In many optimization problems, especially in the multidimensional case, the objective function may have many local extrema and may not be smooth. The simulated annealing algorithm is suitable for *Global Optimization* i.e., it is able to move through local extrema and recognize when global optimum located. Origin of the method is the *Metropolis’s Algorithm*; actually the Metropolis algorithm was first devised as a method to implement simulated annealing. It is thus a *probabilistic method* for optimization and belongs to the class of *stochastic optimization* methods. It does not require calculation of derivatives, and thus be considered as a derivative-free method.

The name of the algorithm comes from the process of annealing metals i.e. heating to a high temperature and then cooling slowly in order to strengthen them by changing the crystalline structure, by allowing to move it to a lower energy state, which might not occur if cooled too fast. In the original description the algorithm it was described in terms of a minimization problem associated with an objective function \( \phi() \). If \( \phi() \) is unimodal then most algorithms available for for optimization would suffice; for instance, Newton-Raphson with a good starting point would work well. However, if \( \phi() \) has one or more local minima in addition to global one, then these algorithms might terminate without finding the global minimum. Originally, the algorithm was presented as follows:

1. Choose an initial point. If it cannot be specified choose randomly.
2. Compute \( \phi_0 \), the initial value of the function.
3. Starting from the current point choose a random point on the unit \( n \)-dimensional sphere, where \( n \) is the dimensionality of the problem. This specifies a random direction.
4. Take a step of size \( \Delta r \) (depends on the objective function and the desired accuracy; its determination requires some experimentation.)
5. At the end-point of the step, compute new value of the function \( \phi_1 \) and accept the step with probability 1 if \( \Delta \phi = \phi_1 - \phi_0 \leq 0 \). If \( \Delta \phi > 0 \), accept the step with probability

\[
p = \exp (-\beta \Delta \phi)
\]

where \( \beta \) is a positive parameter.
6. If the step is not accepted go to 3. If step accepted set new starting point and go to 3.

Before discussing later generalizations of the simulated annealing algorithm, in particular, the introduction of an *annealing or cooling schedule*, note the following points:

- The beneficial steps (\( \phi_1 \leq \phi_0 \)) are accepted unconditionally and the detrimental steps are accepted according to an auxiliary experiment:
  1. Generate \( u \sim U(0, 1) \).
  2. If \( u < p \) where \( p = \exp(-\beta \Delta \phi) \), then the step is accepted.
  3. Otherwise try new direction.
• Probability of accepting a step depends on the size of the increment $\Delta \phi$: larger the increment smaller the probability of acceptance.

• Probability of accepting a detrimental step, however, is positive. So walking out of local (and
global) mimina is possible.

• The parameter $\beta$ determines the number of steps required and depends of the objective function.
  Usually select $\beta$ s.t. $0.5 < \exp(-\beta \Delta \phi) < 0.9$ to avoid too many acceptances (if closer to 1) or too
  many function evaluations (of $< 0.5$). In the above description this is kept constant.

• $\Delta r$ should be such that it allows escape from local minima in a few (2–3) steps.

• It is clear from studying the algorithm carefully that it is exactly the same as the Metropolis
  algorithm if $\beta$ is taken to be the constant 1.

Thus from the point of view of considering this algorithm as a special case of the Metropolis algorithm,
it is presented here using the notation we have already developed in discussing the Metropolis algorithm.
We denote the objective function again as $\phi()$ First define the acceptance probability $\alpha(x, y)$ as

$$
\alpha(x, y) = \exp \left\{ \frac{\log(\phi(y)) - \log(\phi(x))}{T_t} \right\}
= \left( \frac{\phi(y)}{\phi(x)} \right)^{\frac{1}{T_t}}
$$

Also choose a proposal disribution (candidate generating density) $q(x_t, \cdot)$. The temperature parameter
$T_t$ will be described shortly.

**Simulated Annealing Algorithm**

**Step 0.** Set $t = 0$ and choose starting value $x_0$

**Step 1.** Generate $y$ from $q(x_t, \cdot)$

**Step 2.** If $\phi(y) < \phi(x_t)$ then
  Set $x_{t+1} = y$
  Else
  Generate $u$ from $U(0, 1)$
  If $u \leq \alpha(x_t, y)$
    Set $x_{t+1} = y$
  Else
    Set $x_{t+1} = x_t$

**Step 3.** Set $t = t + 1$, go to Step 1

**Step 4.** Return $\{x_0, x_1, \ldots, x_N\}$

Thus it clear that for $T_t$ equal to 1, the acceptance probability is the same as that of the Metropolis
algorithm and thus, in that case Simulated Annealing and Metropolis algorithms are identical.
Assume that the parameter space is $p$-dimensional and that we are considering a minimization problem. Generally, a simulated annealing algorithm is implemented as follows:

**Step 0.** Begin with an initial temperature of $T_0$, a starting parameter value $\theta_0$, and calculate the function value at $\theta_0$, $\phi_{old}$.

**Step 1.** Randomly select a direction in $p$-dimensional space and take step in that direction to obtain the new parameter value, $\theta_1$ one co-ordinate is changed randomly and the step length is pre-determined by experimentation. Calculate the function value at $\theta_1$, $\phi_{new}$.

**Step 2.** If $\Delta = \phi_{new} - \phi_{old} < 0$ then move to $\theta_1$; else generate $u$ from $U(0, 1)$ and move to $\theta_1$ only if $u < \exp(-\Delta/T)$

**Step 3.** Repeat Steps 1-2 $N$ times until the system is deemed to be in equilibrium. This may result in $s$ successful moves.

**Step 4.** If $s > 0$ the temperature is lowered usually by letting $T$ decrease to $\rho T$ where $0 \leq \rho \leq 1$, which effectively decreases the probability of a success. The set values of $T$ defines the annealing schedule. If $s = 0$ the system is said to have reached equilibrium for a given temperature.

The values of $N$, $\rho$, and $T_0$ control the speed of convergence and can be usually found by experimentation. They are highly problem dependent. $N$ is usually in the hundreds and a large $N$ gives a more accurate solution but naturally requires more time. Increasing $\rho$ increases the reliability of the algorithm to reach the global minimum and corresponds to a slower cooling rate. It is usually in the range of .95. $T_0$ is chosen to be large enough for every point in the parameter space to have a chance of being visited, but not too large so that the algorithm moves out of the “molten” state quickly enough.

**One-dimensional Example**

Consider the Cauchy density:

$$f(x) = \frac{\beta}{\pi \{\beta^2 + (x-\alpha)^2\}}$$

for $-\infty < x < \infty$, $\beta \geq 0$, $-\infty < \alpha < \infty$. The log-likelihood surface has been shown to give various iterative methods considerable difficulty in the problem of maximum likelihood estimation of $\alpha$ for fixed values of $\beta$. For a random sample $x_1, x_2, \ldots, x_n$, the log-likelihood is

$$\ell = n \log \beta - \sum_{i=1}^{n} \log \{\beta^2 + (x_i - \alpha)^2\} - n \log \pi$$

With $\beta$ fixed, the maximum likelihood estimate for $\alpha$ minimizes

$$\ell(\alpha) = \sum_{i=1}^{n} \log \{\beta^2 + (x_i - \alpha)^2\}$$

For $\beta = 0.1$ and for a random sample $(-4.20, -2.85, -2.30, -1.02, 0.70, 0.98, 2.72, 3.50)$ the log-likelihood is plotted in Figure 1.
anneal<-function(fun,N,rho,t,theta)
{
  y=NULL
  theta.old=theta
  repeat{
    s=0
    for(k in 1:N)
    {
      f.old=fun(theta.old)
      theta.new=runif(1,-1,1)+theta.old
      f.new=fun(theta.new)
      if(f.new<f.old){
        theta.star=theta.new
        s=s+1
      }else{
        u = runif( 1)
        b = ( u <= exp(-(f.new-f.old)/t))
        theta.star = theta.new * b + theta.old * ( 1 - b )
        s=s+b
      }
    }
    theta.old= theta.star
    y=c(y,theta.star)
  }
  write(y, file = "anneal.result", ncol = 1, append = T)
  y=NULL
  if (s==0) break
  t=rho*t
  if(t <.1) break
  return(y)
}

ciauchy.1=function(alpha){
  x=c(-4.20,-2.85,-2.30,-1.02,0.70,0.98,2.72,3.50)
  return(sum(log(.01 + (x-alpha)^2)))
}

> x=seq(-6,6,.300)
> y=sapply(as.list(x),cauchy.1)
> plot(x,y,type="l",ylab="cauchy.1(beta=.1,alpha)",xlab="alpha")
> anneal.result=anneal(cauchy.1,300,.95,10,4)
> x=scan("anneal.result")
> plot(1:length(x),x,type="l"