Lecture 9: Basic Optimization

- Optimization methods are prevalent (盛行的) throughout a vast range of applications. In its simplest form, **an optimization routine simply attempts to find the maximum or minimum of a real-valued function**, i.e. the objective function, by developing an algorithm that systematically (系统地，有条理地) chooses input values from an allowed set (允许数据集), or feasible set (可行数据集), and computes the value of the function.
- Typically, the optimization algorithm is built upon an iteration scheme that continues to choose new input values so that the maximum or minimum of the objective function is achieved.
- More generally, optimization includes finding best available values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains.

Unconstrained Optimization (Derivative-Free Methods)

The concept of optimization is fairly simple: **find the minimum or maximum of a function**. In optimization, the function is real-valued and called the objective function. Moreover, it is often the case that you would like to find the maximum or minimum under some constraints on the input values to the function. Thus there is a concept of **feasibility of the solution (可行解)**.

To begin, we will consider optimization without constraints, thus the aptly (恰当地) named **unconstrained optimization problem (无约束优化)**. With a background in calculus (微积分), the mention of minimization or maximization of a function automatically evokes the concept of the derivative, i.e. setting the derivative of a function to zero implies it is either a minimum of maximum.

In this section, however, **derivative-free(自由导数) methods for computing the minimum or maximum will be considered**. Such methods are often employed when computing the derivatives is either very expensive computationally, or simply intractable (难解) in practice.

Before proceeding forward with the development of the methods, abstracting the unconstrained optimization problem is necessary. Thus consider minimizing the objective function

$$\min\ f(x)$$

where the objective function depends generally on a number of variables

$$x_1, x_2, \ldots, x_n$$

These are called the control variables because we can choose their values. Indeed, our objective is to choose these values so that the objective function is minimized.

By definition, *f(x)* has a minimum at a point $x = x_0$ in a region *R* if

$$f(x) \geq f(x_0) \quad \text{for} \quad x \in R$$

A maximum can be defined in a similar way. However, a maximum of *f(x)* becomes a minimum of $-f(x)$. Thus it suffices from here forward to only consider the minimization problem.

Note that in this definition of the minimum, **it is not specified whether this is a local or global minimum**. For highly nontrivial (高度非平凡) functions *f(x)*, it may be that there exists many local minima, thus a search algorithm that converges to a minimum solution may not be the actual desired solution. This will be illustrated shortly in practice.

For simplicity, we will consider **derivative-free optimization methods for a function of a single variable** (min f(x)).

Golden section search（黄金分割搜索）

Given an interval in which the minimum is known to exist, **the golden section search algorithm** is an efficient method for finding the minimum of the function. To be more specific, **the function must be unimodal (单峰的) in the interval** $x \in [a, b]$, meaning there is only a single relative minimum in the interval chosen.

Specifically, within the interval where the minimum exists, two points (x_1 and x_2) are chosen so that $a < x_1 < x_2 < b$. Evaluation of the function *f(x)* is then performed at x_1 and x_2. The following observations hold:

$$\begin{aligned} &\text{if } f(x_1) \leq f(x_2) \quad \text{then return interval } x \in [a, x_2] \\ &\text{if } f(x_1) > f(x_2) \quad \text{then return interval } x \in [x_1, b] \end{aligned}$$

This ensures that the minimum is now in a smaller subinterval of the original $x \in [a, b]$.

The golden section search gives a simple procedure for selecting the evaluation points x_1 and x_2. Specifically, they are chosen so that (i) we want a constant (常数) reduction factor, say c, for the size of the interval, and (ii) they can be reused at the next iteration, thus saving a function evaluation. From the observations above, criterion (i) is enforced mathematically with the conditions:

$$c = \frac{x_2 - a}{b - a} \quad \rightarrow \quad x_2 = (1 - c)a + cb$$

$$c = \frac{b - x_1}{b - a} \quad \rightarrow \quad x_1 = ca + (1 - c)b$$

where c is the reduction factor to the new intervals $x \in [a, x_2]$ and $x \in [x_1, b]$, respectively

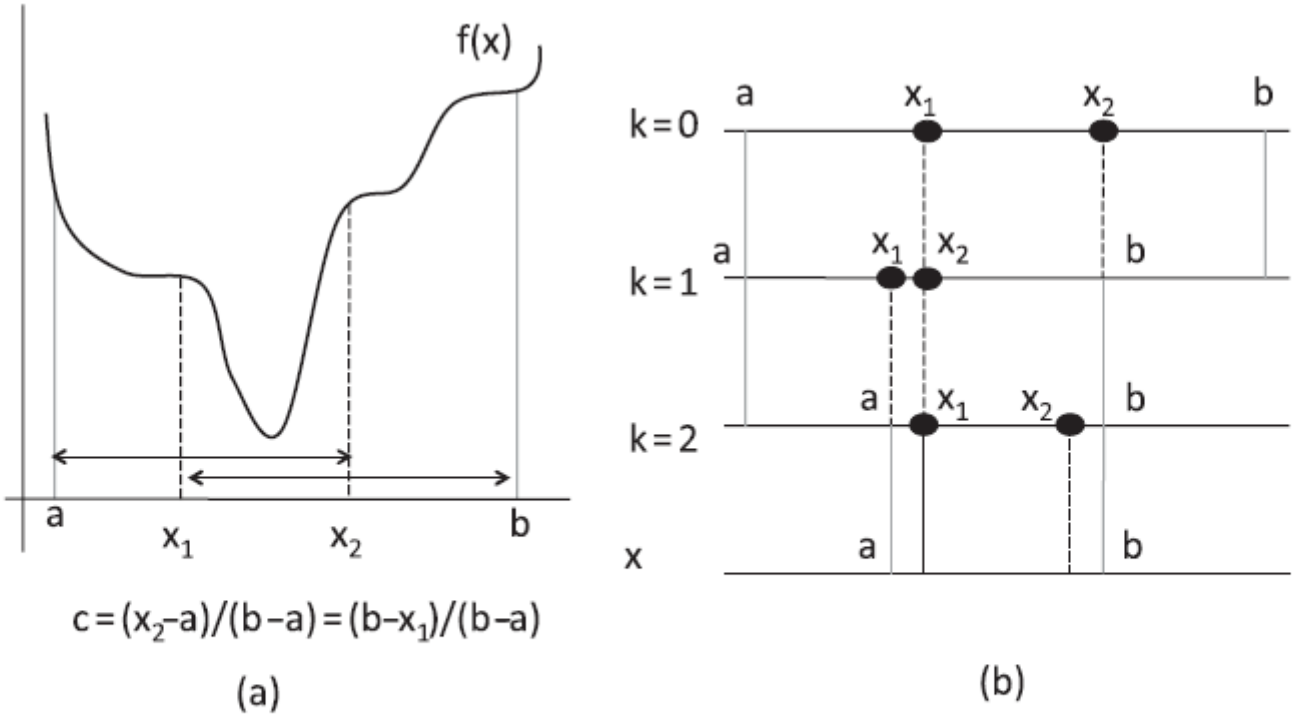

Figure 5.1: Graphical depiction of the golden section search. (a) Two points, x_1 and x_2, are used for the iteration process. The points are chosen so as to keep the ratio of $(b - x_1)/(b - a)$ and $(x_2 - a)/(b - a)$ the same. (b) The first three iterations are depicted.

The key observation is the following: if $f(x_1) < f(x_2)$ then the new interval is $x \in [a, x_2]$. Moreover, a new x_1 must be evaluated while the old x_2 is the x_1. Thus we have the following:

$$x_2^{\text{new}} = x_1^{\text{old}} = ca + (1-c)b$$
$$x_2^{\text{new}} = (1-c)a + cx_2^{\text{old}} = (1-c)a + c[(1-c)a + cb].$$

But these two new values of x_2 must be equivalent, thus we have

$$ca + (1-c)b = (1-c)a + c[(1-c)a + cb]$$
$$c^2(b-a) + c(b-a) - (b-a) = 0$$
$$c^2 + c - 1 = 0$$
$$c = (-1+\sqrt{5})/2 \approx 0.6180$$

where only the positive root of c is kept since c must be positive. Note that the resulting ratio is simply unity minus the golden ratio, i.e. $\varphi = 1 + c$, thus the name golden section search. Now that c is determined, x_1 and x_2 can be found. Such a routine can be easily implemented in MATLAB.

The following finds the minimum of the function $f(x) = x^4 + 10x\sin(x^2)$ on the interval $x \in [-2, 1]$.

```
a=-2;b=1; %initial interval
c=(-1+sqrt(5))/2; %golden section

x1=c*a+(1-c)*b;
x2=(1-c)*a+c*b;
f1=x1.^4+10*x1.*s1n(x1.^2);
f2=x2.^4+10*x2.*sin(x2.^2);

forj=1:100
  if f1<f2 move right boundary
    b=x2;x2=x1;f2=f1;
    x1=c*a+(1-c)*b;
    f1=x1.^4+10*x1.*sin(x1.^2);
  else %move left boundary
    a=x1;x1=x2;f1=f2;
    x2=(1-c)*a+c*bi
    f2=x2.^4+10*x2.*s1n(x2.^2);
  end
  if (b-a)<10(-6)&break if close
    break
  end
end
```

The above algorithm converges in 31 iterations to the minimum $f = -10.0882$ at $x = -1.2742$ with an accuracy of 10^{-6}. A theorem regarding the golden search algorithm states that after k iterations, starting from the interval $x \in [a, b]$, the midpoint of this final interval is within $c^k(b - a)/2$ of the minimum. Thus a guaranteed convergence rate can be established.

Successive parabolic interpolation（连续抛物线插值）

In the golden section search, no information was used about the values of $f(x_1)$ and $f(x_2)$ in selecting a new subinterval. Thus if $f(x_1) \ll f(x_2)$, it would be judicious (明智的) to assume that the **minimum might be closer to the point** x_1 than x_2 and the interval should cut accordingly. A technique that makes use of the function evaluation in choosing how to refine the interval is the method of **successive parabolic interpolation**.

The idea behind successive parabolic interpolation is to choose three points near the vicinity (附近) of the minimum: x_1, x_2 and x_3. These points correspond to left, middle and right points, respectively. With the choice of these three points, one can evaluate the function values associated with each point, $f(x_1)$, $f(x_2)$ and $f(x_3)$ and fit a parabola through them. Using the Lagrange polynomial coefficients, this gives a **parabolic function**

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

where *p(x)* is now locally approximating the function *f(x)*. The minimum of the parabola now serves as a temporary proxy for the minimum of the actual function *f(x)*. The minimum of the parabola can be found by setting the first derivative to zero so that we evaluate $p'(x_0) = 0$. This gives, after some algebra, the following minimum:

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

The idea now is to use this new point x_0 as our new middle point x_2. There are two cases of interest:

$$\begin{aligned}
x_0 < x_2: \quad & x_1^{\text{new}} = x_1^{\text{old}} \\
& x_2^{\text{new}} = x_0 \\
& x_3^{\text{new}} = x_2^{\text{old}} \\
x_0 > x_2: \quad & x_1^{\text{new}} = x_2^{\text{old}} \\
& x_2^{\text{new}} = x_0 \\
& x_3^{\text{new}} = x_3^{old}
\end{aligned}$$

This gives a simple algorithm that progressively (逐步的) converges to the minimum by using information about the function values. Moreover, it only requires a single function evaluation per iterative step.

Figure 5.2 gives a graphical depiction of this local iteration process. The convergence is typically extremely fast once you can find a good neighborhood to work in. However, the method is not guaranteed to converge, unfortunately. Thus great care should be used with this method. Alternatively, very good starting points must always be used.

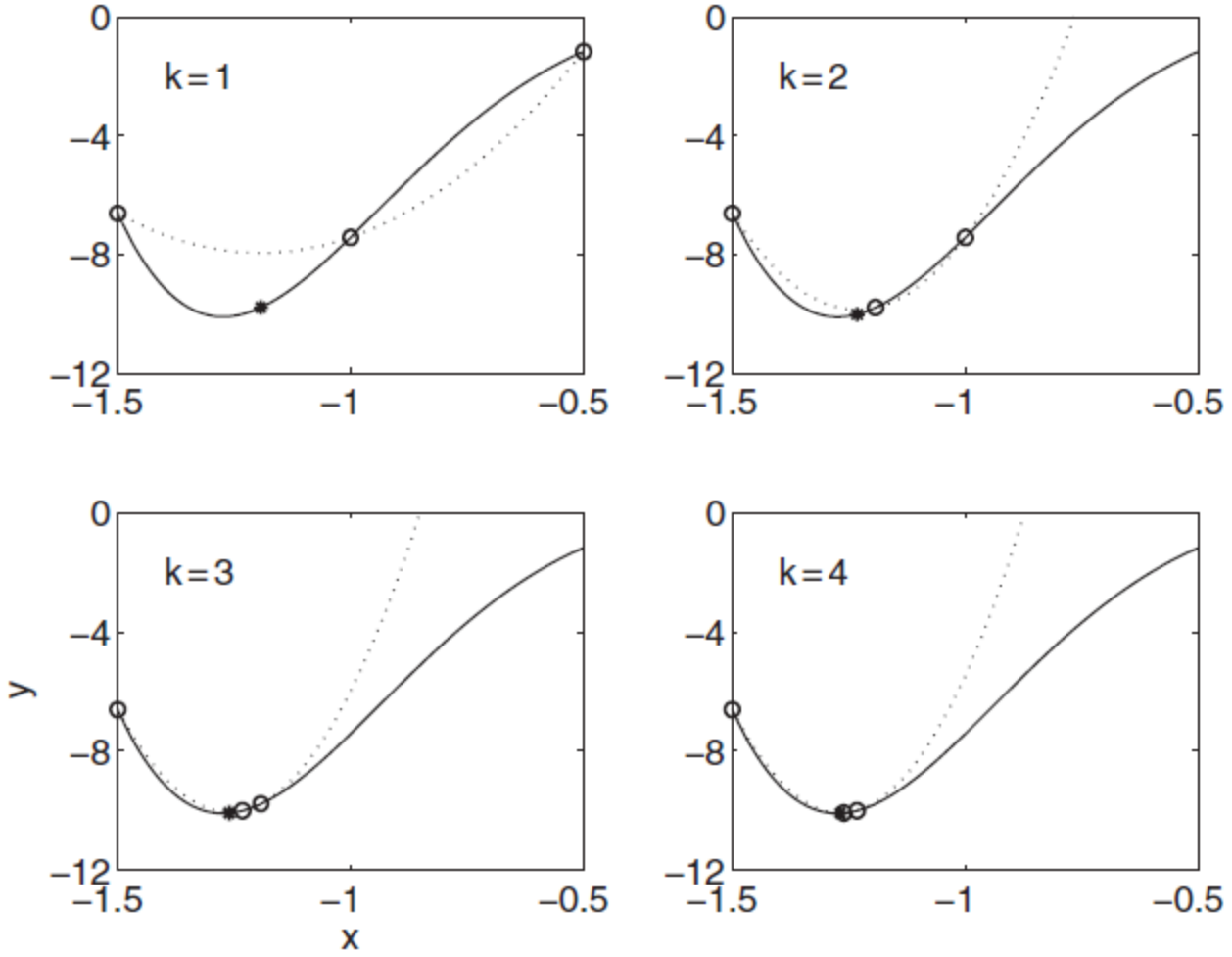

Figure 5.2: Graphical depiction of the successive parabolic interpolation algorithm. The three data points are depicted (circles) along with the point evaluated at the minimum of the parabola (star). The solid line is the function and the dotted line is the parabola generated. In this case, the guesses give a rapid convergence (14 iterations for 10^{-6} accuracy) to the minimum. However, the method is not guaranteed to converge and can, in fact, easily diverge. Thus it is important to have a good local starting point.

The following code implements the successive parabolic approximation.

```
x1=-1.5;x2=-1;x3=-.5; %initial guesses
f1=x1.^4+10*x1.*sin(x1.^2);
f2=x2.^4+10*x2.*sin(x2.^2);
f3=x3.^4+10*x3.*sin(x3.^2);
for j=1:100
   x0=(x1+x2)/2-((f2-f1)*(x3-x1)*(x3-x2))...
        /(2*((x2-x1)*(f3-f2)-(f2-f1)*(x3-x2)));
   if x0>x2
      x1=X2; f1=f2;
      x2=x0; f2=x0.^4+10*x0.*sin(x0.^2);
   else
      x3=x2; f3=f2;
      x2=x0; f2=x0.^4+10*x0.*sin(x0.^2);
   end
   if abs(x2-x3)<10(-6) |abs(x2-x1)<10^(-6)
      break
   end
end
```

This algorithm converges to the solution is less than half the iterations of the golden section search. However, it is easy to show that **if the initial guesses are changed, then the minimization will simply not work**.

fminbnd

MATLAB has a built-in one-dimensional search algorithm where the function and the interval are specified. The function *fminbnd* is based upon a combination of the golden section search and successive parabolic search. Integrated together they form an effective technique for finding minima. The following code gives an example of how to execute the function:

```
x=fminbnd('x2*cos (x)',3,4)
```

Here the left and right values of the search interval are given by $x = 3$ and $x = 4$, respectively. In this case, the function $f(x) = x^2 \cos(x)$ was found to have a minimum at $x = 3.6436$.