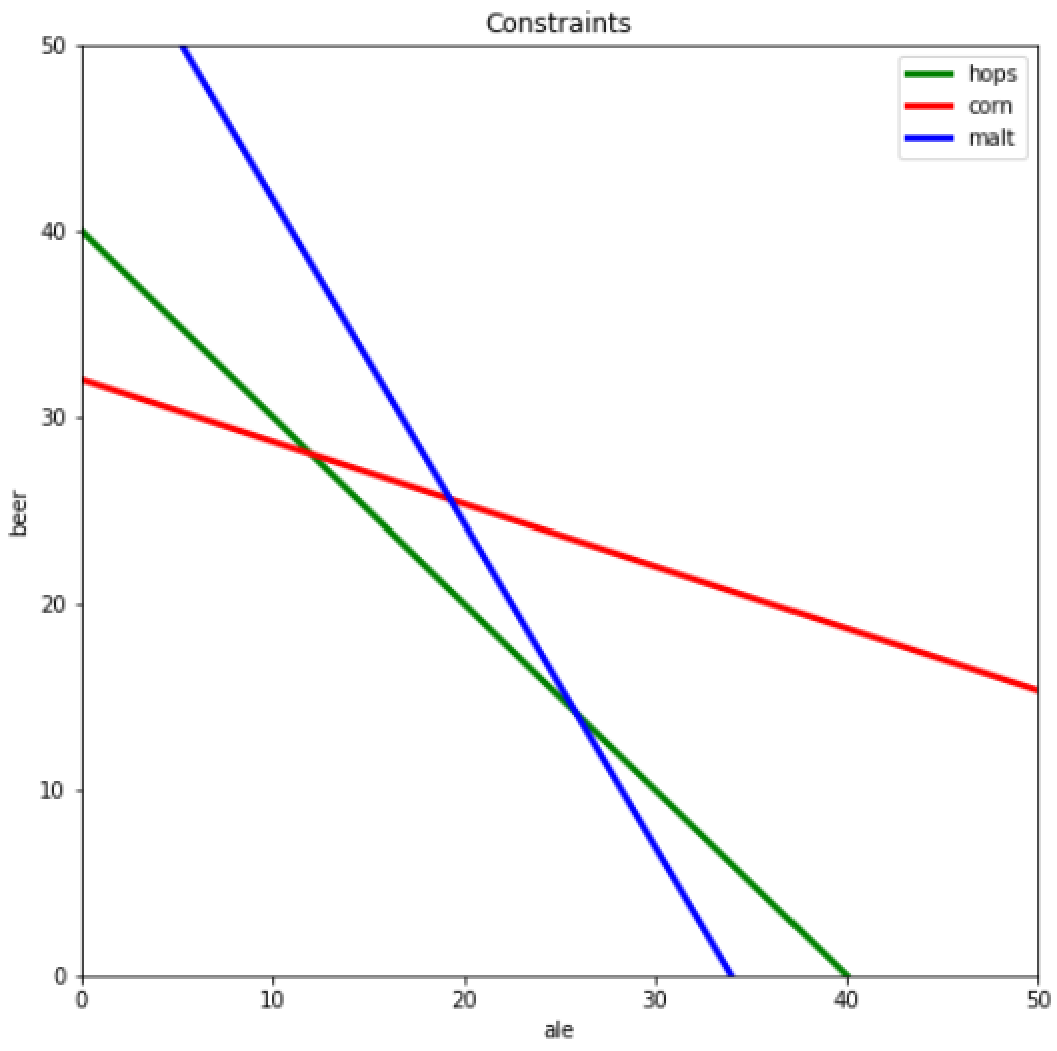
Product mix	corn	hops	malt	profit
all ale (34)	179	136	1190	442
all beer (32)	480	128	540	736
20 ale + 20 beer	400	160	1100	720
12 ale + 28 beer	480	160	980	800
better mix?	?	?	?	>800?

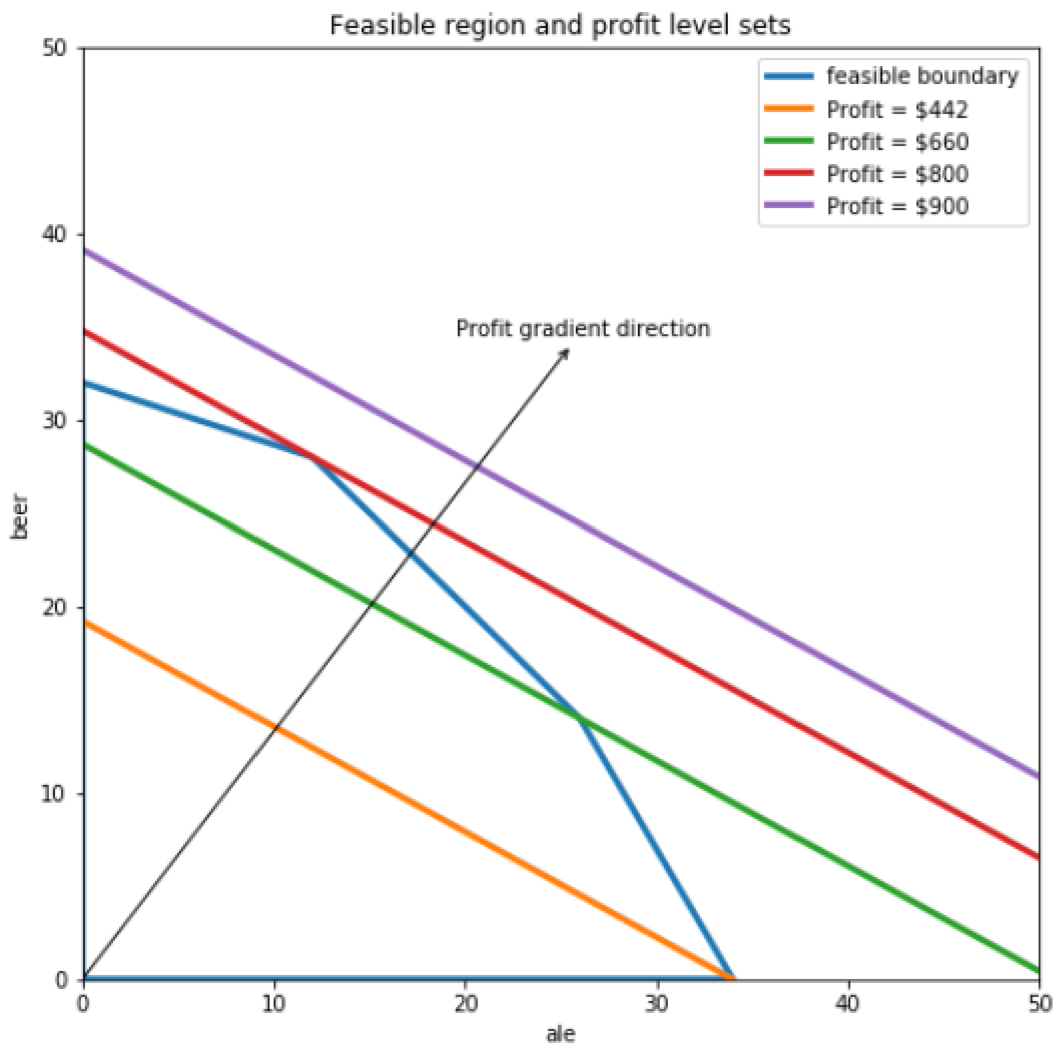
How do we go about solving?

Formulate as linear program: ($A = \text{ ale}, B = \text{ beer}$)

$$\begin{array}{lrll} \text{maximize} & 13A + 23B & & \text{profit} \\ \text{subject to the constraints} & & & \\ & 5A + 15B & \leq 480 & \text{corn} \\ & 4A + 4B & \leq 160 & \text{hops} \\ & 35A + 20B & \leq 1190 & \text{malt} \\ & A & \geq 0 & \\ & B & \geq 0 & \end{array}$$

Plot constraints and identify feasible region (which must lie in the first quadrant, where A and B are positive, and below each of the constraint lines). See notebook for code.


Constraints
hops
corn
malt
beer
ale
0
10
20
30
40
50



Linear objective function $\implies$ level sets are parallel lines normal to the profit (objective function) gradient direction.

Picture motivates a geometric conclusion: Linear objective function $\implies$ an optimal solution occurs at an **extreme point** (a vertex of the polygon bounding the feasible region).

If objective gradient $\perp$ to feasible region edge? Entire edge is optimal solutions so an optimal solution does occur at extreme point (actually 2 of them...).

So we can find an optimal solution by only testing the extreme points.

Even better, the linear constraints create a feasible region that is **convex**: the line connecting any 2 feasible points consists entirely of feasible points.

Convince yourself that the convexity of the feasible region ensures that an extreme point which is locally optimal (compared with neighboring vertices) must actually be the globally optimal solution. (Think about how you would need to change the constraint polygon to create a local optimum that was not a global optimum.)

> **Convexity ensures that a local optimum is the global optimum.**

Together there solutions form the basis of a solution plan:

- Find a solution at an extreme point (constraint vertex).
- Move to neighboring vertices that improve the objective function value.

- When no neighboring vertex offers an improved value, stop because that local optimum is actually the global optimum solution.

The next section looks at the details of implementing this solution method.

Section 5.4 - Simplex Method

Start by introducing slack variables (and moving the objective function definition to the bottom) to write the system in standard form (with only equality constaints):

	ale		**beer**						
maximize	Z								
subject to	5A	+	15B	$+S_c$				=480	corn
	4A	+	4B		$+S_h$			=160	hops
	35A	+	20B			$+S_m$		=1190	malt
	13A	+	23B				-Z	=0	profit
							A,B,S_c, S_h, S_m	≥ 0	

We now have a linear algebra problem with $m+1=4$ equations in the 6 variables A, B, S_c, S_h, S_m, Z.

As usual, we don't want to have to rewrite variable names all the time, so let's construct the coefficient matrix which in this context is called the **simplex tableau**:

$$\begin{bmatrix} 5 & 15 & 1 & 0 & 0 & 480 \\ 4 & 4 & 0 & 1 & 0 & 160 \\ 35 & 20 & 0 & 0 & 1 & 1190 \\ 13 & 23 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which is the matrix of coefficients that multiply the vector $[A, B, S_c, S_h, S_m]^T$ augmented with the right-hand side vector.

Note that this is a particular instance of the following block matrix (with subscripts indicating array shape):

\begin{bmatrix} A_{m \times n} & I_{m \times m} & b_{m \times 1} \ c_{1 \times n}^T & 0_{m \times 1} & 0_{1 \times 1} \end

where A is the $m \times n$ matrix of coefficients from the constraint equations, I is the $m \times m$ identity matrix, b is the $m \times 1$ vector of constraint values, and c is the $n \times 1$ coefficient vector from the objective function (also $c = \nabla f$).

> **Note**: The notation in the text corresponds to ordering the vector of variables with the slack variables before the control variables.

The plan to compute the solution involves row reductions (just as we did in Ch. 2 - Linear Systems).

> No other row contains a Z, so it is common not to bother explicitly including Z or its coefficient. Insteadm the Z information is contained implicitly:
>
> - The bottom row specifies how the input variables affect the profit (objective function value).

- The bottom-right entry gives the negative of the profit for a particular set of input values.

Given all of that, our example involves the following values:

- Number of constraints: $m = 3$
- Number of control variables: $n = 2$
- Total number of variables (control + slack + objective): $N = m + n + 1 = 3 + 2 + 1 = 6$

Having assembled the tableau, this is a good time to introduce the idea of basic variables.

A **basic variable** is any variable that appears only with coefficient 1 in tableau.

These variables are "basic" in the sense that their coefficient subarray provides a basis for m-dimensional space. At this point, the basic variables are the slack variables S_c, S_h, S_m and their coefficient submatrix (the $m \times m$ identity matrix) provides a basis for 3-space.

At this point, we can already obtain a **basic feasible solution (BFS)**:

- Set $n - m$ variables to 0
- Solve m equations for remaining m unknowns.
- If unique and feasible $\implies$ BFS.
- BFS $\iff$ extreme point

We can already carry out these steps (but perhaps not in the most interesting way) to get:

Initial basic feasible solution:

- Set the the non-basis (control) variables to zero: A=B=0 (and Z=0)
- Solve for the slack variables: $S_c = 480, S_h = 160, S_m = 1190$ (Initially we can read them from the last column.)
- Extreme point on simplex (at the origin) with zero profit

Now "pivot" to an adjacent vertex by a row operation that moves a different variable into the basis.

Here let's choose to move B into the basis, so divide the top equation by 15 and do row operations to zero out the other coefficient in second column.

The original tableau was:

$$\begin{bmatrix} 5 & 15 & 1 & 0 & 0 & 480 \\ 4 & 4 & 0 & 1 & 0 & 160 \\ 35 & 20 & 0 & 0 & 1 & 1190 \\ 13 & 23 & 0 & 0 & 0 & 0 \end{bmatrix}$$

with the variables $[A, B, S_c, S_h, S_m]^T$, and the updated tableau is:

$$\begin{bmatrix} 1/3 & 1 & 1/15 & 0 & 0 & 32 \\ 8/3 & 0 & -4/15 & 1 & 0 & 32 \\ 85/3 & 0 & -4/3 & 0 & 1 & 550 \\ 16/3 & 0 & -23/15 & 0 & 0 & -736 \end{bmatrix}$$

We moved B into the basis by moving another variable out. Which one?

And we get a new basic feasible solution (BFS):

$A = S_c = 0, B = 32, S_h = 32, S_m = 550$. The updated profit is improved to Z=736 (from the previous Z=0).

Before we go on, why did we choose to "pivot" on B (i.e. introduce B into the basis)?

Objective coefficient was positive (each unit of B improves objective function by 23). We could also have pivoted on A and canceled out the rest of the left column, but a unit of A only produces $13...

Why pivot on the first equation?

- Preserves feasibility by ensuring RHS ≥ 0
- Minimum ratio rule: min{480/15, 160/4, 1190/20} = min{32, 40, 59.5} so move S_c out of basis first.

Now we move on and pivot to an adjacent extreme point by moving A into the basis and pivoting on the second row (based on ratios of 32/(1/3) = 96, 32/(8/3)= 12, and 550/(85/3) = 19.4). Starting from the existing tableau

\begin{bmatrix} 1/3 & 1 & 1/15 & 0 & 0 & 32 \ 8/3 & 0 & -4/15 & 1 & 0 & 32 \ 85/3 & 0 & -4/3 & 0 & 1 & 550 \ 16/3 & 0 & -23/15 & 0 & 0 & -736 \end

we now want to divide the second row by $8/3$ and do row operations to cancel out the remaining coefficients in the left column. This is getting tedious to do by hand, so it is time to code things up. Let's start by defining the function `update_tab()` to compute the updated version of the tableau given the existing tableau and the row and column indices of the desired pivot.

```python
def update_tab(old_tab,pi,pj):
    '''
    Update a linear programming tableau.

    Inputs:
        old_tab: numpy float array corr. to the tableau to be updated
        pi, pj: int row/column index of pivot

    Returns:
        tab: numpy float array corr. to update tableau
    '''
    m,n = old_tab.shape
    tab = old_tab.copy()
    p = tab[pi,pj]
    tab[pi] = 1/p * tab[pi]
    for i in range(m):
        mult = tab[i,pj]
        if i!=pi:
            tab[i] -= mult*tab[pi]
    return tab
```

The original tableau was:

$$\begin{bmatrix} 5 & 15 & 1 & 0 & 0 & 480 \\ 4 & 4 & 0 & 1 & 0 & 160 \\ 35 & 20 & 0 & 0 & 1 & 1190 \\ 13 & 23 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now let's create the corresponding numpy array so we can let python compute the desired updates.

```
np.set_printoptions(precision=2, suppress=True) #set formatting so we can read entrie
tab0 = 1.0*np.array([[5,15,1,0,0,480],[4,4,0,1,0,160],[35,20,0,0,1,1190],[13,23,0,0,0
tab0
```

```
array([[ 5., 15., 1., 0., 0., 480.], [ 4., 4., 0., 1., 0., 160.], [ 35., 20., 0., 0., 1., 1190.], [ 13., 23., 0., 0., 0., 0.]])
```

As we did above, shift B into the basis by pivoting at position [0,1]. Assign `tab1` as name referring to the updated tableau.

```
tab1 = update_tab(tab0, 0, 1)
tab1
```

```
array([[ 0.33, 1. , 0.07, 0. , 0. , 32. ], [ 2.67, 0. , -0.27, 1. , 0. , 32. ], [ 28.33, 0. , -1.33, 0. , 1. , 550. ], [ 5.33, 0. , -1.53, 0. , 0. , -736. ]])
```

Confirming the initial step we did by hand, we obtain the feasible solution

$$A = S_c = 0, B = 32, S_h = 32, S_m = 550 \text{ with profit } Z = 736.$$

Now we are ready to execute our plan for the next step: Introduce A into the basis by pivoting about position [1,0].

We update the tableau using `update_tab` and assign the name `tab2` to store the result.

```
tab2 = update_tab(tab1, 1, 0)
tab2
```

```
array([[ 0. , 1. , 0.1 , -0.12, 0. , 28. ], [ 1. , 0. , -0.1 , 0.37, 0. , 12. ], [ 0. , 0. , 1.5 , -10.62, 1. , 210. ], [ 0. , 0. , -1. , -2. , 0. , -800. ]])
```

We now obtain a new feasible solution corresponding to

$$S_c = S_h = 0, A = 12, B = 28, S_m = 210 \text{ with profit } Z = 800.$$

Let's see if we can further improve the profit by introducing S_c into the basis by pivoting at position [2,2], assigning the name `tab3` to the updated tableau.

```
tab3 = update_tab(tab2, 2, 2)
tab3
```

array([[0. , 1. , 0. , 0.58, -0.07, 14.], [1. , 0. , 0. , -0.33, 0.07, 26.], [0. , 0. , 1. , -7.08, 0.67, 140.], [0. , 0. , 0. , -9.08, 0.67, -660.]])

We obtain a new feasible solution corresponding to

$$S_h = S_m = 0, A = 12, B = 28, S_c = 140$$

however the resulting profit is reduced to $Z = 660$.

We now see the previous solution (obtained from `tab2` with $Z = 800$) is **locally optimal** (by giving profit larger than that at neighboring vertices of the feasible region).

We can now also identify the `tab2` solution as the **global optimal solution** because (as discussed above) convexity implies that a local optimum is also a global optimum.

In fact, we could have concluded that the `tab2` solution was locally (and therefore globally) optimal solution before even computing the `tab3` update. To do that, we need to think about the bottom row of the tableau.

- The bottom row started by representing the profit as $Z = 13A + 24B$.
  - The coefficients correspond to the marginal increase in profit associated with an additional unit of each variable.
  - To increase the profit, that marginal profit must be positive.
- In `tab0`, there were 2 positive entries in the bottom row, specifically in the first 2 columns associated with A and B.
  - We could potentially increase profit by increasing either A or B by introducing the variable into the basis to enable a positive value.
  - We chose to introduce B, and produced `tab1` (with profit $Z = 736$).
- Bottom row of `tab1` has 1 positive entry in the initial column associated with A.

- We introduced A into the basis to produce `tab2` (with improved profit $Z = 800$).
- No positive entries remain in the bottom row of `tab2`.
  - It encodes the equation $Z = 800 - S_c - 2S_h$.
  - Anything we might try (including our test of introducing S_c into the basis) leads to a decrease in profit.

> Verify that the algebraic results agree with the geometric results obtained from the plots of constraints and objective function (profit) level sets shown above.

Section 5.5 Genetic Algorithms

Please read on your own if this is of interest.

Lagrange Multipliers

What if your objective function and/or constraints are nonlinear?

The usual approach involves introduction of **Lagrange multipliers**.

Here we consider constrained optimization with a nonlinear objective function and nonlinear equality constraints:

$$\begin{aligned}
\text{minimize} \qquad & f(x_0, x_1, \ldots, x_{n-1}) \\
\text{subject to} \qquad & g_0(x_0, x_1, \ldots, x_{n-1}) = 0 \\
& g_1(x_0, x_1, \ldots, x_{n-1}) = 0 \\
& \vdots \\
& g_{m-1}(x_0, x_1, \ldots, x_{n-1}) = 0
\end{aligned} \tag{5}$$

The **Lagrange multiplier method** converts the minimization problem to a multi-dimensional root-finding problem as follows:

- For each constraint $g_j = 0$, introduce a multiplier λ_j so the full set of $n+m$ coordinates becomes:

$$x_0, x_1, \ldots, x_{n-1}, \lambda_0, \lambda_1, \ldots, \lambda_{m-1}$$

- Form the Lagrangian:

$$L = f + \sum_{j=0}^{m-1} \lambda_j g_j$$

- Apply the first order optimality conditions to the Lagrangian:

$$\begin{aligned}
\frac{\partial L}{\partial x_0} &= 0 \\
\frac{\partial L}{\partial x_1} &= 0 \\
&\vdots \\
\frac{\partial L}{\partial x_{n-1}} &= 0 \\
\frac{\partial L}{\partial \lambda_0} &= 0 \\
\frac{\partial L}{\partial \lambda_1} &= 0 \\
&\vdots \\
\frac{\partial L}{\partial \lambda_{m-1}} &= 0
\end{aligned} \tag{6}$$

Simple example

Optimize

$$f(x_0, x_1) = x_0 + x_1$$

subject to the constraint

$$x_0^2 + x_1^2 - 2 = 0$$

Here we need a single multiplier λ and the Lagrangian is:

$$L(x_0, x_1, \lambda) = f + \lambda g = x_0 + x_1 + \lambda(x_0^2 + x_1^2 - 2)$$

We apply the first order conditions (set the partial derivatives to zero to obtain:

$$\begin{aligned} \frac{\partial L}{\partial x_0} &= 1 + 2\lambda x_0 &= 0 \\ \frac{\partial L}{\partial x_1} &= 1 + 2\lambda x_1 &= 0 \\ \frac{\partial L}{\partial \lambda} &= x_0^2 + x_1^2 - 2 &= 0 \end{aligned}$$

We end up having to solve a system of nonlinear equations (which can be challenging) but, in this case, we can readily obtain the solutions:

- First 2 equations $\implies x_0 = x_1 = -\frac{1}{2\lambda}$
- 3^{rd} equation $\implies x_0^2 = x_1^2 = 1$
- Solutions:

$$\lambda = -1/2, x_0 = x_1 = 1$$

$$\lambda = 1/2, x_0 = x_1 = -1$$

These are the critical points. By evaluating the function, we can see that the first solution is the maximum and the second is the minimum.

You can also tell this from the second order condition involving the Hessians:

$$H(f + \lambda g) = H(f) + \lambda H(g) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \lambda \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 2\lambda & 0 \\ 0 & 2\lambda \end{bmatrix} \tag{7}$$

Is the Hessian positive/negative definite (indicating a minimum/maximum) for arbitrary displacement $d = (d_0, d_1)$ satisfying the constraint at the critical point.

Focused on behavior at the critical point, we can linearize the constraint nearby:

$$g(x_0 + d_0, x_1 + d_1) = g(x_0, x_1) + \nabla g(x_0, x_1)^T d + \ldots = 0$$

$$g(x + d) = 0 \rightarrow \nabla g(x_0, x_1)^T d = 0$$

For the first solution:

$$\nabla f^t d = \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = 2d_0 + 2d_1 = \alpha \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{8}$$

The quadratic form becomes:

$$d^T H(L) d = \alpha^2 \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -2\alpha^2 \leq 0 \tag{9}$$

The quadratic form describing the function near the critical point is negative definite, and the first solution is confirmed as the maximum.

At the second critical point, things go pretty much the same, but $\lambda = 1/2$ and the Hession has 1 on the diagonal instead of -1:

$$d^T H(L) d = \alpha^2 \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 2\alpha^2 \leq 0 \tag{10}$$

The quadratic form describing the function near the critical point is positive definite, and the second solution is confirmed as the maximum.

A more interesting example: Find r and h to minimize

$$f = -\pi r^2 h$$

subject to the constraint

$$g = 2\pi r^2 + 2\pi r h - A = 0$$

Find the shape of a cylinder that maximizes the volume for a given surface area.

$$Lf + \lambda g = -\pi r^2 h + \lambda \left(2\pi r^2 + 2\pi r h - A\right)$$

$$\begin{aligned}
\frac{\partial L}{\partial r} &= -2\pi r h + 4\pi r \lambda + 2\pi h \lambda & &= 0 \\
\frac{\partial L}{\partial h} &= -\pi r^2 + 2\pi r \lambda & &= 0 \rightarrow \lambda = \frac{\pi r^2}{2\pi r} = r/2 \\
\frac{\partial L}{\partial \lambda} &= 2\pi r^2 + 2\pi r h - A & &= 0
\end{aligned}$$

Plugging in gives: $2r = h = \sqrt{\frac{2A}{3\pi}}$

Do you think this is a max or a min?