# 再谈MCMC方法

2015-10-15 宾狗 学术 数学

## 0x00 前言

最初了解到MCMC方法是因为学习LDA算法。我的博客中也有好几篇文章均涉及到了MCMC方法(Markov Chain Monte Carlo Methods)，它是一组用马氏链从随机分布取样的算法。MCMC中第一个MC指的是马尔可夫链，这个就是**随机过程**课程里面的东西了。第二个MC指的是蒙特卡洛方法。蒙特卡洛方法是一种随机模拟方法。随机模拟的思想由来已久，蒲丰投针就是一个非常典型的应用。

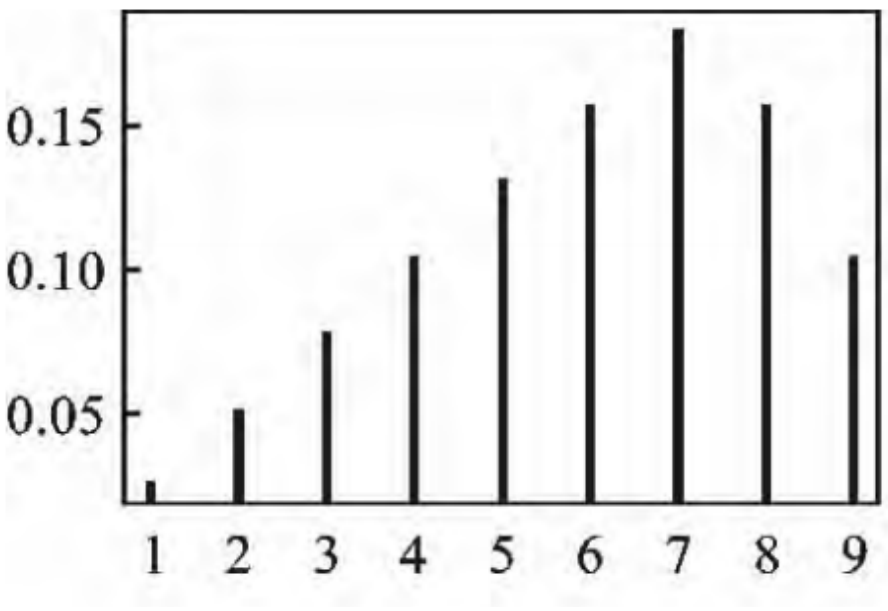
本文先简单介绍MCMC方法，再分别从几个具体场景(模拟分布、求积分、破解密码、最优化算法法)介绍MCMC方法的应用，最后讨论一下MCMC方法与模拟退火算法之间的联系。

## 0x01 MCMC方法简介(千岛湖植物考查)

关于MCMC方法网上已有非常丰富的资料，认真研究一下就很容易搞懂。这里借用一个非常形象的例子描述MCMC中最比较常用的`M-H算法`的基本流程。

植物学家要前往千岛湖进行植物考查，考察的时间安排是依据各岛上植物种类数来确定的，植物种类数多的岛屿考察时间长，植物种类数少的岛屿考察时间短。但在考察之前，植物学家并不清楚各个岛的植物种类数，甚至不知道一共有多少个岛，所以不能事先确定留在每一个岛上的天数，也无法得知在各岛考察时间占总考察时间的比率———考察时间分布。

假设有9个岛屿(为了方便说明，即使岛屿数量未知，按照这个方法也是可行的)，一字排开，植物学家来到某个岛屿后，不仅可以了解当前岛屿的植物数量，还可以了解相邻岛屿的植物数量。植物学家只能从当前岛屿移动到相邻岛屿。假设这9个岛屿的植物数量分布如下：

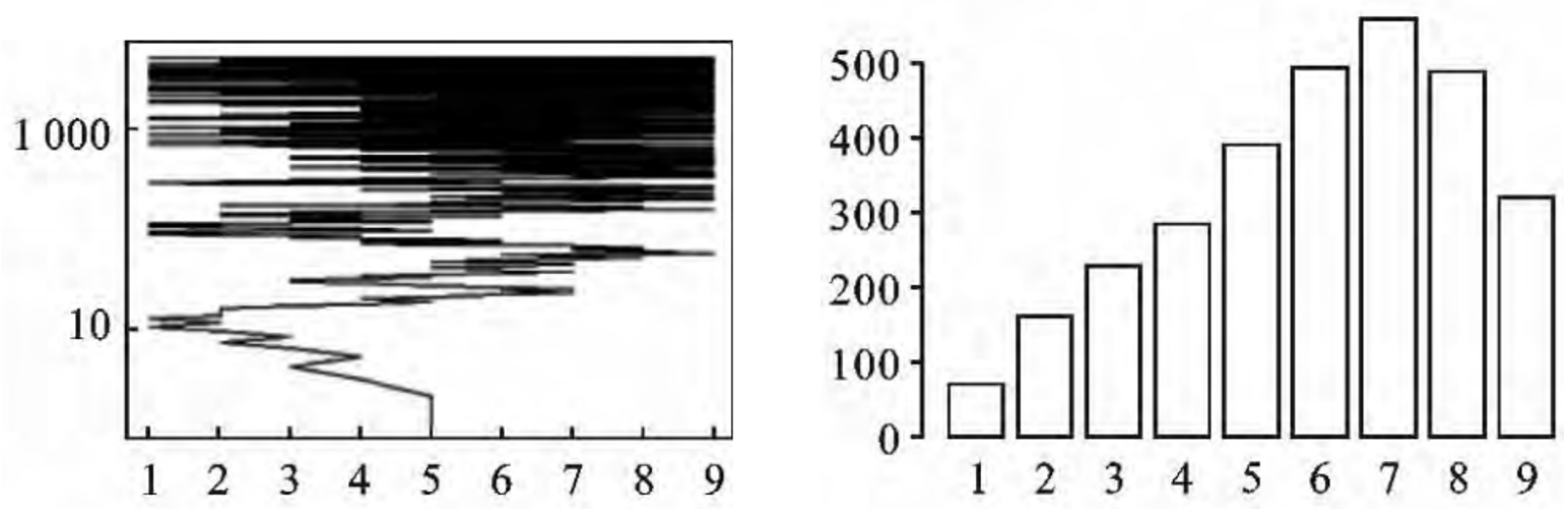



精通MCMC的统计学者向植物学家建议用`M-H算法`算法的思想，通过“两步随机试探法”来决定考察路线和停留时间。植物学家到达某岛屿后，由于岛屿是一字排开，所以面临三种选择（三个状态）：一是继续停留在岛屿上（保持原状态）；二是到临近左面岛屿（转移到前状态）；三是到临近右面岛屿（转移到后状态）。这三种状态可以转化为两种随机选择：一是去或留，二是左或右。但`M-H算法`突破常理，用“逆向思维”来确定选择过程，先确定左或右，再决定去或留（究其原因可能是先基于左或右岛的情况，再与当前岛屿比较，最后决定是走还是留）。“两步随机试探法”的具体实施过程如下：

1. 随机确定方向，即向左还是向右：植物学家在某岛屿上考察时（假设在第5个岛屿），就“可能”计划前往邻近岛屿（第4个岛或第6个岛），到底是向左还是向右呢？植物学家通过随机掷一枚均匀硬币来决定，如果硬币为正面，就计划向左去第4个岛；如果硬币为反面，则计划向右去第6个岛。
2. 基于第一步的信息，部分随机确定意愿—出发还是停留：假设第一步硬币为正面，根据第一步规则，植物学家应计划“可能”向左去第4个岛，但“可能”去，也“可能”不去。如何确定呢？统计学者给出的策略是，从当前岛（第5个岛）的管理者那里可以得到目标岛（左面第4个岛）的植物种类数，比较目标岛和当前岛的植物种类数量，再决定是否出发。**如果目标岛的植物种类数比所处当前岛的植物种类数多，那么植物学家就一定前往目标岛进行考察研究；如果目标岛的植物种类数比所处当前岛的少，那么植物学家又需要“另外一步随机试探法”来确定意愿—依概率出发或是停留。**什么是依概率出发呢？举例说明：假设目标岛有150种植物,而所处当前岛有200种植物,则前往目标岛的概率就为0.75(150/200),继续留在当前岛的概率为0.25(1-0.75)。那么植物学家到底是出发还是停留呢？似乎还没有确切答案,此时的随机判断方法是：在地上画一条1米长的线段,然后随机向该线段内投一个石子,如果石子落在0和0.75米之间,则出发去目标岛考察研究;如果石子落在0.75和1之间,则继续留在当前岛。

以上植物学家的随机试探方法本质上就是马氏链的转移矩阵，那么该植物学家设计的马式链的平 稳分布（各个岛屿的考察研究的时间分布）是否与其目标分布（即各个岛屿的植物种类数分布）相同呢？

通过程序模拟，下图是植物学家的移动路线和最终在每个岛屿上停留的时间分布




可以看到最终科学家们在每个岛屿停留时间的分布与每个到植物数量的分布是一致的，这就是MCMC方法的神奇之处~

# 0x02 MCMC模拟Beta分布

在花式生成正态分布中，提到了一种拒绝采样的方法，这背后蕴含的其实就是蒙特卡洛方法的思想。

这里以Beta分布为例，Beta分布的概率密度函数PDF是$f(x) = Cx^{\alpha-1}(1-x)^{\beta-1}$，其中$C = 1/B(\alpha,\beta)$，回顾《花式生成正态分布》里面提到的方法，如果采用反变换法，则需要对这个函数求积分和反函数，非常麻烦。如果采用拒绝采样就简单直观了，对照千岛湖植物考察的例子，为了对Beta分布进行采样，需要定义转移概率矩阵和接受概率，这里可以忽略转移概率矩阵(不同状态之间的转移概率是相同的)，只考虑接受概率

$$
\begin{aligned}
a_{ij} &= \min(1, \pi_j p_{ji} / \pi_i p_{ij}) \\
&= \min(1, \pi_j / \pi_i)
\end{aligned}
$$

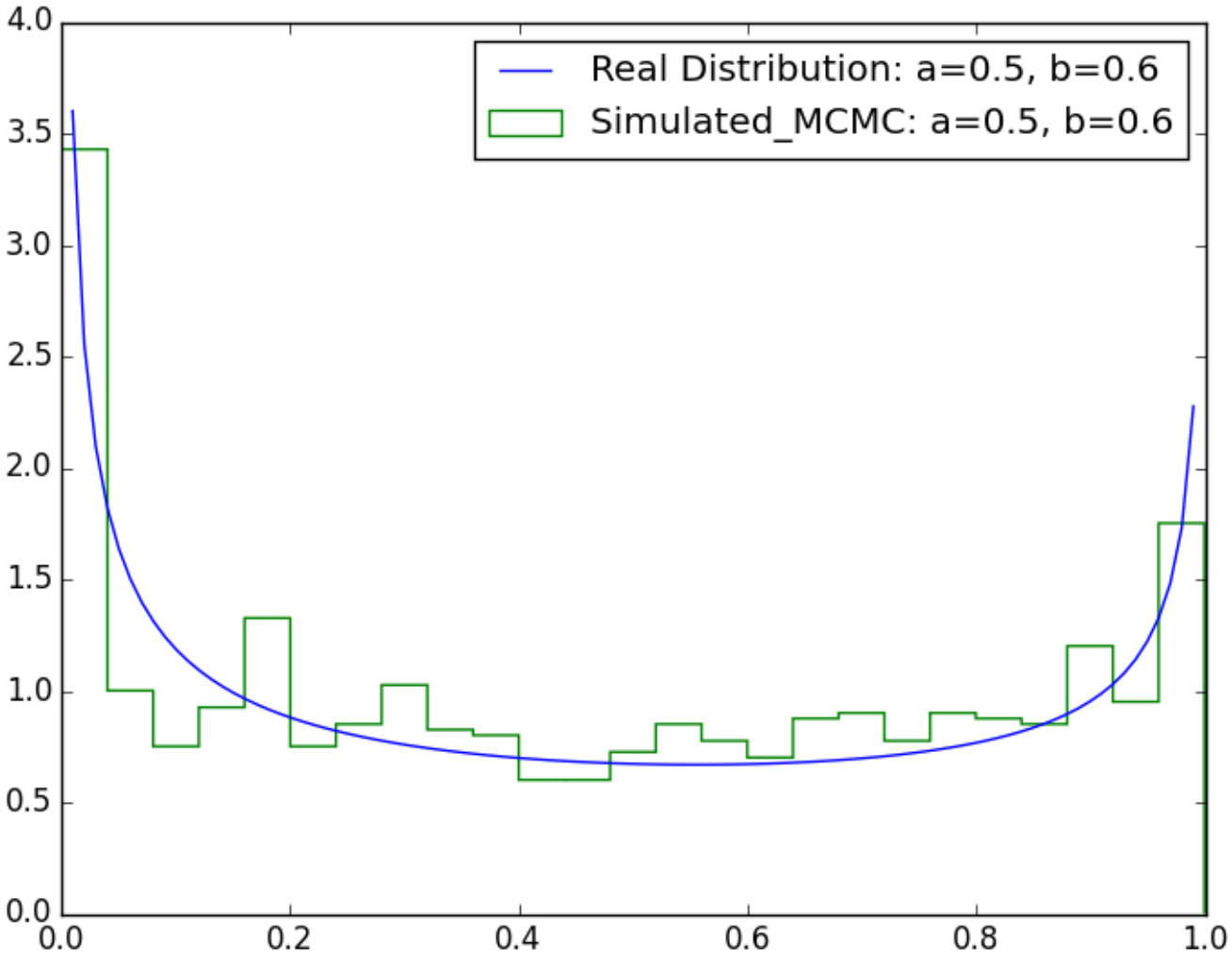
$\pi_i$和$\pi_j$为平稳分布概率，与Beta分布的概率密度是一致的，因此

$$
\begin{aligned}
\pi_i &= Ci^{\alpha-1}(1-i)^{\beta-1} \\
\pi_j &= Cj^{\alpha-1}(1-j)^{\beta-1}
\end{aligned}
$$

```
# -*- coding: utf-8 -*-
import random
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

# 真实Beta分布概率密度函数
def beta(x, a, b):
    return (1.0 / ss.beta(a,b)) * x**(a-1) * (1-x)**(b-1)

# Beta分布概率密度函数(忽略了Beta函数)
def beta_fpdf(x,a,b):
    return x**(a-1) * (1-x)**(b-1)

# 根据接受概率决定是否转移
def transform(ap):
    stone = random.uniform(0,1)
    if stone>=ap:
        return False
    else:
        return True

def beta_mcmc(N_hops,a,b):
    states = []
    cur = random.uniform(0,1)
    for i in range(0,N_hops):
        states.append(cur)
        next = random.uniform(0,1)
        ap = min(beta_fpdf(next,a,b)/beta_fpdf(cur,a,b),1) # 计算接受概率
        if transform(ap):
            cur = next
    return states[-1000:] # 返回进入平稳分布后的1000个状态


# 绘制通过MCMC方法抽样的Beta分布
```

```
def plot_beta(a, b):
    Ly = []
    Lx = []
    i_list = np.mgrid[0:1:100j]
    for i in i_list:
        Lx.append(i)
        Ly.append(beta(i, a, b))
    # 绘制真实的Beta分布进行对照
    plt.plot(Lx, Ly, label="Real Distribution: a="+str(a)+", b="+str(b))
    plt.hist(beta_mcmc(100000,a,b),normed=True,bins=25, histtype='step',label="Simulated_MCMC:
a="+str(a)+", b="+str(b))
    plt.legend()
    plt.show()

plot_beta(0.5, 0.6)
```

结果如下图所示




## 0x03 高维积分

长期以来，由于贝叶斯后验分布在高维计算上的困难，贝叶斯统计推断难以实现，贝叶斯方法的应用受到了很大限制。而MCMC方法的出现和发展使得现代贝叶斯分析的理论和应用得到了迅速发展。

关于如何用MCMC方法求积分，统计之都上早有一些文章进行了介绍，如邓一硕的蒙特卡洛方法与定积分计算，所以本文不再重复叙述，只列举一个“高维积分”的例子，下图是高数中多重积分的一个例子，用MC方法就非常简单直观

如果积分区域是其他的情形，可以用类似的方法计算.

例 1 计算三重积分 $\iiint_\Omega xdV$，其中 $\Omega$ 是由三个坐标面和平面 $x+y+z=1$ 所围的立体区域.

解　积分区域如图所示，可以用不等式表示为

$$0\le x\le 1, 0\le y\le 1-x, 0\le z\le 1-x-y,$$

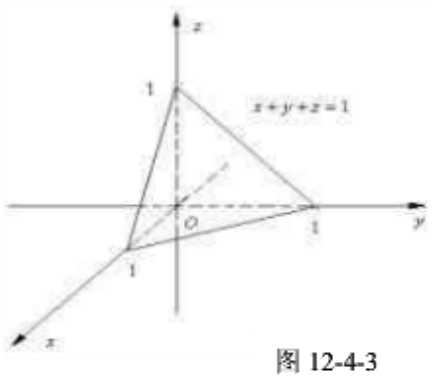


图 12-4-3

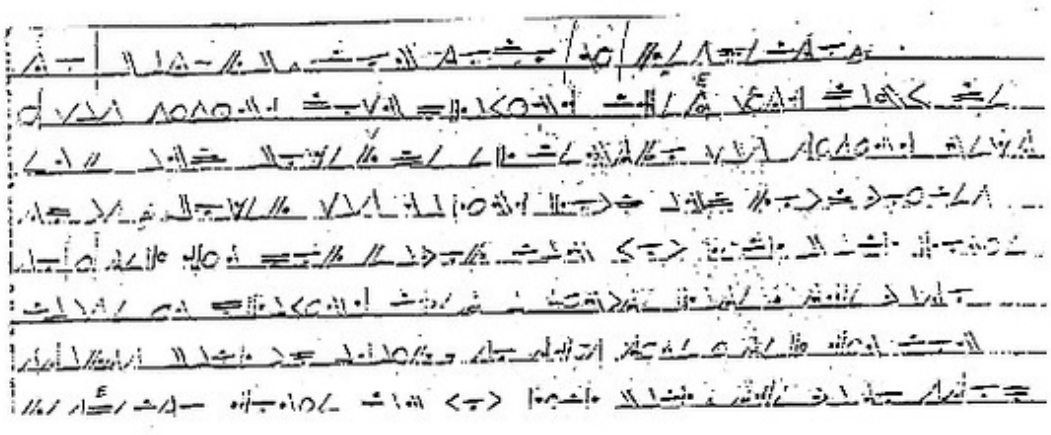
所以积分可以化为

$$\begin{aligned}\iiint_\Omega xdV &= \int_0^1 dx\int_0^{1-x} dy\int_0^{1-x-y} xdz\\ &= \int_0^1 dx\int_0^{1-x} x(1-x-y)dy\\ &= \int_0^1 \frac{1}{2}x(1-x)^2 dx\\ &= \frac{1}{8}x^4-\frac{1}{3}x^3+\frac{1}{4}x^2\Big|_0^1 = \frac{1}{24}\end{aligned}$$

```
import random

def samplexyz():
    x = random.uniform(0,1)
    y = random.uniform(0,1)
    z = random.uniform(0,1)
    return x,y,z

def fx(x,y,z):
    return x

multidimi = 0.0
for i in range(100000):
    x,y,z = samplexyz()
    if x+y+z <= 1:
        ti = fx(x,y,z)
        multidimi += ti

print multidimi/100000.0
```

# 0x04 破解凯撒密码

斯坦福统计学教授Persi Diaconis是一位传奇式的人物，统计之都上有关于他的介绍，Persi Diaconis (1)，Persi Diaconis(2). 下面要讲的这个故事，是Diaconis 在他的文章The Markov Chain Monte Carlo Revolution中给出的破译犯人密码的例子。 一天，一位研究犯罪心理学的心理医生来到斯坦福拜访Diaconis。他带来了一个囚犯所写的密码信息。他希望Diaconis帮助他把这个密码中的信息找出来。 这个密码里的每个符号应该对应着某个字母，但是如何把这些字母准确地找出来呢？Diaconis和他的学生Marc采用了一种叫做MCMC（马尔科夫链蒙特卡洛）的方法解决了这个问题。

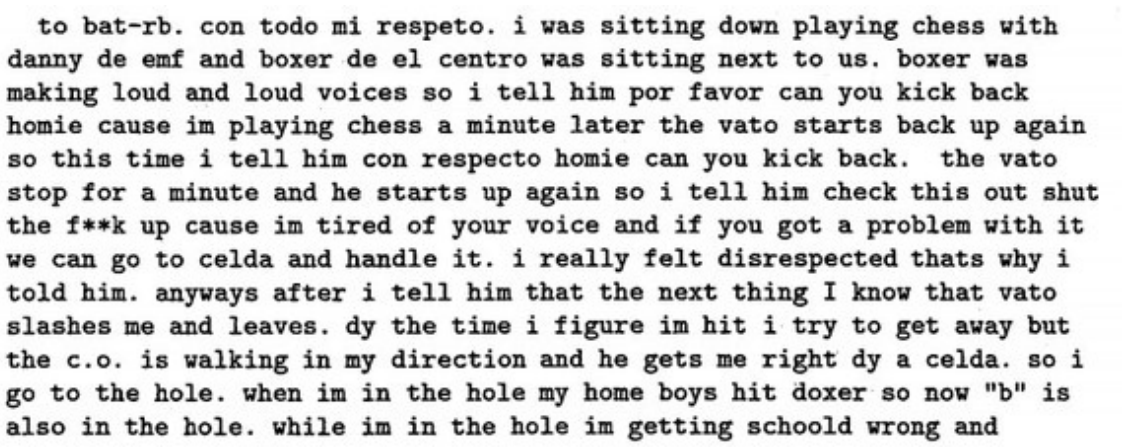


是不是联想到了福尔摩斯里面跳舞的小人？这其实是一个非常典型的凯撒密码，手工用频率分析法，尝试不同的组合，观察结果是否有意义可以解决这个问题。但是除了部分高频字母，大部分字母的出现频率是差不多的，而且与文本内容有关，这样需要尝试的组合非常多，而且需要人为的判断结果是否有意义。

因此单纯的依靠字母频率分析是不够的，应该考虑更一般的特征。比如字母之间的共同出现的频率，更进一步的说，考虑字母之间出现的转移概率，当前一个字母为辅音时，后一个字母出现元音的概率更大，进一步，连续几个辅音出现之后再出现辅音的概率将非常低。

这样就可以请出MCMC方法了，以大量英文语料为基础，统计从字母x到字母y的转移概率。无论是加密前还是加密后的文本，特定位置之间的转移概率是一致的，大致趋近于正常英文语料的转移概率。

Diaconis和他的学生Marc按照这个思路对密文进行解密，程序大概跑了2000多步，得到的信息就已经有意义了，下图给出解密后的结果：



```
  to bat-rb. con todo mi respeto. i was sitting down playing chess with
danny de emf and boxer de el centro was sitting next to us. boxer was
making loud and loud voices so i tell him por favor can you kick back
homie cause im playing chess a minute later the vato starts back up again
so this time i tell him con respecto homie can you kick back.  the vato
stop for a minute and he starts up again so i tell him check this out shut
the f**k up cause im tired of your voice and if you got a problem with it
we can go to celda and handle it. i really felt disrespected thats why i
told him. anyways after i tell him that the next thing I know that vato
slashes me and leaves. dy the time i figure im hit i try to get away but
the c.o. is walking in my direction and he gets me right dy a celda. so i
go to the hole. when im in the hole my home boys hit doxer so now "b" is
also in the hole. while im in the hole im getting schoold wrong and
```

说了这么多，动手实践一下，下面是一个凯撒密码的例子：

> XZ STAVRK HXVR MYAZ OAKZM JKSSO SO MYR OKRR XDP JKSJRK XBMASD SO YAZ TWDHZ MYR JXMBYNSKF BSVRKTRM NYABY NXZ BXKRTRZZTQ OTWDH SVRK MYR AKSD ERPZMRXP KWZMTRP MYR JXTR OXBR SO X QSWDH NSIXD NXZ KXAZRP ORRETQ OKSI MYR JATTSN XDP X OXADM VSABR AIJRKORBMTQ XKMABWTXMRP MYR NSKPZ TRM IR ZRR MYR BYATP XDP PAR MYR ZWKHRSD YXP ERRD ZAMMADH NAMY YAZ OXBR MWKDRP MSNXKPZ MYR OAKR HAVADH MYR JXTIZ SO YAZ YXDPZ X NXKI XDP X KWE XTMRKDXMRTQ XZ MYR QSWDH NSIXD ZJSFR YR KSZR XDP XPVXDBADH MS MYR ERP Z YRXP ZXAP NAMY ISKR FADPDRZZ MYXD IAHYM YXVR ERRD RGJRBMRP SO YAI

我们按照MCMC方法的流程分析一下：

1. 随机生成一个密钥(其实就是表示字母替换规则的字符串，如`ICZNPKXGMBRJSHWDYEALQVUOTF`表示I替换为A，C替换为B，依此类推)，表示当前状态$C$
2. 随机交换旧密钥中两个字母的顺序，生成一个新的密钥，表示可能转移到的下一状态$P$
3. 利用一个评分函数$Score(x)$，计算当前状态评分$Score_C$和下一状态评分$Score_P$
4. 如果$Score_P > Score_C$，则转移至下一状态
5. 否则抛一枚正面朝上概率为$Score_P/Score_C$的硬币，如果正面朝上则转移至下一状态
6. 重复第2步之后的步骤

这个流程是符合MCMC的Sytle的，那么问题来了，评分函数怎么定义？怎么让程序知道某一个密钥比另一个要好？这里我们使用n-gram，为了简单起见只考虑两个字母(bigram)。对任意两个字母$\beta_1$和$\beta_2$(假设$\beta_1 = T$, $\beta_2 = H$)，定义$R(\beta_1, \beta_2)$为字母对$\beta_1\beta_2$(即TH)在参考语料中出现的次数。定义$F_x(\beta_1, \beta_2)$为用密钥$x$解密后解密文本中字母对$\beta_1\beta_2$出现的次数。那么密钥$x$的评分为：

$$Score(x) = \prod R(\beta_1, \beta_2)^{F_x(\beta_1, \beta_2)}$$

当bigram在参考语料和解密后文本中出现的频率越相近，$Score(x)$的值越大，为了方便计算，我们使用$log(Score(x))$进行计算

用python实现

```python
# -*- coding: utf-8 -*-
import math
import random

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# 根据传入的密钥，生成字母替换规则字典
# 例如传入"DGHJKL..."，生成字典{D:A, G:B, H:C...}
def create_cipher_dict(cipher):
    cipher_dict = {}
    alphabet_list = list(alphabet)
    for i in range(len(cipher)):
        cipher_dict[alphabet_list[i]] = cipher[i]
    return cipher_dict

# 使用密钥对文本进行替换(加密/解密)
def apply_cipher_on_text(text,cipher):
    cipher_dict = create_cipher_dict(cipher)
    text = list(text)
    newtext = ""
    for elem in text:
```

```
        if elem.upper() in cipher_dict:
            newtext+=cipher_dict[elem.upper()]
        else:
            newtext+=" "
    return newtext

# 统计参考语料的bigram
# 例如 {'AB':234,'TH':2343,'CD':23 ..}
def create_scoring_params_dict(longtext_path):
    scoring_params = {}
    alphabet_list = list(alphabet)
    with open(longtext_path) as fp:
        for line in fp:
            data = list(line.strip())
            for i in range(len(data)-1):
                alpha_i = data[i].upper()
                alpha_j = data[i+1].upper()
                if alpha_i not in alphabet_list and alpha_i != " ":
                    alpha_i = " "
                if alpha_j not in alphabet_list and alpha_j != " ":
                    alpha_j = " "
                key = alpha_i+alpha_j
                if key in scoring_params:
                    scoring_params[key]+=1
                else:
                    scoring_params[key]=1
    return scoring_params

# 统计解密文本的bigram
# 例如 {'AB':234,'TH':2343,'CD':23 ..}
def score_params_on_cipher(text):
    scoring_params = {}
    alphabet_list = list(alphabet)
    data = list(text.strip())
    for i in range(len(data)-1):
        alpha_i =data[i].upper()
        alpha_j = data[i+1].upper()
        if alpha_i not in alphabet_list and alpha_i != " ":
            alpha_i = " "
        if alpha_j not in alphabet_list and alpha_j != " ":
            alpha_j = " "
        key = alpha_i+alpha_j
        if key in scoring_params:
            scoring_params[key]+=1
        else:
            scoring_params[key]=1
    return scoring_params

# 根据公式计算密钥的评分
def get_cipher_score(text,cipher,scoring_params):
    decrypted_text = apply_cipher_on_text(text,cipher)
    scored_f = score_params_on_cipher(decrypted_text)
    cipher_score = 0
    for k,v in scored_f.iteritems():
        if k in scoring_params:
            cipher_score += v*math.log(scoring_params[k])
    return cipher_score

# 通过随机交换两个字母的顺序 生成一个新的密钥
def generate_cipher(cipher):
    pos1 = random.randint(0, len(list(cipher))-1)
    pos2 = random.randint(0, len(list(cipher))-1)
    if pos1 == pos2:
        return generate_cipher(cipher)
```

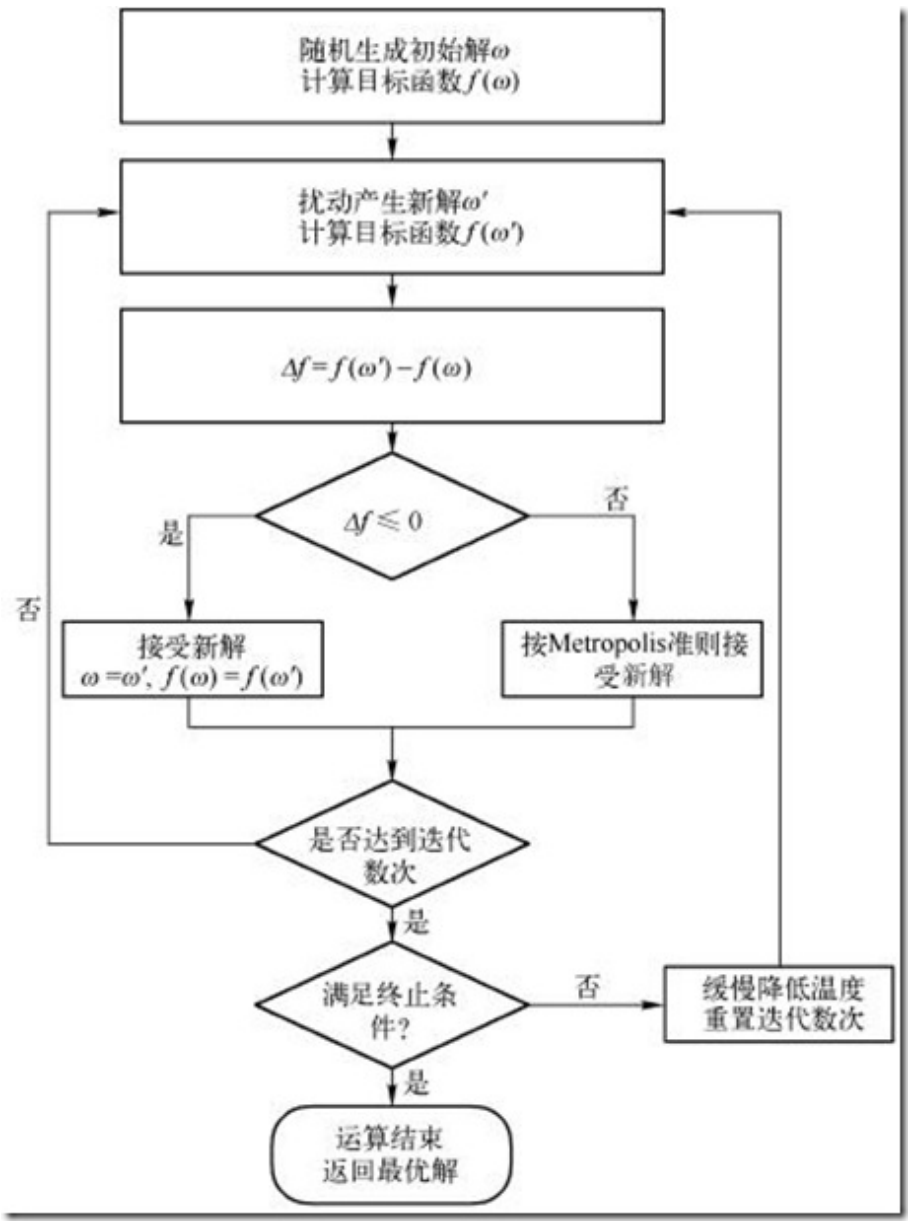
```
    else:
        cipher = list(cipher)
        pos1_alpha = cipher[pos1]
        pos2_alpha = cipher[pos2]
        cipher[pos1] = pos2_alpha
        cipher[pos2] = pos1_alpha
        return "".join(cipher)

# 抛一枚出现正面概率为p的硬币，出现正面返回True，出现反面返回False
def random_coin(p):
    unif = random.uniform(0,1)
    if unif>=p:
        return False
    else:
        return True

# MCMC方法解密 运行n_iter轮
def MCMC_decrypt(n_iter,cipher_text,scoring_params):
    current_cipher = alphabet # 以随机密钥开始
    state_keeper = set()
    best_state = ''
    score = 0
    for i in range(n_iter):
        state_keeper.add(current_cipher)
        proposed_cipher = generate_cipher(current_cipher)
        score_current_cipher = get_cipher_score(cipher_text,current_cipher,scoring_params)
        score_proposed_cipher = get_cipher_score(cipher_text,proposed_cipher,scoring_params)
        acceptance_probability = min(1,math.exp(score_proposed_cipher-score_current_cipher))
        if score_current_cipher>score:
            best_state = current_cipher
        if random_coin(acceptance_probability):
            current_cipher = proposed_cipher
        if i%500==0:
            print "iter",i,":",apply_cipher_on_text(cipher_text,current_cipher)[0:99]
    return state_keeper,best_state

# 主程序开始

# 参考语料：《战争与和平》
scoring_params = create_scoring_params_dict('war_and_peace.txt')
# 测试文本
plain_text = "As Oliver gave this first proof of the free and proper action of his lungs, \
the patchwork coverlet which was carelessly flung over the iron bedstead, rustled; \
the pale face of a young woman was raised feebly from the pillow; and a faint voice imperfectly
\
articulated the words, Let me see the child, and die. \
The surgeon had been sitting with his face turned towards the fire: giving the palms of his
hands a warm \
and a rub alternately. As the young woman spoke, he rose, and advancing to the bed's head, said,
with more kindness \
than might have been expected of him: "

encryption_key = "XEBPROHYAUFTIDSJLKZMWVNGQC"
cipher_text = apply_cipher_on_text(plain_text,encryption_key)
decryption_key = "ICZNBKXGMPRQTWFDYEOLJVUAHS"

print"Text To Decode:", cipher_text
print "\n"
states,best_state = MCMC_decrypt(10000,cipher_text,scoring_params)
print "\n"
print "Decoded Text:",apply_cipher_on_text(cipher_text,best_state)
print "\n"
print "MCMC KEY FOUND:",best_state
print "ACTUAL DECRYPTION KEY:",decryption_key
```

# 0x05 MCMC与模拟退火

其实刚才的凯撒密码问题其实可以看作一个最优化的问题，相信学习过计算机的同学在本科时一定接触过算法设计这门课程，例如贪心法、分治法和回溯法等等，高级一点的还有遗传算法、蚁群算法，还有我们要谈的模拟退火算法(Simulated Annealing)。

模拟退火算法来源于固体退火原理，将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

其算法流程图如下所示：




可以看到，这个算法思想与MCMC方法是非常相似的，如果没有那个以一定概率接受新解的步骤，那么就是单纯的贪心算法，很容易陷入局部最优解。而当我们了解了MCMC方法之后就可以知道，正是这个步骤保证了采样结果符合目标分布，使算法不会陷于局部最优解，能够找到全局最优或近似全局最优解.

# 0x06 总结

MCMC方法不仅是一种引领近代贝叶斯分析复兴的方法，更重要的是一种思维方式的革新。这一点对于非数学专业的研究人员更是一个启示，从有限到无限，从小规模到大数据，从精确计算到算法模拟，这告诉我们：兵无常势，水无常形，大数据时代就要以大数据的思维来思考和分析。

# 0x07 参考资料


- My Tryst With MCMC Algorithms
- Behold the power of MCMC
- 贝叶斯集锦（3）：从MC、MC到MCMC
- LDA数学八卦-LDA-math-MCMC 和 Gibbs Sampling
- MCMC方法的发展与现代贝叶斯的复兴


---