目录

湖南工商大学 算法导论 课程教案

授课题目（教学章、节或主题）　　课时安排: 2学时

第十五讲：遗传算法与模拟退火算法　　授课时间 :第十五周周一第1、2节

教学内容（包括基本内容、重点、难点）：

基本内容： （1）蒙特卡洛算法：算法的思想；

（2）遗传算法：算法设计思路，算法流程，程序实现；

（3）模拟退火算法：算法设计思路，算法流程，程序实现；

（4）举例说明.

教学重点、难点：重点为遗传算法和模拟退火算法的原理、算法的程序设计

教学媒体的选择：本章使用大数据分析软件Jupyter教学，Jupyter集课件、Python程序运行、HTML网页制作、Pdf文档生成、Latex文档编译于一身，是算法导论课程教学的最佳选择。

板书设计：黑板分为上下两块，第一块基本定义，推导证明以及例子放在第二块。第一块 整个课堂不擦洗，以便学生随时看到算法流程图以及基本算法理论等内容。

课程过程设计： （1）讲解基本算法理论； （2）举例说明； （3）程序设计与编译； （4）对本课堂进行总 结、讨论； （5）布置作业与实验报告

第十五讲 遗传算法与模拟退火算法

一. 蒙特卡洛方法

蒙特卡洛方法（Monte Carlo method），也称统计模拟方法，是二十世纪四十年代中期由于科学技术的发展和电子计算机的发明，而被提出的一种以概率统计理论为指导的一类非常重要的数值计算方法。是指使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。与它对应的是确定性算法。蒙特卡洛方法在金融工程学，宏观经济学，计算物理学（如粒子输运计算、量子热力学计算、空气动力学计算）等领域应用广泛。

例 1：用蒙特卡洛方法求定积分 $\int_0^1 x^2 dx$

思想：定积分面积 = 阴影部分的随机数与正方形面积中总的随机数之比。

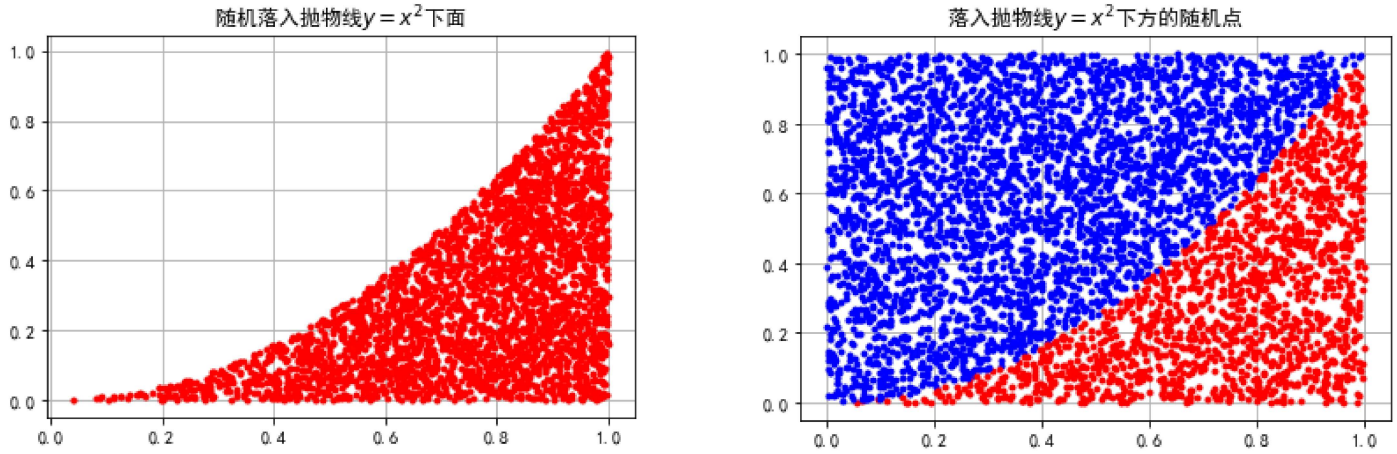


图 1: 蒙特卡洛算法求解定积分

python代码如下：

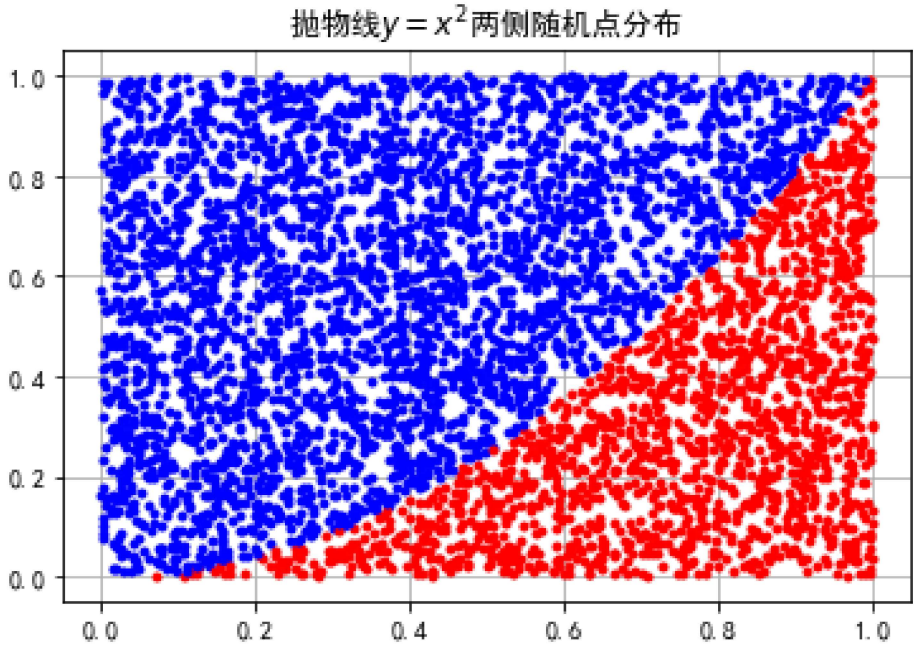
In [4]:

```
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
plt.rcParams['axes.unicode_minus']=False     #解决负数坐标显示问题

N=5000
x=np.random.random(N)
y=np.random.random(N)
a=0
for i in range(N):
    if y[i]<=x[i]**2:
        plt.plot(x[i],y[i],'r.')
        a+=1
    if y[i]>x[i]**2:
        plt.plot(x[i],y[i],'b.')
print(r"定积分的结果为：",a/N)
plt.grid()
plt.title(r"抛物线$y=x^2$两侧随机点分布")
```

定积分的结果为： 0.3252

Out[4]:

Text(0.5, 1.0, '抛物线$y=x^2$两侧随机点分布')



Matlab代码如下：

```
staus=10;
for i=1:4            %4次模拟
point=staus.^i;       %模拟的随机点数
RandData=rand(2,point);
 %根据随机点数，产生随机的(x,y)散点
scatter(RandData(1,:),RandData(2,:))
Below=find(RandData(1,:).^2>RandData(2,:));
%寻找位于曲线下的散点
Outcome(i)=length(Below)/length(RandData);
 %最终结果的表示
end
BelowData=RandData(:,Below);
hold on
scatter(BelowData(1,:),BelowData(2,:))
%x轴为生成的随机数，y轴为随机数的平方
```

例 2： 利用蒙特卡洛方法解非线性规划问题

$$\min\ f(x)=\frac{(-2x_1^2)}{-x_2^2}+x_1x_2+8x_1+3x_2$$

$$\begin{cases}3x_1+x_2=10\\ x_1>0\\ x_2>0\end{cases}$$

Matlab代码：

```
%首先编写目标函数的m文件，mbfunc.m
function z=mbfunc(x)
z=2*x(1)^2+x(2)^2-x(1)*x(2)-8*x(1)-3*x(2);
%编写约束条件的m文件，limitst.m
function st=limitst(x)
if 3*x(1)+x(2)-10>=-0.01&&3*x(1)+x(2)-10<=0.01
    st=1;
else
    st=0;
end
end
clc
clear all
a=1;b=3;maxp=20000;
x1=unifrnd(a,b,maxp,1);
x2=unifrnd(a,b,maxp,1);
sol=[x1,x2];
z0=Inf;
j=0;
z=zeros(1,maxp);
for i=1:maxp
    st=limitst(sol(i,:));
    if st==1
        j=1;
        z(i)=mbfunc(sol(i,:));
 if z(i)<z0
            X=sol(i,:);
            z0=z(i);
        end
    end
end
if j==1
    Q=-z0;
    disp(['迭代产生的最优解：X=',num2str(X)])
    disp(['迭代产生的最优解：Q=',num2str(Q)])
else
    return
end
```

运行结果：

迭代产生的最优解：X=2.4706，X=2.5926

迭代产生的最优解：Q=15.0186

二. 遗传算法

1. 遗传算法（Genetic Algorithm,GA）

遗传算法是计算数学中用于解决最优化的搜索算法，是进化算法的一种。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等

遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个体）的抽象表示（称为染色体）的种群向更好的解进化。传统上，解用二进制表示（即0和1的串），但也可以用其他表示方法。进化从完全随机个体的种群开始，之后一代一代发生。在每一代中，整个种群的适应度被评价，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择和突变产生新的生命种群，该种群在算法的下一次迭代中成为当前种群。

2. 遗传算法的生物学基础

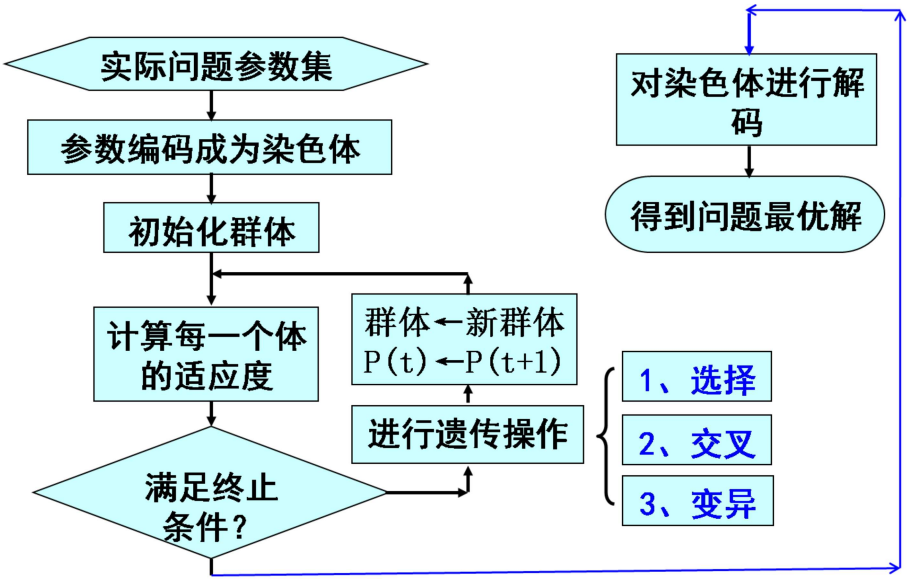
- 基因型(genotype)：性状染色体的内部表现；
- 表现型(phenotype)：染色体决定的性状的外部表现，或者说，根据基因型形成的个体的外部表现；
- 进化(evolution)：种群逐渐适应生存环境，品质不断得到改良。生物的进化是以种群的形式进行的。
- 适应度(fitness)：度量某个物种对于生存环境的适应程度。
- 选择(selection)：以一定的概率从种群中选择若干个个体。一般，选择过程是一种基于适应度的优胜劣汰的过程。
- 复制(reproduction)：细胞分裂时，遗传物质DNA通过复制而转移到新产生的细胞中，新细胞就继承了旧细胞的基因。
- 交叉(crossover)：两个染色体的某一相同位置处DNA被切断，前后两串分别交叉组合形成两个新的染色体。也称基因重组或杂交；
- 变异(mutation)：复制时可能（很小的概率）产生某些复制差错，变异产生新的染色体，表现出新的性状。
- 编码(coding)：DNA中遗传信息在一个长链上按一定的模式排列。遗传编码可看作从表现型到基因型的映射。
- 解码(decoding)：基因型到表现型的映射。
- 个体（individual）：指染色体带有特征的实体；
- 种群（population）：个体的集合，该集合内个体数称为种群

思想：优胜劣汰、适者生存；生物历经基因复制、交叉、变异的过程逐步进化。



图 2: 遗传算法生物学基础

3. 遗传算法的标准流程：



4. 遗传算法的基本概念

个体（Individual）：称 $s=0,1^l$ 为个体空间，个体空间的元素$a=a_0,a_1,\cdots a_{(i-1)}\in S$称为个体，它是染色体带有特征的实体。分量$a_j\in\{0,1\}$ 称为基因，正整数L称为个体的基因长度。

种群（Population）：称个体空间S中N个个体组成的一个子集（个体允许重复）称为一个种群，记为：

$$A=(A_1,A_2,\cdots,A_N)$$

其中 $A_j\,(j=1,2,\ldots,N)\in S$，$N$ 称为种群规模。

适应度（Fitness）：在研究自然界中生物的遗传和进化现象时，生物学家使用适用度这个术语来度量某个物种对于生存环境的适应程度。对生存环境适应程度较高的物种将获得更多的繁殖机会，而对生存环境适应程度较低的物种，其繁殖机会就会相对较少，甚至逐渐灭绝。在遗传算法中，一般通过适应度函数（Fitness function）来衡量某一个体的适应度高低。

编码（Coding）：将一个待求解的问题的实际可行解从其解空间转换到遗传算法所能处理的搜索空间（即个体空间）的过程，就称为编码。

解码（Decoding）：解码是将遗传算法所搜索到的最优个体的染色体转换成待求解问题都实际最优解的过程，即编码的逆过程。

选择操作（Selection）：根据各个个体的适应度，按照一定的规则，从第t代群体P(t)中选择出一些优良的个体遗传到下一代群体P(t+1)中。一般地，选择操作通过选择算子（Selection Operator）进行。

交叉操作（Crossover）：将群体P(t)内的各个个体随机搭配成对，对每一对个体，以某个概率（称为交叉概率，Crossover Rate）遵循某一种规则交换它们之间的部分染色体。

变异操作（Mutation）：对群体P(t)中的每一个个体，以某一概率以某一概率（称为变异概率，Mutation Rate）改变某一个或某一些基因座上的基因值为其他的等位基因。

5. 遗传算法的应用步骤

（1）确定决策变量及各种约束条件，即确定出个体的表现型X和问题的解空间。

（2）建立优化模型，确定出目标函数的类型及其数学描述形式或量化方法。

（3）确定表示可行解的染色体编码方法，即确定出个体的基因型X^*，及遗传算法的搜索空间。

- 标准遗传算法多采用二进制编码方法，将决策变量用二进制字符串表示，二进制编码串的长度由所求精度决定。然后将各决策变量的二进制编码串连接在一起，构成一个染色体。
- 例如：变量x的定义域为[-2，3]，要求其精度为10-5，则需要将[-2，3]分成至少500 000个等长小区域，而每个小区域用一个二进制串表示。于是有 $2^L = 500000$，即

$$\log_2^{500000} \approx 18.9316, \quad \delta = \frac{X_r - X_l}{2^L - 1}$$

向上取整，可得到L=19。即可用19位二进制串 $a_{18}\,a_{17}\ldots a_0$ 来表示。

（4）确定解码方法，即确定出由个体基因型 X^*，到个体表现型X的对应关系和转换方法。

例如：对于二进制编码，其解码过程如下：若X的取值范围为 $[X_l, X_r]$，参数的二进制编码码长为L，码串对应的十进制整数为k，则解码公式为:

$$K = \sum_{i=0}^{L-1} a_i 2^i, \qquad X = \frac{(X_r - X_l)k}{(2^L - 1)} + X_l$$

式中 $[X_l, X_r]$为参数最小、最大值； L 为参数编码长度； k 为二进制串对应的实数值。

（5）确定个体适应度的量化评价方法，就是确定出由目标函数值到个体适应度的转换规则。标准遗传算法的适应度函数常用一下三种：

- 直接以待求解的目标函数为适应度函数. 若目标函数为最大值问题，则 Fit(f(X))=f(X); 若目标函数为最小值问题，则 Fit(f(X))= - f(X)
- 界限构造法：若目标函数为最大值问题，则

$$Fit(f(X)) = \begin{cases} c_{\max} + f(x), & f(x) < c_{\max} \\ 0, & \text{其他} \end{cases}$$

C_{max} 为 $f(x)$ 的最大值估计。若目标函数为最小值问题，则

$$Fit(f(X)) = \begin{cases} f(x) - c_{\min}, & f(x) > c_{\min} \\ 0, & \text{其他} \end{cases}$$

C_{min} 为 $f(x)$ 的最小值估计。

- 倒数法：若目标函数为最小值问题，则

$$Fit(f(X) = \frac{1}{1 + c + f(X)}, \qquad c \geq 0,\ c + f(x) \geq 0.$$

若目标函数为最大值问题，则

$$Fit(f(X) = \frac{1}{1 + c - f(X)}, \qquad c \geq 0,\ c - f(x) \geq 0.$$

C 为目标函数界限的保守估计值。

（6）选择算子和选择操作

个体选择概率的常用分配方法有以下两种：

- 按比例的适应度分配（Proportional Fitness Assignment） 亦可称为选择的蒙特卡罗方法，是利用比例于各个个体适应度的概率决定其子孙的遗留可能性。若某个个体 i，其适应度为 f_i，则其被选取的概率表示为:

$$P_i = \frac{f_i}{\sum_{i=1}^{M} f_i}$$

显然选择概率大的个体，能多次被选中，它的遗传因子就会在种群中扩大。

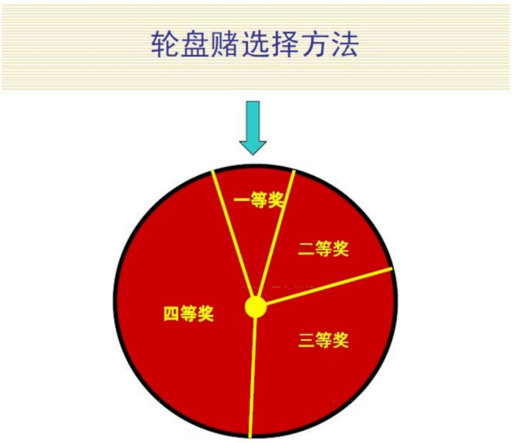
- 基于排序的适应度分配（Rank-based Fitness Assignment），在基于排序的适应度分配中，种群按目标值进行排序。适应度仅仅取决于个体在种群中的序位，而不是实际的目标值。排

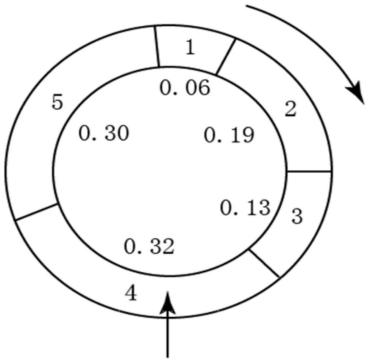
序方法比比例方法表现出更好的鲁棒性，它能在一定程度上克服了比例适应度计算的尺度问题和过早收敛问题。

轮盘赌选择法：

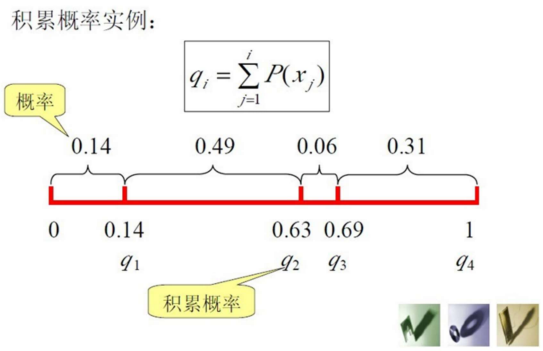
个体选择概率确定后，可以选用的常用选择算法有轮盘赌选择法（Roulette Wheel Selection）、随机遍历抽样法（Stochastic Universal Sampling）、局部选择法（Local Selection）、截断选择法（Truncation Selection）和锦标赛选择法（Tournament Selection）等。

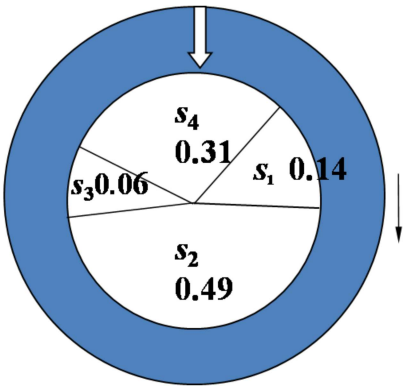
标准遗传算法常用的轮盘赌选择法的原理如下图所示：





轮盘赌选择示意图:





轮盘赌选择法可用如下过程模拟来实现：

(1)在［0, 1］内产生一个均匀分布的随机数 r。

(2)若 $r \leq q_1$,则染色体x1被选中。

(3)若 $q_{k-1} < r \leq q_k \ (2 \leq k \leq N)$, 则染色体 x_k 被选中。 其中的qi称为染色体 x_i (i=1, 2, …, n)的积累概率, 其计算公式为：

$$q_i = \sum_{j=1}^{i} p(Xj)$$

6. 遗传算法程序实现

遗传算法伪代码：

```
Procedure  GA
Begin
      initialize  P(0)
      t=0
      while (t<T)  do
           for  i=1  to  M   do
                Evaluate   fitness  of  P(t);
           end   for
           for  i=1  to  M   do
                Seclect  operation  of  P(t);
           end   for
           for  i=1  to  M/2   do
                  Crossover   operation   to  P(t);
           end  for
           for  i=1  to  M   do
                  Mutation   operation   to  P(t);
           end  for
           for  i=1  to  M  do
                  P(i+1) = P(t);
           end  for
           t = t+1
       end  while
end
```

遗传算法Python代码：

In [ ]:

```
# -*- coding: utf-8 -*-
#遗传算法python代码
import numpy

# 遗传算法求最大值
class GenericAlgorithm:
    # func为要求最大值的函数
    # encoding决定是否对自变量进行编码
    # min为args对应最小值
    # max为args对应最大值
    # dnaLength为dna编码长度
    def __init__(self, func, min, max, encoding = False, dnaLength = 128):
        self._func = func
        self._min = min
        self._max = max
        self._flag = encoding
        self._dnaLen = dnaLength

    def __del__(self):
        return

    # 编码，仅在交叉和变异时使用，此处用最简单的二进制编码
    def _encode(self, matrix, length):
        rows = matrix.shape[0]
        columns = matrix.shape[1]
        realen = int(length / columns)
        bmax = 2**(realen) - 1
        r = [self._max[i] - self._min[i] for i in range(len(self._min))]
        mkb = []
        for i in range(rows):
            tmp = ''
            for j in range(columns):
                bstr = str(bin(int((matrix[i][j] - self._min[j])/r[j] * bmax))
                bstr = bstr[2:].zfill(realen)
                tmp += bstr
            mkb.append(tmp)
        return mkb

    # 解码，同上
    def _decode(self, matrix, length):
        rows = len(matrix)
        columns = len(self._min)
        realen = int(length / columns)
        bmax = float(2**(realen) - 1)
        r = [self._max[i] - self._min[i] for i in range(len(self._min))]
        mk = numpy.zeros((rows, columns))
```

```
        for i in range(rows):
            for j in range(columns):
                tmp = matrix[i][j*realen:(j+1)*realen]
                mk[i][j] = int(tmp,2) / bmax * r[j] + self._min[j]
        return mk

    def _evolve(self, mk, prop_cross, prop_mut):
        mk = self._cross(mk, prop_cross)
        pk = self._mutation(mk, prop_mut)
        return pk

    # 交叉
    def _cross(self, mk, prop_cross):
        rows = mk.shape[0]
        columns = mk.shape[1]
        idx = numpy.zeros(1)
        while (idx.shape[0] % 2 != 0):
            idx = numpy.where(numpy.random.rand(rows,1) < prop_cross)[0]
        if (self._flag):
            mkb = self._encode(mk, self._dnaLen)
            for i in range(0, idx.shape[0], 2):
                bit = int(numpy.random.rand()*self._dnaLen)
                tmp1 = mkb[idx[i]][:bit] + mkb[idx[i+1]][bit:]
                tmp2 = mkb[idx[i+1]][:bit] + mkb[idx[i]][bit:]
                mkb[idx[i]] = tmp1
                mkb[idx[i+1]] = tmp2
            mk = mkb
        else:
            for i in range(0, idx.shape[0], 2):
                rn = numpy.random.rand()
                w = numpy.random.randn(1,columns)
                tmp1 = rn*mk[idx[i]] + (1-rn)*mk[idx[i+1]] + w
                tmp2 = (1-rn)*mk[idx[i]] + rn*mk[idx[i+1]] - w
                mk[idx[i]] = tmp1
                mk[idx[i+1]] = tmp2
            for i in range(rows):
                for j in range(columns):
                    mk[i][j] = numpy.clip(mk[i][j], self._min[j], self._max[j]
        return mk

    # 变异
    def _mutation(self, mk, prop_mut):
        if (self._flag):
            rows = len(mk)
            for i in range(rows):
                tmp = list(mk[i])
                idx = numpy.where(numpy.random.rand(self._dnaLen) < prop_mut)[
                for j in range(len(idx)):
```

```
                    if (tmp[idx[j]] == '0'):
                        tmp[idx[j]] = '1'
                    elif (tmp[idx[j]] == '1'):
                        tmp[idx[j]] = '0'
                    else:
                        print("just for extension.")
                mk[i] = ''.join(tmp)
            mk = self._decode(mk, self._dnaLen)
        else:
            rows = mk.shape[0]
            columns = mk.shape[1]
            idx = numpy.where(numpy.random.rand(rows,1) < prop_mut)[0]
            mk[idx] = mk[idx] + numpy.random.randn(idx.shape[0], columns)
            for i in range(rows):
                for j in range(columns):
                    mk[i][j] = numpy.clip(mk[i][j], self._min[j], self._max[j]
        return mk
```

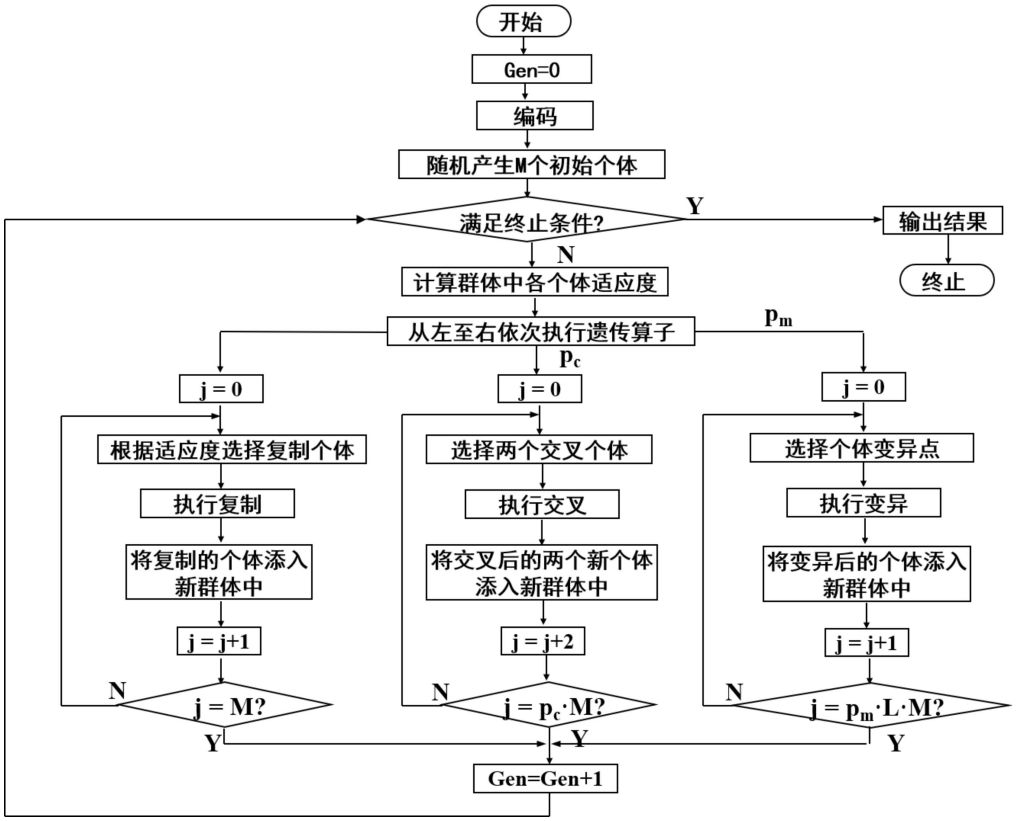
In [ ]:

```python
# 轮盘赌选择法python代码
    def _select(self, values, pk):
        indexes = []
        min = numpy.min(values)
        values -= min     # 将values的值转换为正值，方便下面的操作
        summation = numpy.sum(values)
        population = values.shape[0]
        data = [(pk[i], values[i]) for i in range(population)]
        sorteddata = sorted(data, key = lambda ele: ele[1])
        for i in range(population):
            test = numpy.random.rand()
            sum = 0.0
            for j in range(population):
                sum += sorteddata[j][1]
                if (sum >= test*summation):
                    indexes.append(j)
                    break
        mk = numpy.zeros(pk.shape)
        for i in range(population):
            idx = indexes[i]
            mk[i] = sorteddata[idx][0]
        return mk

    # iterations:  最大迭代次数
    # population:  种群数量
    # prop_cross:  交叉概率
    # prop_mut:    变异概率
    def run(self, iterations, population, prop_cross, prop_mut):
        columns = len(self._min)
        p0 = numpy.random.randn(population, columns)  # 初始种群，每行为一组输
        for i in range(population):
            for j in range(columns):
                p0[i][j] = numpy.clip(p0[i][j], self._min[j], self._max[j])
        values = numpy.zeros(population)
        pk = p0
        max = 0.0
        x_max = 0.0
        for iter in range(iterations):      # 开始迭代
            for i in range(population):
                values[i] = self._func(*pk[i])
            idx = numpy.argmax(values)
            x = pk[idx]
            y = values[idx]
            if (y > max):
                max = y
                x_max = x
```

```
            mk = self._select(values, pk)    # 选择mk
            pk = self._evolve(mk, prop_cross, prop_mut)  # 进化
        return (x_max, max)


# -*- coding: utf-8 -*-


def function(x, y):
    term1 = 3*numpy.power(1-x, 2)*numpy.exp(-numpy.power(x,2)-numpy.power(y+1,
    term2 = -10*(x/5 - numpy.power(x,3) - numpy.power(y,5))*numpy.exp(-numpy.
    term3 = -numpy.exp(-numpy.power(x+1,2)-numpy.power(y,2))/3
    return term1 + term2 + term3

ga = GenericAlgorithm(func=function, min=[-5,-5], max=[5,5],
                      encoding=True, dnaLength=32)
(x, y) = ga.run(500, 40, 1.00, 0.005)
print("GA: x = ", x, ", y = ", y)
xmax = (-0.0093, 1.5814)
ymax = function(xmax[0], xmax[1])
print("Real: x = ", xmax, ", y = ", ymax)
```

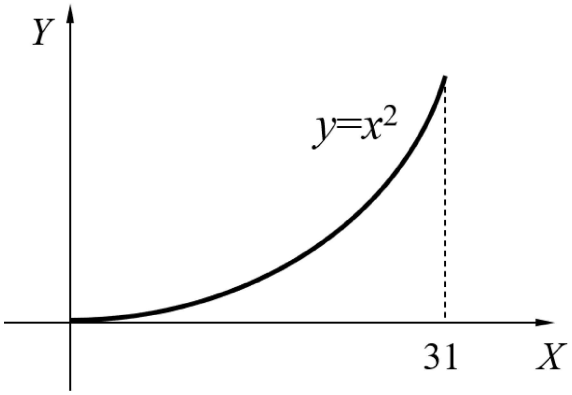
7. 遗传算法具体步骤

- 选择编码策略，把参数集合（可行解集合）转换染色体结构空间；
- 定义适应度函数，便于计算适应值；
- 确定遗传策略，包括选择群体大小，选择、交叉、变异方法以及确定交叉概率、变异概率等遗传参数；
- 随机产生初始化群体；
- 计算群体中的个体或染色体解码后的适应值；
- 按照遗传策略，运用选择、交叉和变异算子作用于群体，形成下一代群体；
- 判断群体性能是否满足某一指标，或者已完成预定的迭代次数，不满足则返回第五步，或者修改遗传策略再返回第六步.



8. 遗传算法举例：极值问题

例 3 利用遗传算法求解区间［0,31］上的二次函数 的最大值，精度要求达到个位。



求解$y=x^2$在区间[0,31]最大值:

原问题可转化为在区间［0, 31］中搜索能使y取最大值的点a的问题。那么，［0, 31］ 中的点x就是个体, 函数值f(x)恰好就可以作为x的适应度，区间［0, 31］就是一个(解)空间 。这样, 只要能给出个体x的适当染色体编码, 该问题就可以用遗传算法来解决。

(1) 设定种群规模,编码染色体，产生初始种群。

将种群规模设定为4；用5位二进制数编码染色体；取下列个体组成初始种群S1:

s1=13 (01101),　　s2=24 (11000)

s3=8 (01000),　　s4=19 (10011)

(2)定义适应度函数：$f(x)=x^2$

(3)计算各代种群中的各个体的适应度，并对其染色体进行遗传操作,直到适应度最高的个体(即31（11111）)出现为止。

首先计算种群S1中各个体

s1=13(01101),　　s2=24(11000)

s3=8 (01000),　　s4=19(10011)

的适应度f(si) 。

容易求得:

f(s1) = f(13) = 13^2 = 169

f(s2) = f(24) = 24^2 = 576

f(s3) = f(8) = 8^2 = 64

f(s4) = f(19) = 19^2 = 361

再计算种群S1中各个体的选择概率:

选择概率的计算公式为:

$p(x_i)=\frac{f(X_i)}{\sum_{j=1}^{n} f(X_j)}$

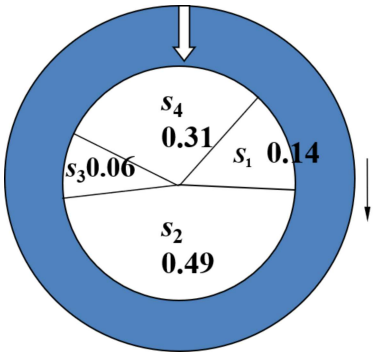
由此求得:

P(s1) = P(13) = 0.14

P(s2) = P(24) = 0.49

P(s3) = P(8) = 0.06

P(s4) = P(19) = 0.31



选择-复制：

染色体	适应度	选择概率	选中次数
S_1=01101	169	0.14	1
S_2=11000	576	0.49	2
S_3=01000	64	0.06	0
S_4=10011	361	0.31	1

经选择复制得群体：（淘汰了s3染色体）

s1’=11000（24），　s2’ =01101（13）

s3’=11000（24），　s4’ =10011（19）

交叉：设交叉率pc=100%，即S1中的全体染色体都参加交叉运算。 设s1’与s2’配对，s3’与s4’配对。分别交换后两位基因，得新染色体：

s1”=11001（25），　s2”=01100（12）

s3”=11011（27），　s4”=10000（16）

变异: 设变异率pm=0.001。这样，群体S1中共有 **5×4×0.001=0.02** 位基因可以变异。0.02位显然不足1位，所以本轮遗传操作不做变异。

于是，得到第二代种群S2：

s1=11001（25），　s2=01100（12）

s3=11011（27），　s4=10000（16）

第二代种群S2中各染色体的情况:

染色体	适应度	选择概率	选中次数
S_1=11001	625	0.36	1
S_2=01100	144	0.08	0
S_3=11011	729	0.41	2
S_4=10000	256	0.15	1

假设这一轮选择-复制操作中，种群S2中的4个染色体都被选中，则得到群体：

s1’=11001（25），　s2’= 11011（27）

s3’=11011（27），　s4’= 10000（16）

做交叉运算，让s1’与s4’，s2’与s3’分别交换后两位基因，得

s1”=11111（31），　　s2”=11100（28）

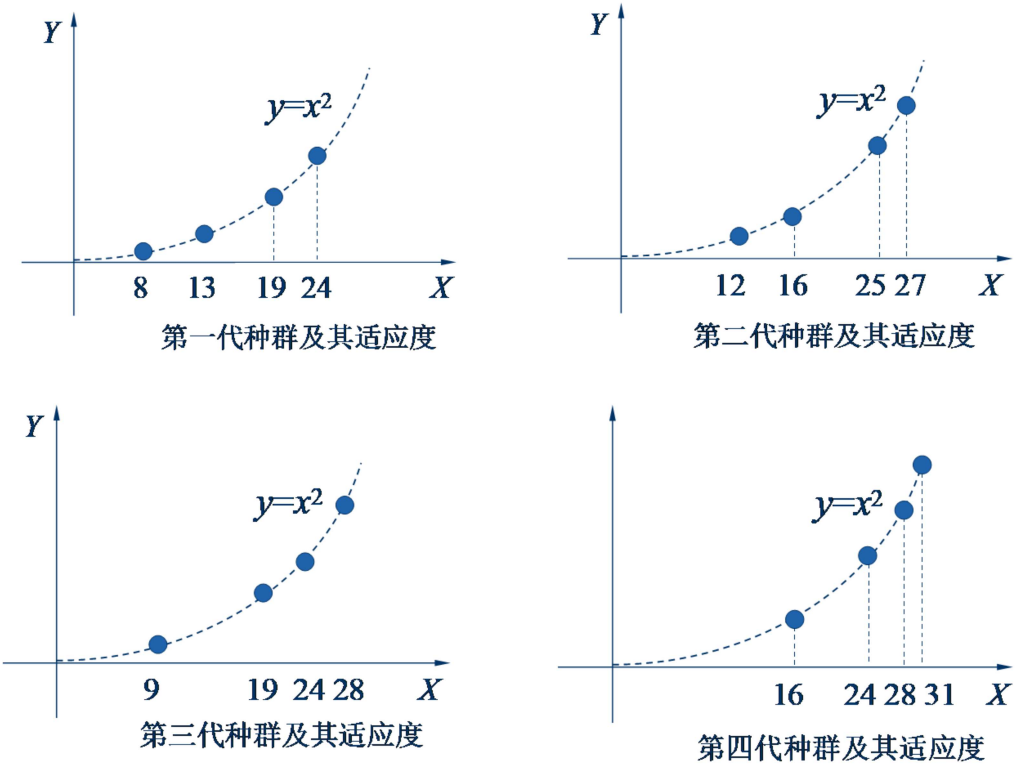
s3”=11000（24），　　s4”=10000（16）

这一轮仍然不会发生变异。 于是，得第四代种群S4：

s1=11111（31），　　s2=11100（28）

s3=11000（24），　　s4=10000（16）

显然，在这一代种群中已经出现了适应度最高的染色体s1=11111。于是，遗传操作终止，将染色体“11111”作为最终结果输出。然后，将染色体“11111”解码为表现型，即得所求的最优解：**31**。



遗传算法求解无约束优化问题

例 4：求解无约束问题

$$\max f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$$
$$\begin{cases} -3.0 \le x_1 \le 12.0, \\ 4 \le x_2 \le 5.8 \end{cases}$$

matlab代码:

```
#1. 首先，我们需要构建目标函数 fun1
function y = fun1(x)      %目标函数
y = 21.5+x(:,1).*sin(1*pi*x(:,1))+x(:,2).*sin(2*pi*x(:,2));
#2. 种群初始化函数
function pop = initialization(pop)
%种群初始化
pop.individual = rand(pop.size,2);
pop.individual(:,1) = pop.individual(:,1)*15-3;
pop.individual(:,2) = pop.individual(:,2)*2+4;
pop.solution = pop.individual;%种群的个体即为解
pop.obj = fun1(pop.solution);%计算目标函数值
n = find(pop.obj == max(pop.obj),1);%当前种群最优解下标
%记录最优二进制个体和实数解，以及最佳及平均目标函数值
pop.bestIndividual = pop.individual(n,:);
pop.bestSolution = pop.solution(n,:);
pop.perfom = [max(pop.obj),mean(pop.obj)];
#构建变异算子，同时为了防止在适应度评价过程中将最优解扔掉，我们选择每次评价之后直接将最优解放入到下一代中。
function pop = newPop(pop)%选择算子产生新种群
pop = selection(pop);%执行交叉和变异算子
pop = crossover(pop);
pop = mutation(pop);%适应度评价（计算目标函数值）
pop.solution = pop.individual;
pop.obj = fun1(pop.solution);
%随机将子代中某一个个体替换为最优解  即将最优解保存下来
t = ceil(rand*pop.size);
pop.individual(t,:) = pop.bestIndividual(end,:);
pop.solution(t,:) = pop.bestSolution(end,:);
pop.obj(t,:) = pop.perfom(end,1);
n = find(pop.obj == max(pop.obj),1);%当前种群最优解下标
%记录最优二进制个体和实数解，以及最佳及平均目标函数值pop.bestIndividual(end+1,:) = pop.individual(n,:);
pop.bestSolution(end+1,:) = pop.solution(n,:);
pop.perfom(end+1,:) = [max(pop.obj),mean(pop.obj)];
%-------------------------------------------------------
function pop = selection(pop)
%选择函数
p = pop.obj/sum(pop.obj);%计算生存概率
pp = cumsum(p);%计算累计概率
for i = 1:pop.size
    r = rand;%产生一个随机数
```

```
    k(i) = find(pp>=r,1);
%随机数落在哪个累积概率区间，即对应的个体被选中，记录下标
end
%把选中的个体放入新种群
pop.individual = pop.individual(k,:);
pop.solution = pop.solution(k,:);
pop.obj = pop.obj(k,:);
%-------------------------------------------------------------
function pop = crossover(pop)
%交叉算子
for i = 1:pop.size/2%遍历每一对个体
    if rand<pop.cr%依交叉概率执行交叉操作
        %提取待交叉个体
        a1 = pop.individual(2*(i-1)+1,:);
        a2 = pop.individual(2*i,:);
        aa = rand;
        pop.individual(2*(i-1)+1,:) = aa*a1+(1-aa)*a2;
        pop.individual(2*i,:) = aa*a2+(1-aa)*a1;
    end
end
%-------------------------------------------------------------
function pop = mutation(pop)
%变异算子
for i = 1:pop.size%遍历每个个体
    if rand<pop.mr%依概率执行变异
        n = ceil(rand*2);
        pop.individual(i,n) = pop.individual(i,n) + pop.ma*randn;
% 变异
        if n == 1
            pop.individual(i,n) = max(min(pop.individual(i,n),12),-3);
        else
            pop.individual(i,n) = max(min(pop.individual(i,n),6),4);
        end
    end
end
#接下来，我们要开始来运行我们的遗传算法。
#首先，画个等值线图
clear all
close all
rand('state',1) %设置随机数生成器的状态%
randn('state',1)
```

```
[X,Y] = meshgrid(-3:0.15:12,4:0.02:6);
% 生成绘图用均匀采样的二维决策变量点坐标
for i = 1:size(X,2)
    x = [X(:,i),Y(:,i)];
    Z(:,i) = fun1(x); %计算目标函数值
end
figure(1)
contour(X,Y,Z) %绘制等值线图
hold on
#设置相应的参数，初始化
pop.size = 20;%种群规模
pop.cr = 0.4;%交叉概率
pop.mr = 0.4;%变异概率
pop.ma = 1;
pop = initialization(pop);%初始化种群
figure(1)
plot(pop.solution(:,1),pop.solution(:,2),'ok','MarkerSize',10,'LineWidth',2)
plot(pop.bestSolution(end,1),pop.bestSolution(end,2),'ok','MarkerSize',10,'Li
neWidth',2,'MarkerFaceColor','g')
hold off

for i = 1:100
    pop = newPop(pop);%生成新一代种群
    %重新绘制等值线图
    figure(1)
    contour(X,Y,Z)
    hold on
    plot(pop.solution(:,1),pop.solution(:,2),'ok','MarkerSize',10,'LineWidt
h',2)  plot(pop.bestSolution(end,1),pop.bestSolution(end,2),'ok','MarkerSiz
e',10,'LineWidth',2,'MarkerFaceColor','g')
    hold off
    pause(0.05)
end
```

运用GA工具箱求解

例 5：求解多约束非线性规划问题

$$\min\ f(x) = e^{x_i}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1);$$
$$s.t.\begin{cases} 1.5 + x_1x_2 - x_1 - x_2 \le 0 \\ -x_1x_2 \le 10 \end{cases}$$

matlab代码

```
%主程序采用遗传算法接力进化
clc;
close all;
clear all;
T=100;
optionsOrigin=gaoptimset('Generations',T/2);
[x,fval,reason,output,finnal_pop]=ga(@ch14_2f,2,optionsOrigin);
options1=gaoptimset('Generations',T/2,'InitialPopulation',finnal_pop,'PlotFcn
s',@gaplotbestf);
[x,fval,reason,output,finnal_pop]=ga(@ch14_2f,2,options1);
Bestx=x
BestFval=fval
%子函数：适应度函数同时也是目标函数，ch14_2f.m

function f=ch14_2f(x)
g1=1.5+x(1)*x(2)-x(1)-x(2);
g2=-x(1)*x(2);
if(g1>0|g2>10)
    f=100;
else
    f=exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
end
#还可以使用matlab2016以上版本（直接在界面用工具箱）
#在菜单栏点击  App  ---> Optimization Tool
#弹出对话框，左边Solver栏 下拉选取 ga-  Genetic Algorithm
#在相应栏目中 Fitness Function:   @ch14_2f
#                          Number of Variables:  2
#点star（ 主要需要把文件放在当前文件夹，会有提示）
```

三. 模拟退火算法

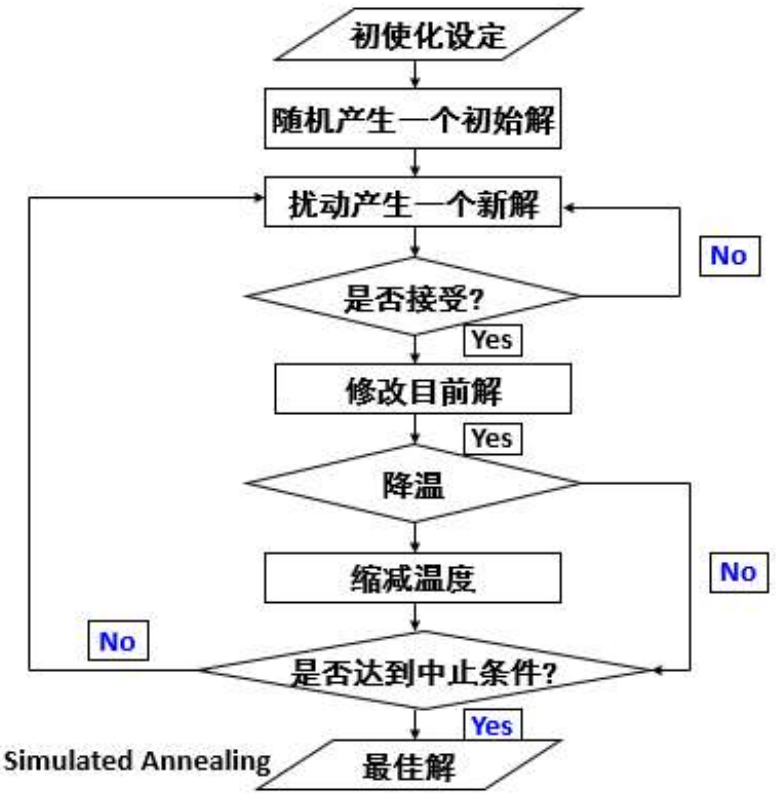
模拟退火算法(Simulated Annealing，SA)最早的思想是由N. Metropolis 等人于1953年提出。1983 年,S. Kirkpatrick 等成功地将退火思想引入到组合优化领域。它是基于Monte-Carlo迭代求解策略的一种随机寻优算法，其出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。模拟退火算法从某一较高初温出发，伴随温度参数的不断下降,结合概率突跳特性在解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。模拟退火算法是一种通用的优化算法，理论上算法具有概率的全局优化性能,目前已在工程中得到了广泛应用，诸如VLSI、生产调度、控制工程、机器学习、神经网络、信号处理等领域。

1. 模拟退火算法-流程

目标：求最优化问题 $\min f(x)$

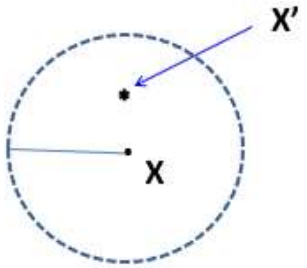
- 需先设定一些参数 t，接着随机产生一个初始的当前解 X ，并计算他的目标函数值 $f(X)$。
- 以当前解为中心对解空间做随机扰动，产生一个扰动解 X'，其目标函数值为$f(X')$。
- 若接受，则以该扰动解取代目前解作为该次迭代的解。
- 让$\Delta f = f(x') - f(x)$，若$\Delta f < 0$，则新解X,被接受; 若 $\Delta f > 0$ 则以概率; $p(\Delta f) = \exp(-\Delta f/T)$接受为新解。这里 $T = kt, k$为Boltzmann常数
- 接着判断是否满足降温条件，若是，则透过冷却机制降温，$T = \alpha \times T, \alpha \in [0, 1]$。转到步骤2继续执行。
- 反之，维持目前温度。之后判断是否达到终止条件，例如达到设定的迭代次数或是连续几次迭代目前解都不再改变时。

模拟退火算法-流程图



2. 扰动方法、概率函数

扰动的作法就是以当前解为中心，对部分或整个解空间随机取样一个解



概率函数为：

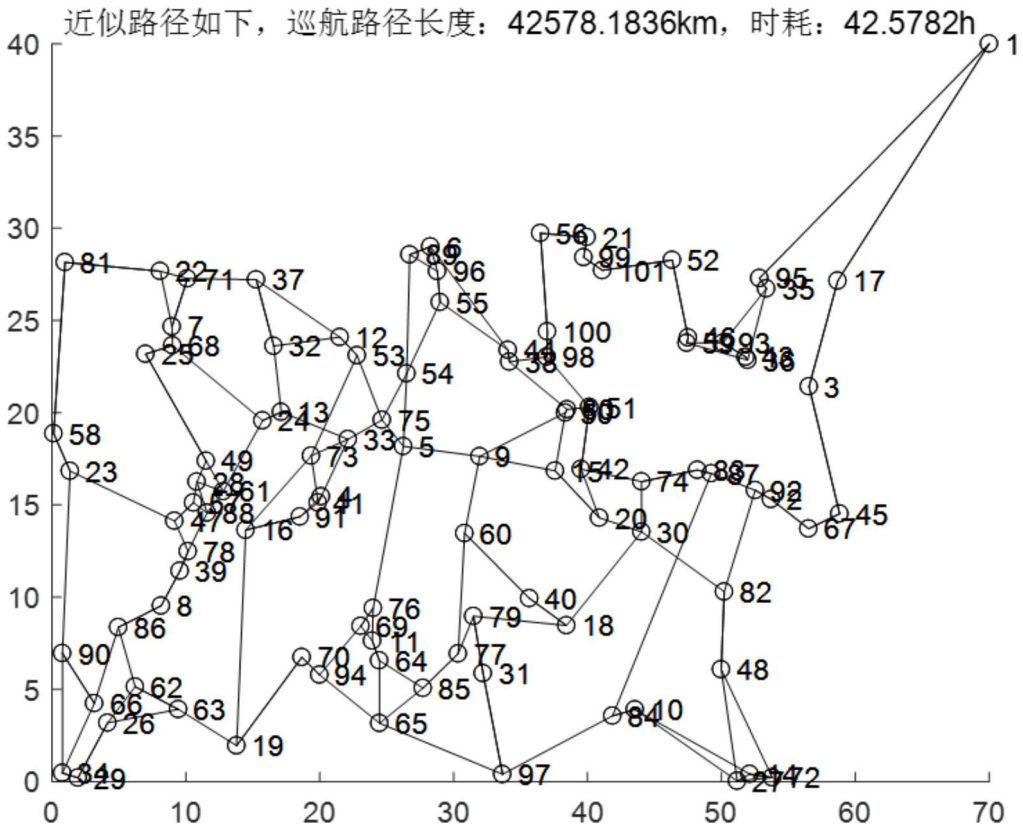
$$P = \begin{cases} 1, \Delta f \le 0 \\ \exp(-\Delta f/T), \Delta f > 0 \end{cases}$$

其中$\Delta f = f(x') - f(x)$

- 当 T 越高或Δf越小时，则 P 越大，相对的扰动解被接受成新解的机率越高。
- 参数 T 随着迭代次数的增加而逐渐下降，所以较差的扰动解被接受成新解的机会也随着 T 的下降而越來越小。
- 所以当迭代到最后因为温度 T 已到达低点，这时系统只会接受较佳的扰动解为新解。
- 当 T 很高时，很可能有扰动解跳出了局部，在全局搜索，因对 Δf没有要求。

例 6：基地巡航

题目：我方有一个基地，经度和纬度为（70，40）。假设我方飞机的速度为 1000 公里/小时。我方派一架飞机从基地出发，侦察完敌方所有目标，再返回原来的基地。在敌方每一目标点的侦察时间不计，求该架飞机所花费的时间（假设我方飞机巡航时间可以充分长）。敌方目标的坐标保存为 sj.txt 文件



最短巡航路径长度：42578.1836km
最短巡航路径时耗：42.5782h

Matlab代码

```
function Simulated_AnneALIng()
%算法：模拟退火算法(Simulated AnneALIng)

%敌方经度纬度数据见 sj.txt

%% 清空
clc,clear;
%% 坐标转换，计算距离
load sj.txt
 %加载敌方100个目标的数据，表格中的位置保存在纯文本文件 sj.txt 中
x=sj(:,1:2:8) ; x=x(:);%将4列变成1列
y=sj(:,2:2:8) ; y=y(:);
sj=[x y];
d1=[70,40] ; sj=[d1;sj;d1] ;
sj1=sj;
sj=sj*pi/180;
n=length(sj);
d=zeros(n);   %距离矩阵 d
for i=1:n-1  %坐标转换为距离
     for j=i+1:n
          temp=cos(sj(i,1)-sj(j,1))*cos(sj(i,2))*cos(sj(j,2))+sin(sj(i,2))*si
n(sj(j,2));
          d(i,j)=6370*acos(temp);
     end
end
d=d+d’ ; S0=[] ; Sum=inf;
%% 设定初始解（MonteCarlo法）
for j=1:1000
     S=[1,1+randperm(n-2),n]; temp=0;
 %用randperm函数随机打乱2到(n-2)之间的数
     for i=1:n-1
          temp=temp+d(S(i),S(i+1));
     end
     if temp<Sum
          S0=S; Sum=temp;
     end
end
e=0.1^30; L=20000; at=0.999; T=1;%参数设置
%% 退火过程
for k=1:L  %产生新解
    %随机找两个点来交换路线
```

```
        c=2+floor((n-2)*rand(1,2)); c=sort(c);
        c1=c(1);c2=c(2);
        df=d(S0(c1-1),S0(c2))+d(S0(c1),S0(c2+1))-d(S0(c1-1),S0(c1))-d(S0(c2),S0
    (c2+1));
       %计算代价函数值
        %接受准则
        if df<0 %路径更短，接受新解
            S0=[S0(1:c1-1),S0(c2:-1:c1),S0(c2+1:n)];
            Sum=Sum+df ; T=T*at;%接受并降温
        elseif exp(-df/T)>rand(1)  %路径没变短，以一定概率接受新解
            S0=[S0(1:c1-1),S0(c2:-1:c1),S0(c2+1:n)];
            Sum=Sum+df ; T=T*at;
        end
        if T<e
            break;
        end
    end
    disp(['最短巡航路径长度：',num2str(Sum),'km']);
    disp(['最短巡航路径时耗：',num2str(Sum/1000),'h']);

    %% 画图
    hold on
    plot(sj1(1:101,1), sj1(1:101,2), 'ko');
    for i=1:101
        temp=['  ',int2str(i)];
        text(sj1(i,1), sj1(i,2),temp);
    end
    for i=1:101
        plot(sj1(S0(i:i+1),1)', sj1(S0(i:i+1),2)','k-');
    end
    title(['近似路径如下，巡航路径长度：',num2str(Sum),'km，','时耗：',num2str(Su
    m/1000),'h']);
    hold off
```

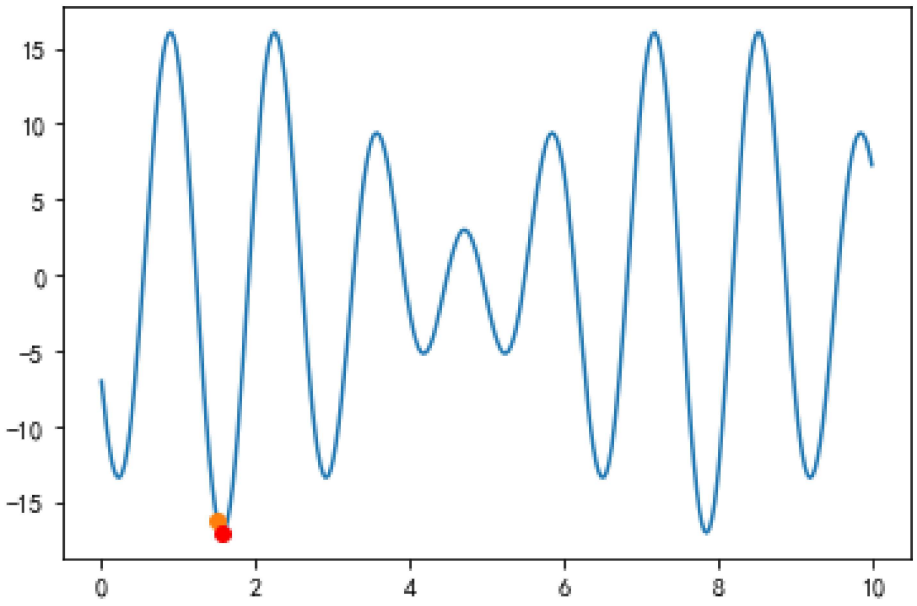
例 7 求解最优化问题 $\max f(x) = 10\sin(5x) + 7\cos 4x$

Python代码：

In [9]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
def function(x):
    return -(10*math.sin(5*x)+7*math.cos(4*x))

initT = 1000
minT = 0.005
iterL = 50
eta = 0.95
k = 1
x0 = 1.5  #10*(2*np.random.rand()-1)
t = initT
print("init solution :",x0)
yy=[]
xx = np.linspace(0,10,300)
for i in range(len(xx)):
    yy.append(function(xx[i]))

plt.figure()
plt.plot(xx,yy)
plt.plot(x0,function(x0),'o')

x_old = x0
while t > minT:
    for i in range(iterL):#MonteCarlo method reject propblity
        value_old = function(x_old)
        x_new = x_old+(np.random.rand()-0.5)*t*0.8
        if x_new>=0 and x_new<=10:
            value_new = function(x_new)
            res = value_new-value_old
            if res<0 or np.exp(-res/(k*t))>np.random.rand():
                x_old = x_new
    t = t*eta

print("最优解: ",x_old)
print("最优值: ",function(x_old))
plt.plot(x_old,function(x_old),'or')
plt.show()
```

```
init solution : 1.5
最优解:  1.572231061435348
最优值:  -16.999627419528416
```



3. 模拟退火算法-设计

实际距离的求解：

设A, B两点的地理坐标分别为(x1,y1), (x2,y2)，过 A, B两点的大圆的劣弧长即为两点的实际距离。以地心为坐标原点o，以赤道平面为XOY平面，以0度经线圈所在的平面为XOZ平面建立三维直角坐标系。地球半径为R=6370km。

则 A, B两点的直角坐标分别为：

A(R*cos(x1)*cos(y1), R*sin(x1)*cos(y1), R*sin(y1))

B(R*cos(x2)*cos(y2), R*sin(x2)*cos(y2), R*sin(y2))

A、B两点实际距离：

d=Rarccos[cos(x1-x2)*cos(y1)*cos(y2) + sin(y1)*sin(y2)]。

建立矩阵D，将所有点之间的距离存放进去。问题转化为从（70，40）出发，走遍所有点，并返回出发点。

先用蒙特卡洛方法（Monte Carlo)得到一组初始解

算法原理：

设 $S = \{s_1, s_2, \cdots, s_n\}$ 为所有可能的状态所构成的集合，

$f: S \rightarrow R$ 为非负代价函数，

即优化问题抽象如下：

寻找 $s^* \in S$，使得 $f(s^*) = \min f(s_i)$ 任意 $s_i \in S$。

具体步骤：

(1)给定一较高初始温度 T，随机产生初始状态S
(2)按一定方式，对当前状态作随机扰动，产生一个新的状态 $S' = S + sign(\eta).\delta$（其中 δ 为给定的步长，η 为[-1,1]的随机数）
计算 $\Delta = f(x') - f(x)$
(3)若 $\Delta < 0$，则令 $S = S'$，转第（5）步
(4)若 $\Delta \leq 0$,则以概率 $\exp(-\Delta/T)$ 接受 S'，即 $S = S'$
　　具体操作：产生一个在 $[0，1]$ 上服从均匀分布的随机数x，若 $x < \exp(-\Delta/T)$，则 $S = S$；否则S保持不变
(5)按一定方式降温，使 $T \leq T_i$， 如： $T = \alpha T_i$，习惯上取 $\alpha \in (0.8，0.9999)$
(6)检查退火是否结束， 是——转向第(7)步， 否——转向第(2)步
(7)以当前 S_i 作为最优解输出

---

4. 模拟退火算法-巡回旅行商问题

巡回旅行商问题（Traveling salesman problem, TSP）可描述如下：已知N个城市之间的相互距离，现有一推销员必须遍历这N个城市，并且每个城市只能访问一次，最后又必须返回出发城市，如何安排他访问这些城市的次序，使其旅行路线总长度最短?

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

　　TSP具有广泛的应用背景和重要理论价值，特别在机器人运动规划中得到许多应用，例如：移动机器人的全局路径规划问题、焊接机器人的任务规划问题等等。
　　但是，巡回旅行商问题中可能的路径数目与城市数目N呈指数型增长，所有的旅程路线组合数 $\frac{(n-1)!}{2n}$
　　其已经被证明属于NP难题。例如30个城市的路线对应约有 $\frac{(30-1)!}{2\$\times\$30}$=1.4736×$10^{29}$ 条
　　对庞大的搜索空间寻求最优解，常规方法和现有的计算工具存在着诸多的计算困难。
　　TSP的解空间S是遍访每个城市恰好一次的所有回路，是所有城市排列的集合。TSP问题的解空间S可表示为{1,2,…,n}的所有排列的集合，即S = {(c_1,c_2,…,c_n) |((c_1,c_2,…,c_n)为{1,2,…,n}的排列）}，其中每一个排列 S_i 表示遍访n个城市的一个路径，c_i = j表示在第i次访问城市j。模拟退火算法的最优解与初始状态无关，故初始解为随机函数生成一个{1,2,…,n}的随机排列作为 S_0。
新解产生:
新解的产生对问题的求解非常重要。新解可通过分别或者交替用以下2种方法产生：

①二变换法：任选序号u,v(设uvn)，交换u和v之间的访问顺序，若交换前的解为

$S_i=(C_1,C_2,...C_{u-1},C_u,C_{u+1},...C_v,C_{v+1},...,C_n)$

交换后的路径为新路径，即：

$S_i'=(C_1,C_2,...C_{u-1},C_v,C_{v-1},...C_u,C_{v+1},...,C_n)$

新解产生:

新解的产生对问题的求解非常重要。新解可通过分别或者交替用以下2种方法产生：

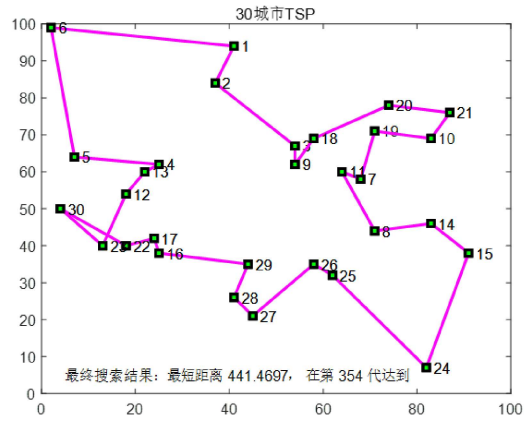
①②三变换法：任选序号u,v和ω(u≤vω)，将u和v之间的路径插到ω之后访问，若交换前的解为s

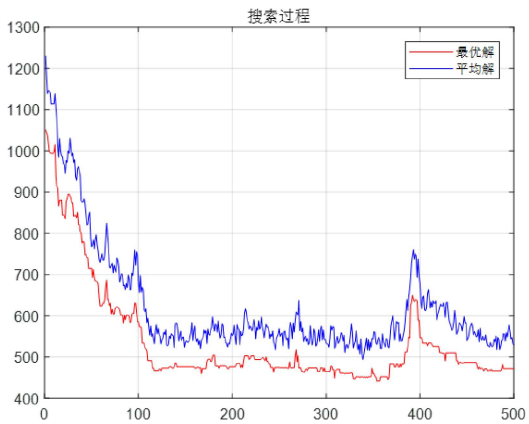
$S_i=(C_1,C_2,...C_{u-1},C_u,C_{u+1},...C_v,C_{v+1},...,C_n)$

交换后的路径为新路径，即：

$S_i'=(C_1,C_2,...C_{u-1},C_{v-1},...,C_w,C_u,,C_{u+1},...,C_v,C_{w+1},...,,C_n)$

学习启发式算法时，旅行商问题是一个经典的例子。其中，遗传算法可以用来求解该问题。遗传算法是一种进化算法，由于其启发式算法的属性，并不能保证得到最优解。求解效果与初始种群选取，编码方法，选择方法，交叉变异规则有关。





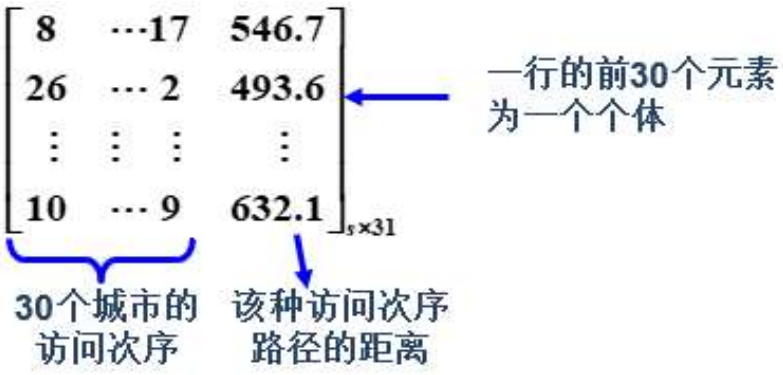
已知30个城市的坐标，求巡回旅行最短路：

序号	X坐标	Y坐标	序号	X坐标	Y坐标	序号	X坐标	Y坐标
1	18	54	11	71	44	21	7	64
2	87	76	12	18	64	22	22	60
3	74	78	13	68	58	23	25	62
4	71	71	14	83	69	24	62	32

序号	X坐标	Y坐标	序号	X坐标	Y坐标	序号	X坐标	Y坐标
5	25	38	15	58	69	25	87	7
6	58	54	16	54	62	26	91	38
7	4	50	17	51	67	27	83	46
8	13	40	18	37	84	28	41	26
9	18	40	19	41	94	29	45	21
10	24	43	20	2	99	30	44	35

编码与解码：

采用对访问城市序列进行排列组合的方法编码，即某个巡回路径的染色体是该巡回路径的城市序列。对于N（N为城市总数）进制编码，即每个基因仅从1到N得整数里面取一个值，每个个体的长度为N。



将个体的染色体值作为存储30个城市坐标的矩阵的下标来引用，输出对应的矩阵元素，便可实现解码。

适应度函数：

在TSP问题中，用路径的总长度作为适应度函数来衡量求解结果是否最优，路径越短对应的个体越优，其适应度值应越大。 两城市间的距离为：$d_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$

个体代表的路径的总长度为：$L=\sum_{k=1}^{29} d(k)$

采用倒数法将适应度函数取为：*fitness*=1/L

选择操作：将群体中适应度较大的C个个体直接替换适应度较小的C个个体。

$$C=Select\_Operator\times Population\_Size$$

交叉操作：

本例中采用有序交叉执行交叉操作。有序交叉能够有效地继承双亲的部分基因成分，达到了进化的遗传功能，使该遗传算法并不盲目搜索，二是趋向于使群体具有更多的优良基因。交叉后，考察父个体与子个体的适应度来决定是否更新种群。具体操作过程如下（以0~9编码为例）：

父个体：（“|”表示交叉点）

A1：（0 1 2 | 3 4 5 6 | 7 8 9）

A2：（4 2 9 | 0 8 5 3 | 1 7 6）

保留中间部分，交换对应位置的子串

A1： (8 2 9 | 3 4 5 6 | 1 7 0)

A2： (6 1 2 | 0 8 5 3 | 7 4 9)

变异操作：

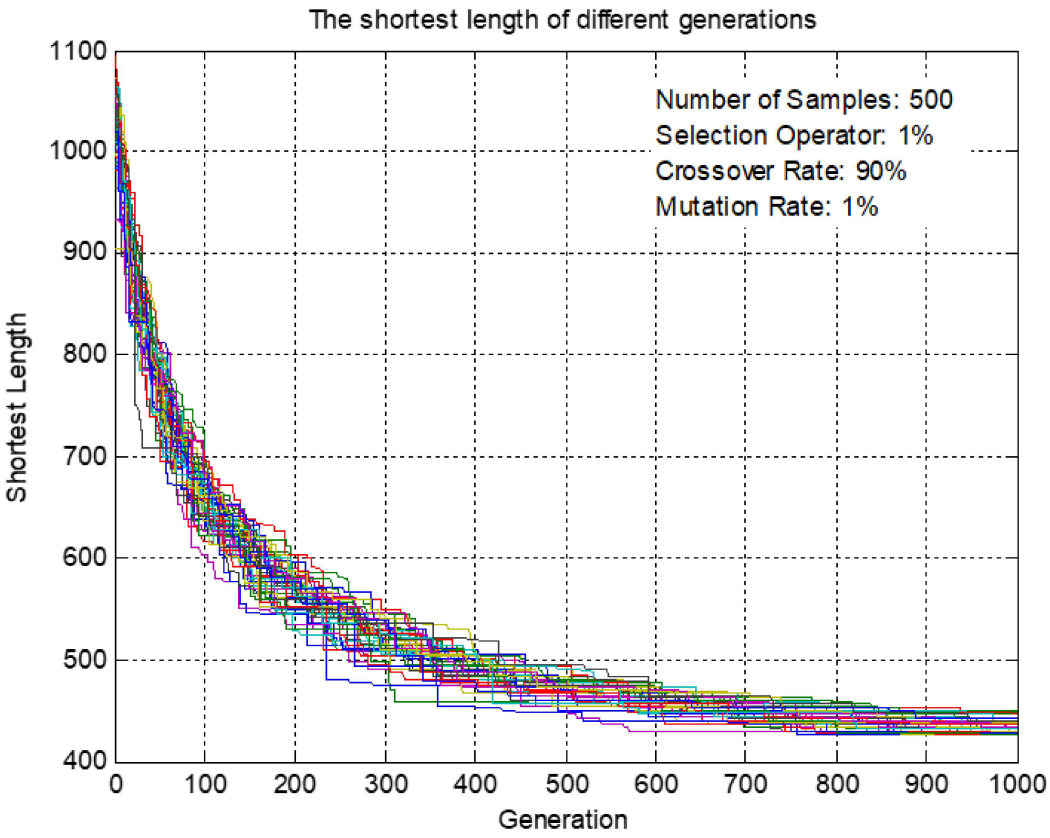
若变异后子代的适应度值更加优异，则保留子代染色体，否则仍保留父代染色体。本例中，采用倒置变异法。例如：假设当前个体为“5678412390”，如果当前随机值p∈[0，1]≥pmutation，则随机选择来自同一个体的两个点（设为“8”和“2”），执行变异操作，即倒置该两点的中间部分。产生的新个体为“5672148390”。

变异前父个体：

A1： (5 6 7 | 8 4 1 2 | 3 9 0)

变异后子代个体：

A1： (5 6 7 | 2 1 4 8 | 3 9 0)



引用及参考：

[1]《Python数据结构与算法分析》布拉德利.米勒等著，人民邮电出版社，2019年9月.

[2]

课后练习

1. 写出遗传算法的完整Python代码或 C语言代码。

2. 写出模拟退火算法的完整Python代码或 C语言代码，举例说明

3. 对遗传算法和模拟退火算法做时间复杂度分析

讨论、思考题、作业：

参考资料（含参考书、文献等）：算法导论. Thomas H. Cormen等，机械工业出版社，2017.

授课类型（请打√）：理论课□ 讨论课□ 实验课□ 练习课□ 其他□

教学过程设计（请打√）：复习□ 授新课□ 安排讨论□ 布置作业□

教学方式（请打√）：讲授□ 讨论□ 示教□ 指导□ 其他□

教学资源（请打√）：多媒体□ 模型□ 实物□ 挂图□ 音像□ 其他□

填表说明：1、每项页面大小可自行添减；2、教学内容与讨论、思考题、作业部分可合二为一。