统计咨询与实践　MCMC 方法

统计 81 于越 *

November 15, 2021

Contents

*学号：2183210516

1 摘要与引言

本篇报告主要介绍了贝叶斯统计中较为被广泛应用的 MCMC(Markov Chain Monte Carlo) 方法。该方法是在 Monte-Carlo 模拟的基础上，保证贝叶斯统计中的先验分布与后验分布模拟参数的结果形成具有较好性质的 Markov Chain 的情况下，进行参数估计以及抽样的统计方法。本文主要对 MCMC 中较为著名的一些方法，例如 Gibbs 抽样，两种 Metropolis-Hastings 算法进行了原理上的介绍以及数据实验，并且得到了一些相关结论。

首先，我对于 MCMC 方法的原理进行了一定程度上的简介，将传统的贝叶斯统计与 MCMC 方法进行了比较，发现 MCMC 方法在传统贝叶斯统计的基础上，可以保证分布的收敛性质，由此保证平均值近似的合理性。与此同时，为了更好地保证得到的收敛结果为一平稳分布，在 MCMC 方法中通常“燃烧”掉前一定数量的迭代值，由此尽量取避免初值的影响。

接着，我介绍了 Gibbs 抽样的原理以及实现步骤。Gibbs 抽样是最为简单的 MCMC 算法，其主要思想在于推导出先验分布与后验分布之间的关系，以该关系作为纽带不断生成下一步的参数迭代值，重复该过程直到目标分布收敛为止。在此我主要用一维正态分布数据集进行实验，对于均值与方差的参数分别构造出正态分布与倒 Gamma 分布作为先验分布进行抽样实验。在进行 50000 次迭代后，可以得到原正态分布的参数估计值。对于该实验我还探讨了其抽样误差的影响因素，发现先验分布的标准差会很大程度上影响抽样的误差结果，先验分布的标准差越大，抽样误差也越大。对于二维分布我也进行了正态分布的抽样实验，发现参数估计的结果非常成功。

对于 Metropolis-Hastings 算法，我首先介绍了 Metropolis-Hastings 算法的基本原理。此处涉及到建议分布与接受概率的概念，其基本原理是，每一步迭代过程中，需要建议分布生成下一个参数迭代的“候选值”，接着通过建议分布与目标后验分布结合计算出接受参数迭代“候选值”的概率，再依照该概率值构造伯努利分布决定下一步迭代值的选择，是仍旧遵照上一步迭代值还是转换为建议分布所生成的“候选值”。接着主要探究了两种 Metropolis-Hastings 算法，随机游走 Metropolis-Hastings 算法与独立抽样 Metropolis-Hastings 算法。这两种算法的主要区别在于建议分布的选择上。

对于随机游走 Metropolis-Hastings 算法，其主要考虑建议分布为具有较好性质的分布，具有一定区间上的对称性，因此此时构造的接受概率的形式完全由后验目标分布决定；而对于独立抽样 Metropolis-Hastings 算法，我们需要另外通过二阶导数近似的方法构造出一个建议分布协助算法进行下一步的迭代，因此计算更为复杂，但是其精度更高。在此处，对于随机游走 Metropolis-Hastings 算法，我主要探讨了建议分布的方差对于算法性能的影响，利用 Monte-Carlo 误差、算法“候选值”拒绝次数以及 acf 图作为评判算法性能的指标，发现选择建议分布的方差越大时，随机游走 Metropolis-Hastings 算法的 Monte-Carlo 误差值更大且迭代“候选值”被拒绝次数更多，算法的性能相对更差；同时对于选择目标后验分布为 t 分布情况下对其自由度对算法性能的影响进行了实

验，发现分布自由度越高，迭代“候选值”拒绝次数越多，该迭代算法趋近于无效化。最后发现对于 t 分布为目标后验分布，正态分布为建议分布的情况下，选择 t 分布自由度为 5-10 是较为正确的选择。

对于独立抽样 Metropolis-Hastings 算法，主要对于目标后验分布为混合正态分布的情况进行了实验，对于混合参数 p 进行估计，发现当 p 取区间边缘值时的估计效果要明显更好一些。然后又以一个二项分布模型的例子，将两种 Metropolis-Hastings 算法的性能进行对比，发现此时尽管独立抽样 Metropolis-Hastings 算法的算法复杂度更高，但是其具有估计的 Monte-Carlo 误差更小，参数标准差更小，迭代“候选值”被拒绝次数更少的优势。对于 Metropolis-Hastings 算法，其相比于 Gibbs 抽样方法的一大优势在于其对于复杂分布的刻画与估计效果更好。

最后，我主要调用了 R 语言自带的 bsts 包，作用于一些时间序列数据集，实验了 MCMC 的时间序列分析方法的效果。发现 MCMC 方法分析时间序列数据，可以使得对先有值的拟合与预测结果较好；对于预测未来值的情况，对于具有极端值出现的情况预测效果会差一些。最后我发现 MCMC 方法对于季节性时间序列数据的分析效果很好。这提醒我们在做时间序列分析前，可以将一些特殊的处理方法与 MCMC 方法结合以获得更好的实验结果。

关键词：先验与后验；Metropolis-Hastings 算法；建议分布

2 MCMC 原理简介

[1]

我们已经在数理统计中学习了贝叶斯统计的相关知识。贝叶斯统计需要先验分布与后验分布。当我们给定先验分布 $\pi(\theta)$ 时，后验分布为

$$\pi(\theta|x) = \frac{L(x|\theta)\pi(\theta)}{\int\limits_{\Theta} L(x|\theta)\pi(\theta)d\theta} \tag{1}$$

此时的 $L(x|\theta)$ 为似然函数。

有了后验分布，我们就可以得到 g(θ) 为参数 $theta$ 的函数，可得 g(θ) 的 Bayes 估计为：

$$\hat{g(\theta)} = \frac{\int\limits_{\Theta} g(\theta)L(x|\theta)\pi(\theta)d\theta}{\int\limits_{\Theta} L(x|\theta)\pi(\theta)d\theta} = \int\limits_{\Theta} g(\theta)\pi(\theta|x)d\theta = E[g(\theta)|x] \tag{2}$$

例如我们常用的平方损失下 θ 的 Bayes 估计：

$$\hat{\theta} = \frac{\int\limits_{\Theta} \theta L(x|\theta)\pi(\theta)d\theta}{\int\limits_{\Theta} L(x|\theta)\pi(\theta)d\theta} = E[\theta|x] \tag{3}$$

但是 Bayes 估计为一积分的形式，除非是一些较为简单的分布情形，否则后验分布的计算是十分困难的。而且在数据量较大的情况下，我们更多地采用数值积分的方式去估计结果。这些数值积分中最经典且最为广泛使用的便是 Monte-Carlo 方法。

就例如此时的后验 Bayes 估计 $\hat{g(\theta)}$，我们可以采用平均值近似的方法求得该后验 Bayes 估计：

平均值近似：

$$\bar{g} = \frac{1}{s}\sum_{i=1}^{s} g(\theta^{(i)}) \qquad i = 1, 2, \cdots s \tag{4}$$

其中 $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(s)}$ 为来自后验分布 $\pi(\theta|x)$ 的容量为 s 的样本。

该方法能够用于进行后验 Bayes 估计的理论保证为大数定律。只要样本容量 s 是足够大的，样本均值 $\bar{g}$ 可以依概率收敛于 E[g(θ)|x]. 当 s 足够大时，估计的精度可以达到我们所想要的任意精度。

事实上，此时需要我们抽取出的这 s 个样本是相互独立的。这是大数定律能够成立的一大保证。但是在一些问题中，我们想要从后验分布 $\pi(\theta|x)$ 中抽取出一系列的独立样本的难度是十分大的。因此我们需要改变思路。我们可以通过抽取马氏链的方式以达到类似的结果。

马氏链具有平稳性，马氏性等等优秀的性质。就例如：

[1]涉及原理的内容部分参考《贝叶斯统计学及其应用》，同济大学出版社，韩明编著

马尔科夫链随机过程 $\{\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(T)}\}$ 马氏性:

$$f(\theta^{(t+1)}|\theta^{(t)}, \cdots, \theta^{(1)}) = f(\theta^{(t+1)}|\theta^{(t)}) \tag{5}$$

有了这些性质，我们就可以把这条马氏链上的在不同状态下的值给近似看成一些”非独立” 样本，此时的效果与从后验分布 $\pi(\theta|x)$ 中抽取独立样本的作用类似。因此我们基于马尔科夫链，进而进行 Monte-Carlo 方法，也就形成了 MCMC(Markov Chain Monte-Carlo) 方法的雏形。

对于一条马尔科夫链，如果我们无法保证前一个值和后一个值之间不存在任何的相关性，我们也可以采用间断式采样的形式以保证采样之间的相关性。

事实上，如果我们得到的 Markov Chain 是不可约的，非周期的且是正向发生的，则当 $t \to \infty$ 时，θ^t 的分布收敛于平稳分布，由此我们可以近似地对 θ 进行某些程度上的推断。

MCMC 方法的总览步骤如下所示：

MCMC 算法步骤总览：

(1) 选择初始值 $\theta^{(1)}$.

(2) 生成 T 个样本值，直到收敛到均衡分布。

(3) 诊断样本收敛性，若不收敛说明需要更多的样本。

(4) 删除开始的 B 个模拟值

(5) 对样本 $\{\theta^{(B+1)}, \theta^{(B+2)}, \cdots, \theta^{(T)}\}$ 进行后验分析，进而得到后验分布的描述性统计量，画出后验分布图。

(6) 在得到后验分布的描述性统计量，例如均值、标准差、分位数和相关系数等后进一步分析参数和样本。

值得一提的是，为了使得算法收敛更快，我们可以选择一个较好的样本链的初始值。例如选择后验分布中的众数，先验分布的均值等等，这些都是较好的选择。

而为了避免我们所选择的初值的影响，我们会把得到的样本链的前 B 个样本删除。这里 B 的选择是依赖样本量的。事实上，样本量越大，删除样本值对后验分布推断的影响越小。

3 Gibbs 抽样

3.1 原理与步骤介绍

Gibbs 抽样是最简单且最直观的 MCMC 方法。该抽样方法将 MCMC 与数据添加结合起来，广泛应用于分布的抽样。

我们需要选择参数 θ 的一个初始值为 $\theta^{(0)}$，从而进行迭代。此时的迭代方法需要我们结合之前产生的所有参数值进行迭代计算。例如我们假定第 i 次迭代开始时的参数值为 $\theta^{(i-1)}$.

故我们从满足条件分布 $\pi(\theta_{(j)}|x,\theta_{(1)}^{(i-1)},\theta_{(2)}^{(i-1)},\cdots,\theta_{(j-1)}^{(i-1)},\theta_{(j+1)}^{(i-1)},\cdots,\theta_{(k)}^{(i-1)})$ 中依次抽取一个样本 $\theta_{(j)}^{(i)}$, 其中 $j=1,2,\cdots,k$.

接着对于 $i=1,2,\cdots,n$ 依次重复进行这些步骤,进而就得到了样本 $\theta^{(1)},\theta^{(2)},\cdots,\theta^{(n)}$. 接着我们遵照之前的 MCMC 算法，可以燃烧掉一部分的初始值，进而就得到了满足这些条件分布的抽样。我们也可以依据得到的 $\theta^{(i)}$ 样本对于参数 θ 进行统计推断。

Gibbs 抽样的具体步骤如下所示：

(1) 设定初始值 $\theta^{(0)}$.

(2) 对于 t=1,2,$\cdots$,T 重复以下步骤：

设定 $\theta=\theta^{(t-1)}$.

由 $\theta_j\sim f(\theta_j|y,\theta_{(1)},\theta_{(2)},\cdots,\theta_{(k)})$ 可生成下一步迭代的对应参数 θ_j.

最后我们可得 $\theta^{(t)}=\theta$，并将其作为第 t+1 次迭代生成的值。

3.2 具体实验抽样——一维

[2]

在此处我们主要进行一维和二维分布的 Gibbs 抽样实验。主要是为了可视化的方便，可以更有利于我们观察 Gibbs 抽样本身。

对于一维的 Gibbs 抽样，我们主要采用正态分布的抽样方法进行抽样，这是因为正态分布为一共轭先验分布，可以简化实验程序的设计。

对于一组正态分布数据，我们可以假设该正态分布的均值参数的先验分布为正态分布，而方差参数的先验分布为倒伽马分布:

$$\mu\sim N(\mu_0,\sigma_0^2),\qquad \sigma^2\sim IG(a_0,b_0) \tag{6}$$

依据 Gibbs 抽样的原理，我们从条件分布 $f(\mu|\sigma^2,y)$ 与 $f(\sigma^2|\mu,y)$ 生成样本。经过计算可得条件后验分布为：

$$(\mu|\sigma^2,y)\sim N(w\bar{y}+(1-w)\mu_0,w\frac{\sigma^2}{n})\text{且}w=\frac{\sigma_0^2}{\sigma^2/n+\sigma_0^2} \tag{7}$$

$$(\sigma^2|\mu,y)\sim IG(a_0+\frac{n}{2},b_0+\frac{1}{2}\sum_{i=1}^{n}(y_i-\mu)^2) \tag{8}$$

故对于该正态分布数据集的 Gibbs 抽样具体实现步骤如下所示：

(1) 设定迭代步数 t=1,2,$\cdots$,T.

(2) 设定 $\mu=\mu^{(t-1)},\sigma=\sigma^{(t-1)},\theta=(\mu,\sigma^2)^{\mathrm{T}}$.

(3) 计算 $w=\frac{\sigma_0^2}{\sigma^2/n+\sigma_0^2},m=w\bar{y}+(1-w)\mu_0$ 与 $s^2=w\frac{\sigma^2}{n}$.

(4) 从分布 N(m,s^2) 生成新的迭代参数 $\mu^{(t)}=\mu$.

(5) 接着计算 $a=a_0=\frac{n}{2}$ 与 $b=b_0+\frac{1}{2}\sum_{i=1}^{n}(y_i-\mu)^2$.

[2]此处的实验设计参考文献《贝叶斯统计模型》 中国财经出版传媒集团 经济科学出版社

(6) 从 G(a,b) 生成参数 τ, 故 $\sigma^2 = \frac{1}{\tau}$ 且 $\sigma^{(t)} = \sigma$.

在这里，我们先假定该正态分布数据集的均值为 100，方差为 0.1，进行 GIbbs 抽样实验，进行 50000 次迭代，设定的均值初值为 0，标准差初值为 2，得到的抽样的均值与方差迹图如下图所示：

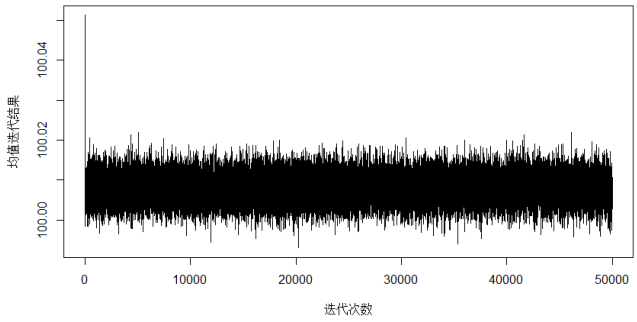


Figure 1: Gibbs 抽样一维均值迹图

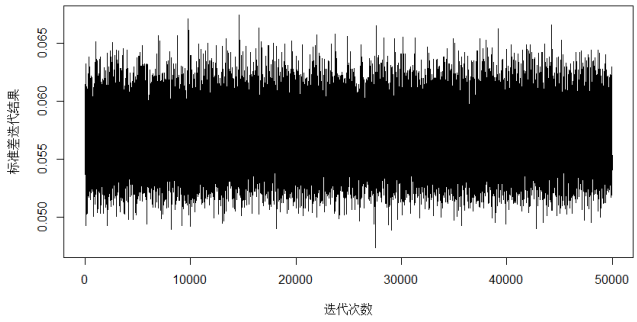


Figure 2: Gibbs 抽样一维方差迹图

可以看到均值的迭代结果在几千次以后就一直在 100 左右波动，而方差的迭代结果一直在 0.06 左右波动。具体地，我们求出均值与方差在 50000 次迭代中的均值进行数值结果比较：

```
> mean(theta[,1])
[1] 100.008
> mean(theta[,2])
[1] 0.05677554
```

Figure 3: Gibbs 抽样一维数值结果

可以看到此时的均值参数估计结果十分好，而标准差迭代就要略差一些。在这里，我认为是因为对于均值而言，此时的先验分布与后验分布是一个共轭分布，因此在不断迭代的过程中，参数的估计值会偏向于向真实值移动，这导致参数估计的结果会稍好一些。

3.3 抽样结果好坏的影响因素

经过一维 Gibbs 抽样的一次实验结果，我们发现尽管我们设定的均值初值与实际均值相差很大，但是在经过大量的迭代步数后，均值的迭代值与真实值十分接近，可以说 MCMC 估计的结果非常好；而尽管我们设定的标准差初值与实际标准差相差不大，但是经过了 50000 次迭代，标准差的迭代值仍旧只有真实值的一半。而且 Gibbs 抽样的误差与什么因素有关？我们依照这一问题进行实验。

在这里，我主要设计实验研究先验分布的标准差对于抽样误差的影响。我构造一个标准差的等差数列，以 0.1 为起始值，10 为末尾值，标准差的变化为每次增加 0.1. 我不断地变化标准差的值，进行 Gibbs 抽样实验，每次实验都经过 20000 次迭代，记录下每次 Gibbs 抽样得到的参数的均值，作为此次 Gibbs 抽样我们可以得到的参数估计值。接着我们对于每次实验的参数估计值与真实值做差求出 MSE 的值，并将 MSE 结果可视化出来，观察实验误差的变化。

我们先更改先验分布的标准差值，对均值进行实验，得出了均值的 MSE 值随标准差变动的图像如下所示：

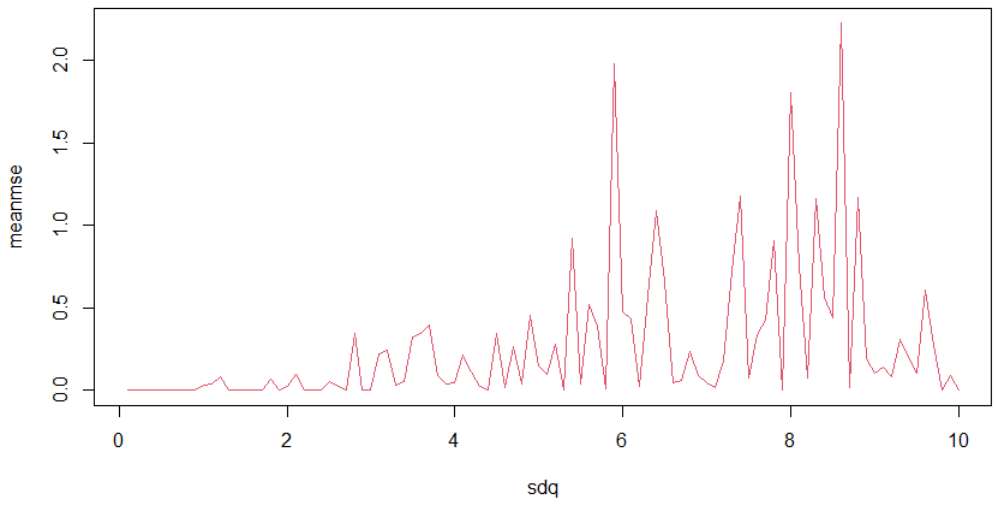


Figure 4: Gibbs 抽样一维均值的 MSE 值随标准差变动

我们发现此时的均值 MSE 呈现一定的波动性，但是并没有什么明显的变化趋势。尤其是在标准差为 6-9 时的波动性非常大，但是在标准差到达 10 时，均值 MSE 又变小了。这说明，均值 MSE 的变动与标准差的变动并没有什么直接关系。

接着我更改先验分布的标准差值，对标准差进行实验，得出了标准差的 MSE 值随标准差变动的图像如下所示：

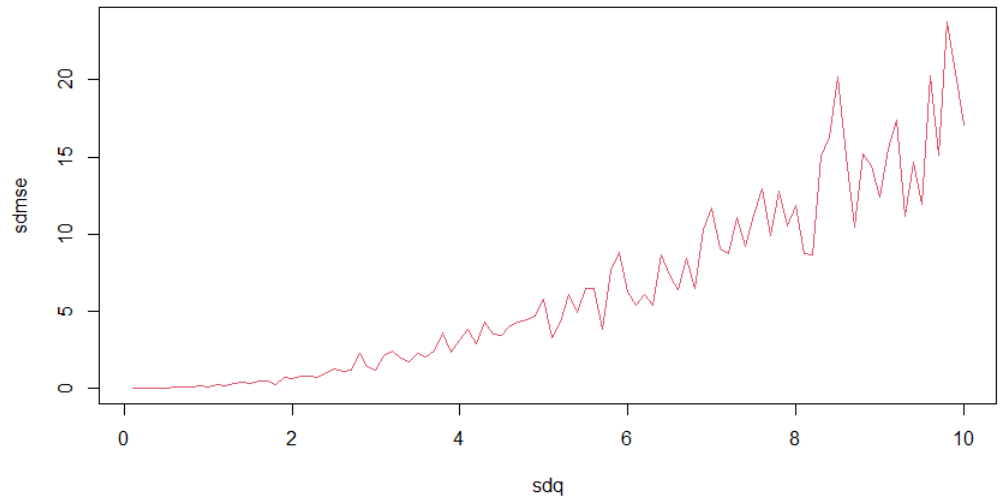


Figure 5: Gibbs 抽样一维标准差的 MSE 值随标准差变动

此时我们发现，标准差的 MSE 也存在着很强的波动性，但是此时的变化趋势却十分明显，即参数估计标准差的 MSE 明显随着先验分布标准差的增大而增大。事实上，这一点是可以预料的。因为先验分布的标准差增大，这表明当我们模拟数据集时，抽样的“抽样池”本身数据波动就较大。故我们抽样得到的结果波动性也会增大，这也就导致了我们经过 20000 次迭代后产生的标准差的误差值也越来越大。而且迭代过程是一个趋近收敛的过程，随着数据集的收敛，误差的大小也愈发难以更改。

3.4 具体实验抽样——二维

经过了前面的一维 Gibbs 抽样，我们在这里继续对二维情况进行实验抽样。为了便于实验的进行与原理的推导计算，我们选择了二维正态分布进行 Gibbs 抽样，观察一下抽样的效果。

选择二维正态分布的优势在于，一方面，二维正态分布的边际分布仍为正态分布，这在一定程度上简化了推导的过程；而且正态分布是一个共轭分布，这使得在形成新分布与抽样时的复杂度更低。

对于二元正态分布的情况，注意到边际分布仍为正态分布。因此进行抽样的条件分布表示为：

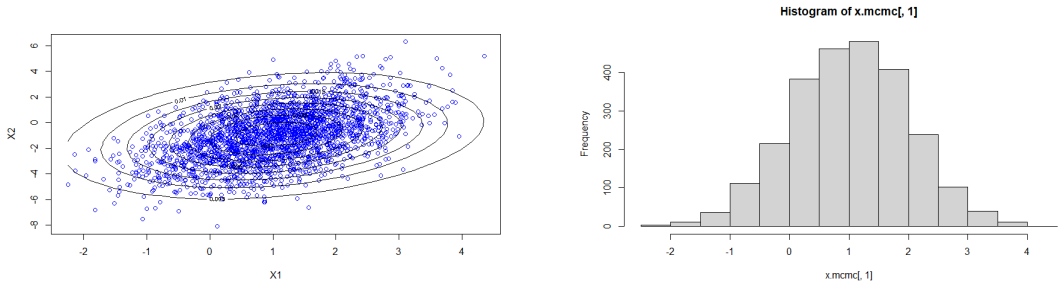
$$f(x_1|x_2) \sim N(\mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1-\rho^2)\sigma_1^2) \tag{9}$$

$$f(x_2|x_1) \sim N(\mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1), (1-\rho^2)\sigma_2^2) \tag{10}$$

此时的 Gibbs 抽样的第 t-1 步到第 t 步迭代过程如下所示：

(1) 设定 $(x_1, x_2) = X^{(t-1)}$.
(2) 由 $f(X_1|x_2)$ 生成样本 $X_1^{(t)}$.
(3) 由 $f(X_2|x_1)$ 生成样本 $X_2^{(t)}$.
(4) 设定 $X^{(t)} = (X_1^{(t)}, X_2^{(t)})$.

我设定二维 Gibbs 抽样的迭代次数为 5000 次，燃烧掉前 1000 次的实验结果，设定二维正态分布的两个边际分布的先验分布初值分别为 N(0,1) 与 N(2,0.5). 我们设定式 (9) 与式 (10) 中的 $\rho = 0.75$. 得到的抽样结果，其中一个维度迭代直方图如下所示：



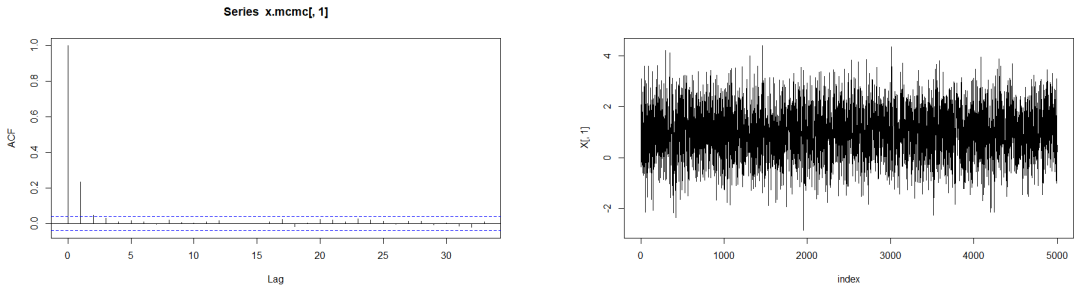
(a) Gibbs 抽样二维结果

(b) Gibbs 抽样二维中第一维度直方图

Figure 6: Gibbs 抽样二维

可以看到抽样结果和二维正态分布的情形确实十分接近。

再给出抽样步骤中产生的 acf 图以及第一维度抽样的迹图如下所示：



(a) Gibbs 抽样二维 acf 图

(b) Gibbs 抽样二维中第一维度迹图

Figure 7: Gibbs 抽样二维

可以发现，acf 图在 lag 项为 3 以后，均在检验水平的范围内，且没有出现拖尾的情况。二迹图也几乎在 1 的位置进行上下波动。这两张图可以说明，此时在迭代 5000 次的过程中，Gibbs 抽样产生的分布早已收敛，这保证了我们抽样过程的严谨性。

接着我们给出分布的均值向量与方差矩阵的数值结果：

```
> colMeans(x)
[1] -0.005515285  2.003460367
> cov(x)
           [,1]       [,2]
[1,]  1.0427265 -0.3924642
[2,] -0.3924642  0.2599256
```

Figure 8: Gibbs 抽样二维的均值标准差数值结果

我所设的数据分布的均值分别为 0 与 2，而标准差为 1 与 0.5. 此时的二维正态分布对于参数的估计效果非常好，可以看到误差是微乎其微的。这归功于对于二维正态分布的实验设计。二维正态分布的边缘分布仍为正态分布以及正态分布的共轭性质推动了较好估计的生成。该实验是较为成功的。

Gibbs 抽样作为最简单但是也是最为广泛应用的 MCMC 方法，仍旧有着广阔的利用价值。此处不做过多赘述。

4 Metropolis-Hastings 算法

前面我们已经介绍了 Gibbs 抽样这一最简单且是最为广泛应用的 MCMC 算法。但是 Gibbs 抽样显然存在着各种各样的问题。可以看到，Gibbs 抽样的适用范围是参数的条件分布容易求出或者是我们相对而言较为熟悉的分布，这样我们才可以从条件分布中去进行抽样，且达到较好的参数估计效果；但是对于后验分布不是常见分布，或者是较为复杂的分布时，用 Gibbs 抽样的效果会十分糟糕，无法起到较好的参数估计效果。因此，需要更为复杂的算法来解答这个问题。

4.1 算法基本原理介绍

Metropolis-Hastings 算法是相对 Gibbs 抽样而言更为复杂的算法，也是一个更为一般化的算法。该算法的基本思想在于，设法找到一个与后验分布相近似的较为简单的分布，作为建议分布，在算法过程中“协助”迭代算法的实现。该算法的目标与大方向与之前所介绍的一致，都是需要形成这一状态到下一状态之间的马氏链，得到一个平稳分布，进而进行后续的抽样与估计操作。

建议分布对于迭代算法的贡献在于生成“候选值”这一方面。对于 Gibbs 抽样方法而言，下一步的迭代值完全取决于先验分布与后验分布之间的关系。我们由先验分布推导得到的后验分布即为其参数值的下一步迭代值。但是这样的操作仅仅适用于较为简单的分布情形。而且这种完全“信任”先验分布的行为在复杂分布的情况下会降低参数估计的准确率，这就导致了参数估计的效果会极为糟糕。因此在 Metropolis-Hastings 算法中，我们给出了建议分布的概念。依据建议分布，我们生成每一次迭代的“候选值”。依据建议分布，我们还可以得到每次迭代之后下一步迭代值的接受概率。此时依据这一接

受概率，我们就可以生成一个伯努利分布，依据这一伯努利分布生成下一个迭代值。也就是说，我们有接受概率这么大的概率会更新下一步迭代值为“候选值”，而又有一定的概率我们要拒绝这一候选值，保持原来的迭代值不变。

值得一提的是，这里的建议分布是依赖于上一步的状态的。例如，我们记上一步的状态为 X_t, 若我们取定了建议分布为正态分布，那么建议分布的均值可以取为上一步状态 X_t, 而方差我们取一个固定值。我们对于建议分布的选取可以灵活任意一些，但是生成的样本链需要满足收敛到平稳分布（目标分布）f 的条件下，以及其他的包括不可约性、非周期性、马氏性等一系列的条件。很多情况下，我们会选择正态分布作为建议分布，这是因为正态分布链可以满足以上条件，且作为最常用的分布，该分布形成的每一步抽样值更有利于我们的数值分析与估计。

下面给出 Metropolis-Hastings 算法的基本流程步骤：

(1) 设定一个初值 θ_0.

(2) 对于迭代步数 t=1,2,$\cdots$,n 重复以下步骤：(假定第 t-1 步迭代到第 t 步迭代)

设定 θ=$\theta^{(t-1)}$.

从建议分布 q($\theta|\theta^{(t-1)}$) 生成新的样本候选值值为 θ^*.

依据建议分布与目标分布计算接受概率：

$$\alpha(\theta;\theta^{(i-1)},\theta^*)=min\{\frac{p(\theta=\theta^*|\theta)q(\theta=\theta^{(i-1)};\theta^*)}{p(\theta=\theta^{(i-1)}|\theta)q(\theta=\theta^{(*)};\theta^{(i-1)})},1\} \tag{11}$$

(3) 我们以概率 α 选取样本 $\theta^{(i)}=\theta^*$，而以概率 1-α 接受 $\theta^{(i)}=\theta^{(i-1)}$.

(4) 依次得到 $\theta_{(1)},\theta_{(2)},\cdots,\theta^{(n)}$，再燃烧到开始的 m 项，求均值可得

$$E[g(\hat{\theta})|x]=\frac{1}{n-m}\sum_{i=m+1}^{n}g(\theta^{(i)}) \tag{12}$$

从最后的估计结果，我们知道该算法最重要的步骤在于如何选取建议分布以及计算接受概率。对于如何选取建议分布的这一至关重要的问题，经过研究人员常年研究，得到了以下两种选择建议分布的方法，也因此衍生出了两种最为著名的 Metropolis-Hastings 算法。

4.2 随机游走 Metropolis-Hastings 算法

4.2.1 基本原理推导

这一算法可以算是最为简单的 Metropolis-Hastings 算法。当 Metropolis 最初研究这一算法时，为了保证建议分布的良好性质，他建议选择的密度函数 $q(\theta|\theta^{(t-1)})$ 为 $g(\theta-\theta^{(t-1)})$，其中的 g 为一个在 0 值附近堆成的密度函数，即满足 g(-t)=g(t). 这就是最早的 Metropolis 算法，主要考虑对称结构的建议分布，即 $q(\theta|\theta^{(t-1)})=q(\theta^{(t-1)}|\theta)$. 而当建议分布满足这种情况时，我们代入接受概率的计算式:

$$\alpha(\theta;\theta^{(i-1)},\theta^*)=min\{\frac{p(\theta=\theta^*|\theta)q(\theta=\theta^{(i-1)};\theta^*)}{p(\theta=\theta^{(i-1)}|\theta)q(\theta=\theta^{(*)};\theta^{(i-1)})},1\} \tag{13}$$

就显而易见地发现，此时的接受概率是仅仅依赖于后验分布 $p(\theta=\theta^*|x)$ 的。故此时的接受概率我们可以直接更新为

$$\alpha(\theta;\theta^{(i-1)},\theta^*)=min\{\frac{p(\theta=\theta^*|\theta)}{p(\theta=\theta^{(i-1)}|\theta)},1\} \tag{14}$$

此时的接受概率形式是十分简单的。这在一定程度上为我的实验提供了较大的方便。

值得一提的是，我们可以选择常用的建议分布为一个多元正态分布

$$q(\theta^*|\theta)=N_d(\theta,\bar{S_\theta}) \tag{15}$$

此处的 d 是参数 θ 的维数。我们都知道，在正态分布中的方差矩阵可以用来控制分布的分散情况，在这里，依据分布的分散情况，我们也就可以控制收敛速度的大小[3]。方差矩阵可以用来控制分布的分散情况，在一定程度上也可以由对角线值决定建议分布给出的值与现有值的接近程度。如果协方差矩阵的对角线值较小，就会相对地产生较高的接受概率，但是也会因此减缓收敛的速度。故此时就需要较多的迭代次数来对于参数空间进行合理的估计。但反之，当协方差矩阵的对角线值较大时，尽管此时每一步生成的参数空间是较为分散的，可以加快参数的收敛速度，但是此时的每一步迭代中产生的接受概率会较低，这样就变相导致了大量的模拟样本值停留在了相同的值上，此时对于参数空间的移步探索过程就是缓慢的，甚至难以对参数空间进行拓展式的探索。与此同时，由于数据的分散性，我们很难产生出不相关的样本，因为此时就算是距离上相对较远的样本也有很大的可能性是来自于同一分布。因此，对于建议分布的协方差的探索与选择就是至关重要的。

为了构建出有效的 Metropolis-Hastings 算法，我们可以使得建议分布与后验分布的协方差矩阵保持一致性。如上所述，随机游走 Metropolis-Hastings 算法是不需要取建议分布的，因此我们在研究协方差的影响时，可以转而研究后验分布本身对于该算法的影响，这是由建议分布与后验分布的协方差矩阵的一致性所保证的。因此，后验分布本身也就成了随机游走 Metropolis-Hastings 算法的重中之重。

接下来我们就影响随机游走 Metropolis-Hastings 算法性能的因素进行一定程度上的研究。在这里，为了防止共轭分布的存在而导致实验结果出现一些较为巧合的情况，我们选择用随机游走 Metropolis-Hastings 算法模拟生成一组自由度为 v 的 t 分布随机变量。而此处我们采用的建议分布为正态分布 $N(X^{(t)},\sigma^2)$. 我们先探讨选择的建议分布 (正态分布) 的标准差 sigma 对于该算法抽样结果的影响。此时的每一步迭代的接受概率形式如下所示：

[3]http://www.cnki.com.cn/Article/CJFDTotal-LSZJ201102020.htm

$$\alpha(\theta; x^t, y) = min\{\frac{p(\theta = \theta^*|y)}{p(\theta = \theta^{(i-1)}|x^t)}, 1\} = \frac{f(Y)}{f(X^t)} = \frac{(1+y^2)^{-(v+1)/2}}{(1+(x^{(t)})^2)^{-(v+1)/2}} \tag{16}$$

4.2.2 实验一：建议分布方差对于算法影响

在这里，我们取定我们所选择的 t 分布自由度为 5 进行实验。此时的 t 分布既厚尾性不会特别严重，因此不会导致抽样时极端值出现的情况过于明显，又不会导致分布过于趋近于正态分布，而导致我实验的先验分布与后验分布形成共轭分布进而导致实验结果出现一定程度的巧合 (防止共轭分布使得参数估计较好的情况成为巧合)。我们先研究自由度固定时，建议分布 (此处主要为正态分布) 的不同标准差值对于随机游走 Metropolis-Hastings 算法抽样结果的影响。

我们主要选择以下几个指标作为随机游走 Metropolis-Hastings 算法性能的评判标准：

(1) 迭代步中被拒绝 (被接受) 次数。鉴于在 Metropolis-Hastings 算法中，每一次的迭代都需要计算出接受概率的值来判断下一个迭代值是保持原样还是继续更新，因此我用迭代步的被拒绝次数去判定此时的算法性能好坏。如果算法迭代过程中，迭代更新被拒绝的次数过多，这会导致参数的更新实际上是停滞不前的，也就是说，参数的实际迭代步很少，迭代前进效率很低，参数经过多步迭代实则还在初值附近，故此时的算法迭代与估计的性能很差。

(2) 关于参数结果的 acf 图。acf 图是一种在时间序列中用来判定链值的前项与后项之间是否存在相关关系的工具。acf 图的每一个 Lag 项的意义为，每一项与其后 lag 滞后项的对应项之间进行相关性的假设检验，acf 值实际上可以理解为相关性检验的对应 p 值。当 p 值在 0.05 范围内时，我们需要拒绝原假设，认为此时该项与其后的 Lag 项之间不存在相关性关系；反之，我们认为前项与其滞后项之间存在着相关关系。显然存在着相关关系的两项之间，我们很难将其认为是一条“独立”的样本链，此时的 Metropolis-Hastings 算法的抽样结果与估计结果显然不会太好。因此我们需要得到的 acf 值随着 lag 项的增加逐渐减小至正负置信水平之间，且不会存在着拖尾的情况出现。

(3)Monte-Carlo 误差。Monte-Carlo 误差是一种在 Monte-Carlo 方法中估计误差的方式，可以用该值来判断 Monte-Carlo 模拟的性能以及模拟样本的可变性。Monte-Carlo 误差较小时，参数估计的准确性较高。依据大数定律，理论的情况是，模拟的次数越多，Monte-Carlo 的误差值越小，模型的估计准确度显然也越高。

我们常用的估计 Monte-Carlo 误差的方法是批均值法。该方法的特点在于需要将模拟出的样本值分为 K 个批次，对于每一批次的情况，计算参数的估计量 $g(\theta)$ 的后验均值的 Monte-Carlo 误差，进而用平均值估计的方法计算出总体误差的值，进而通过批均值的标准差进行估计。具体的做法如下所示：

1、将模拟样本分为 K 个批次，每个批次的样本数位 $v = \frac{T}{K}$.

2、对于后验分布 g(θ), 首先计算每批均值 $g(\bar{\theta})_b$:

$$g(\bar{\theta})_b = \frac{1}{v} \sum_{t=(b-1)v+1}^{bv} g(\theta^{(t)}) \qquad b = 1, 2, \cdots, K \tag{17}$$

3、由此可得总体均值误差为

$$g(\bar{\theta}) = \frac{1}{T} \sum_{t=1}^{T} g(\theta^{(t)}) = \frac{1}{K} \sum_{b=1}^{K} g(\bar{\theta})_b \tag{18}$$

4、通过对批均值的标准差进行估计得到 Monte Carlo 误差的估计值：

$$MCE[g(\theta)] = SE[\hat{g}(\bar{\theta}] = \sqrt{\frac{1}{K(K-1)} \sum_{b=1}^{K} (g(\bar{\theta})_b - g(\bar{\theta}))^2} \tag{19}$$

批均值法估计 Monte-Carlo 误差是一种较为容易实施的方法。尽管其准确性与精度没有达到特别高的地步，但是其可操作性还是获得了一致认可。

我先设定建议分布为正态分布，且该正态分布的标准差分别取定为 0.05,0.5,4,20 进行实验。实验过程中，对于每一种建议分布 (正态分布) 的情况，我们都进行迭代 10000 次进行实验，对每次的结果都做出了轨迹图以及 acf 图，计算出了每次实验的被拒绝次数以及 Monte-Carlo 误差。得到的轨迹图分别如下图所示：

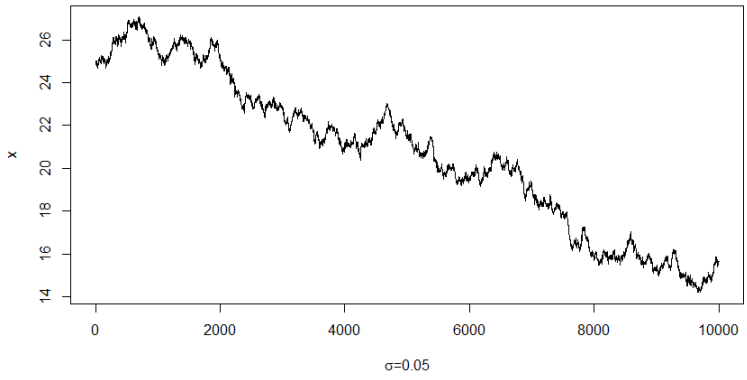


Figure 9: 随机游走 Metropolis-Hastings 方差为 0.05 轨迹图

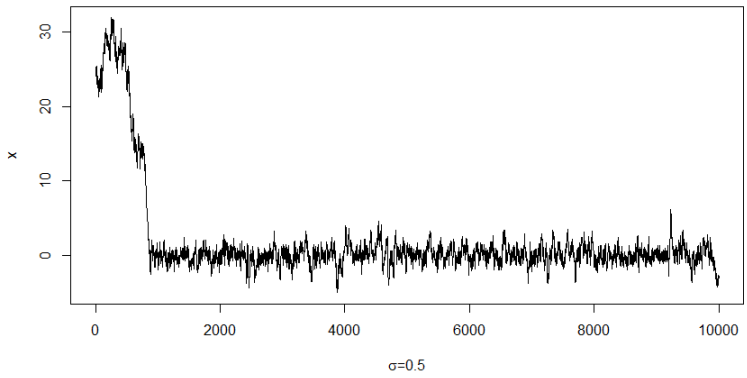


Figure 10: 随机游走 Metropolis-Hastings 方差为 0.5 轨迹图

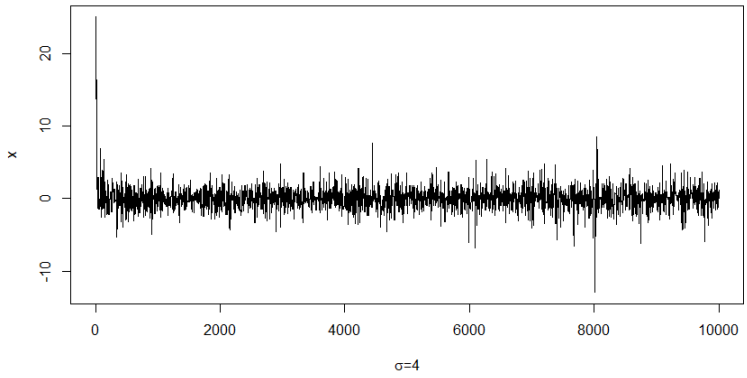


Figure 11: 随机游走 Metropolis-Hastings 方差为 4 轨迹图

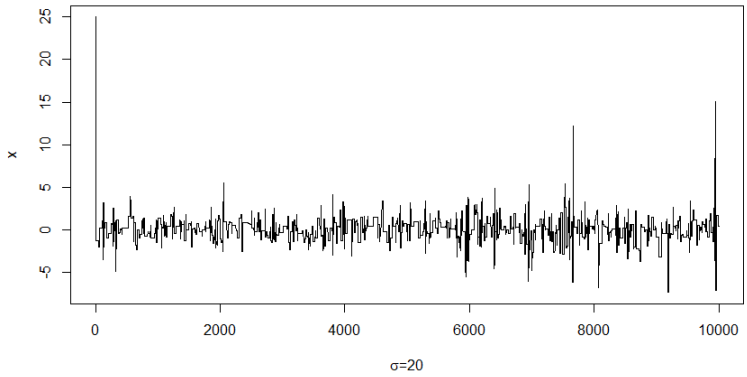
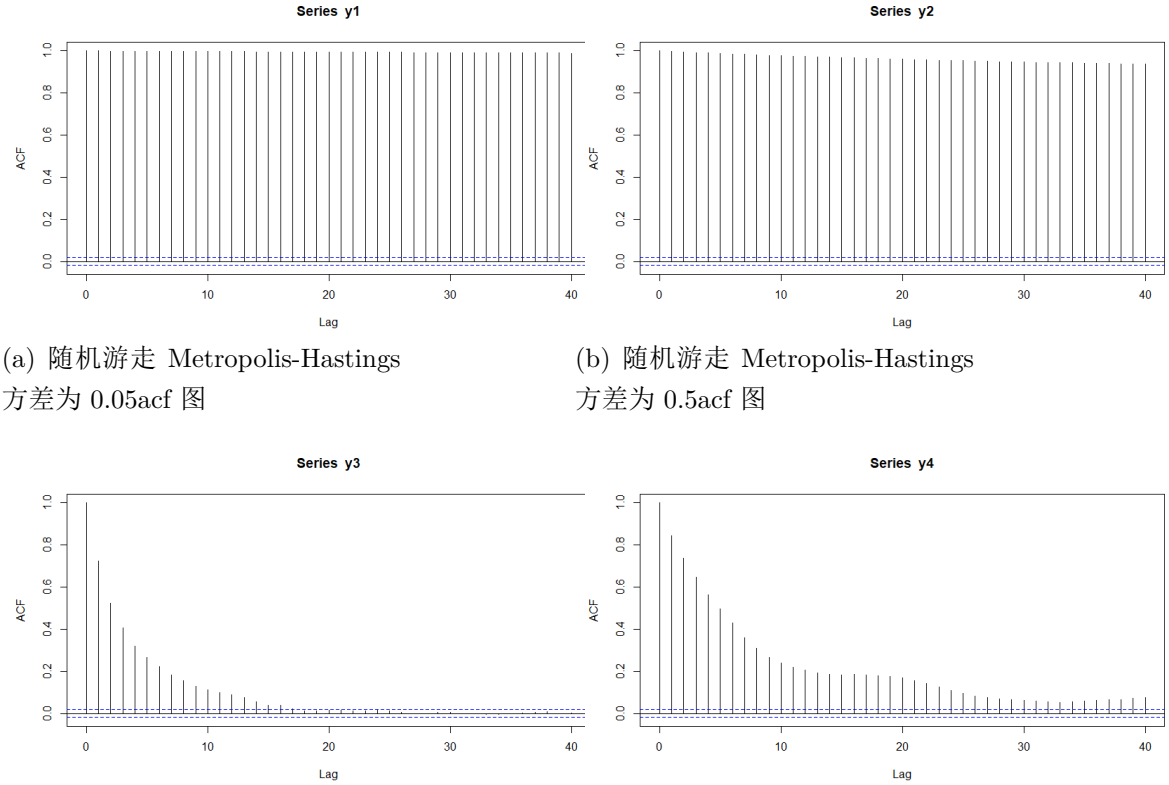


Figure 12: 随机游走 Metropolis-Hastings 方差为 20 轨迹图

从轨迹图的结果我们可以看到，建议分布的方差越大，抽样可以在越少的步数情况下达到收敛的情况。可以看到当我取 σ^2=0.05 时，迭代了 10000 步的情况下，数据链仍

旧没有收敛。且对于 5000-10000 次迭代的结果值进行观察，不难发现，建议分布的方差越大时，每次抽样结果的波动性越小。

但是这并不就代表方差取得越大，用随机游走 Metropolis-Hastings 算法抽样的结果就越好。做出每次实验我们得到的 acf 图进行分析，如下所示：



(a) 随机游走 Metropolis-Hastings 方差为 0.05acf 图

(b) 随机游走 Metropolis-Hastings 方差为 0.5acf 图

(c) 随机游走 Metropolis-Hastings 方差为 4acf 图

(d) 随机游走 Metropolis-Hastings 方差为 20acf 图

Figure 13: 随机游走 Metropolis-Hastingsacf 图

从 acf 图我们可以明显看出，对于 σ^2=0.05 与 0.5 的情况，得到的样本模拟值与其滞后 40 项仍旧存在着极强的相关性，此时我模拟出的样本数值结果相关性过强，无法达到所要求的“独立”样本链的要求，且此时最后样本值的收敛情况可能十分糟糕，样本链无法收敛到平稳分布，故此时的方差较小的两种情况我认为实验结果不好，此时无法进行进一步数值模拟与分析；而对于方差为 20 的情况，可以看到此时的 acf 图呈明显的下降趋势，但是在滞后项到达 40 项时，原本应该在 lag 项为 35 左右时接近拒绝相关性检验假设的 p 值又进一步增大了，这说明此时的样本链中前项与后项之间的相关性还是过强，此时的样本链收敛情况尽管比前两种情况好了很多，但是从实验结果看来，参数估计还是很难进一步进行。而对于方差为 4 的情况，我们可以很明显地发现，此时的样本链是一个短记忆性的情况，在 Lag 项达到 16 左右时，acf 图很明显地将值落在了置信水平范围内，这说明此时的前项与后项之间可以被认为没有任何的相关性，这就比较符合我们对于“独立”样本链的描述。与此同时，这时的样本链收敛情况也较好，可以被认为是收敛到了一个平稳分布上。

接着我们给出了在取不同方差的 4 次实验中，对于 10000 次迭代中的具体拒绝迭代次数，结果如下所示：

```
> print(c(rw1$k,rw2$k,rw3$k,rw4$k))
[1]   54 1346 6720 9232
```

Figure 14: 随机游走 Metropolis-Hastings4 次实验迭代拒绝次数

可以看到，随着模拟实验的建议分布的方差值增大，10000 次迭代中拒绝选择“候选值”的次数增多。事实上这一点可以通过解析式进行分析。在该实验中，依据所设定的 t(5) 的原分布以及正态分布的建议分布，我们得到的接受概率形式如下所示：

$$\alpha(\theta; x^t, y) = min\{\frac{p(\theta = \theta^*|y)}{p(\theta = \theta^{(i-1)}|x^t)}, 1\} = \frac{f(Y)}{f(X^t)} = \frac{(1+y^2)^{-(v+1)/2}}{(1+(x^{(t)})^2)^{-(v+1)/2}} \tag{20}$$

当我选定的建议分布的方差值越来越大时，分布抽样就越容易取到在分布中尾部或者说边缘部的值。这时，候选值 y 的平方值就越大，作用到该式中，会导致接受概率的分子越小，即接受概率值越小。故在构造后续的拒绝/接受“候选值”的伯努利分布中，拒绝候选值的概率会愈发增大。

可以看到，在 10000 次迭代中，对于 σ^2=20 的情况下，尽管观察 figure12 的第四幅图我们可以发现数据链很快就收敛，但是由于迭代过程中的拒绝次数过多，因此此时我们可以认为该算法对于这一参数空间的探索是不够到位的，故此时的估计方法也不够有效。

最后我们给出 4 次实验的在不同方差下进行随机游走 Metropolis-Hastings 算法估计通过批均值法估计得到的 Monte-Carlo 误差值如下所示：

```
> mcerror.batch(y1,batches=50)
[1] 0.5163141
> mcerror.batch(y2,batches=50)
[1] 0.9265999
> mcerror.batch(y3,batches=50)
[1] 0.04400888
> mcerror.batch(y4,batches=50)
[1] 0.06816168
```

Figure 15: 随机游走 Metropolis-Hastings4 次实验 Monte-Carlo 误差值

可以很明显地发现取 σ^2=4 时，估计得到的 Monte-Carlo 误差值结果最小。而且我们综合前面的几项指标，发现取 σ^2=4 时，样本数据链也得以很快收敛，而且 10000 次迭代中被拒绝“候选值”的次数也没有达到十分夸张的 90% 都需要拒绝，而且样本链的 acf 值在 lag 项达到 16 左右时，就可以保证数据的前项与后项之间的相关性近乎不存在，确保了样本链的“独立性”。由此，我们进行横向比较，发现在该实验中取建议分布的方差值为 4 左右是较为合适的。

4.2.3 实验二：“遍历性”实验——建议分布的 σ 值

在经历了上述实验中，我们经过 4 次实验的比较得到了一些数值结果，给予不同的指标给出了一个看上去结果还不错的建议分布。但是显然，影响随机游走 Metropolis-Hastings 算法估计性能好坏的因素肯定不止这一个，而且我们刚才对于建议分布方差大小的选择探讨也仅仅取了 4 个值，所以通过这极为有限的取值个数就断定出建议分布的选择较好这显然是十分武断的。因此我设定了一个方差值的区间，将区间分成了一些小的区段，对于小的区段的每个端点值进行实验，并且综合比较算法性能的各项指标，争取找到建议分布的方差值与算法性能、估计好坏之间的关系。

在这里，我设定 t 分布的自由度为 4，设定建议分布的 sigma 变化值为从 0.05 开始，以 100 结束，每次增加 0.05，以此进行实验。由此我们需要进行 2000 次实验。

在此处，为了防止实验时间过长，每一次实验我都选择了进行 2000 次迭代，进行结果的比较。一方面这时因为正如之前的实验一所示，当建议分布的 sigma 值够大时，经历少量的迭代步数，样本数据链就可以达到收敛的效果了。因此相对地，我们为了实验时间不至于过长又可以使得实验准确率不至于过低的情况下，我们选择一次实验进行 2000 次迭代，进而进行后续的比较。

在此处实验，我还是选择了 acf 值、Monte-Carlo 误差以及迭代过程中的被拒绝次数作为评判在不同的建议分布下随机游走 Metropolis-Hastings 算法性能好坏的依据。在进行 2000 次实验后，我们得到的结果进行可视化，可以得到看上去近似“连续”的图像。接下来进行依次的分析。

首先是 2000 次实验中，不同的建议分布的标准差值选择与 Monte-Carlo 误差之间的关系：

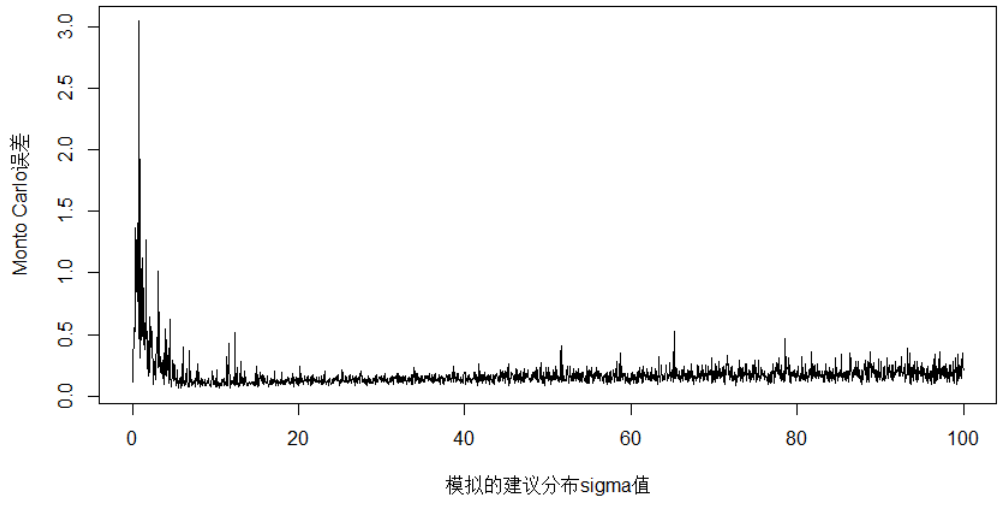


Figure 16: 随机游走 Metropolis-Hastings 实验二 Monte-Carlo 误差值

通过图像可以明显地发现，随着建议分布的 σ 值的不断增大，当 σ 值位于 0 到 10

之间时，Monte-Carlo 误差值不断减小。但继续随着 σ 值的不断增大，Monte-Carlo 误差值就趋于平稳了，不会产生特别大的波动。

接着是 2000 次实验中，不同的建议分布的标准差值选择与 2000 次迭代中“候选值”被拒绝次数之间的关系：

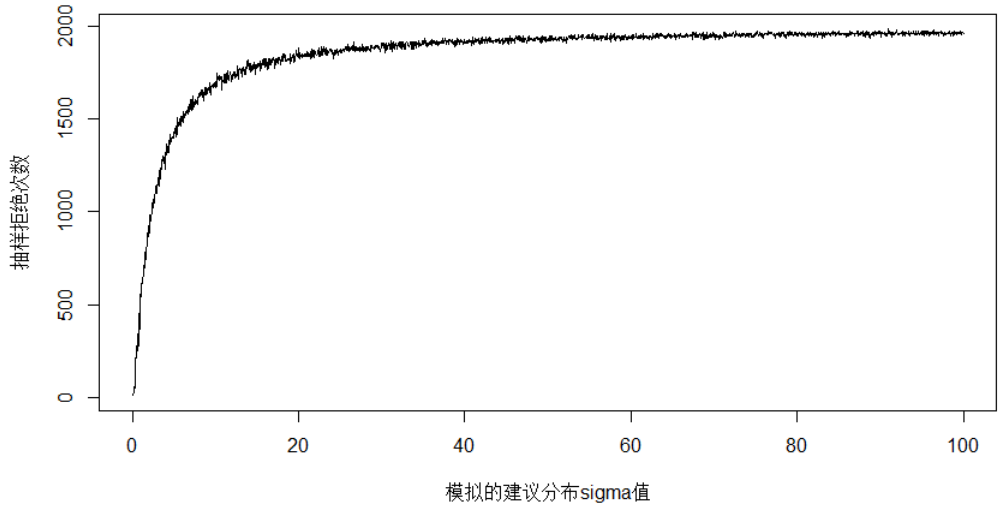


Figure 17: 随机游走 Metropolis-Hastings 实验二“候选值”被拒绝次数

通过图像，我们发现，随着每次模拟实验的建议分布的 σ 值不断增大，在每次实验中的 2000 次迭代中“候选值”被拒绝次数逐渐增加。在 σ 值逐渐增大到 100 时，抽样拒绝次数几乎达到了 2000。因此，“候选值”被拒绝的次数是随着 σ 的值不断增大而不断增大的。

最后我们给出随着模拟实验的建议分布的 σ 值不断增大实验得到样本链的平稳情况的图像。因为此时涉及到 2000 次实验，我们每次都得到样本数据链的 acf 图显然是不切实际的，因此在这里，我采用了 adf 单位根检验的手段。我记录下每次实验得到的对于样本数据链 adf 单位根检验 p 值，将其可视化在图像上：

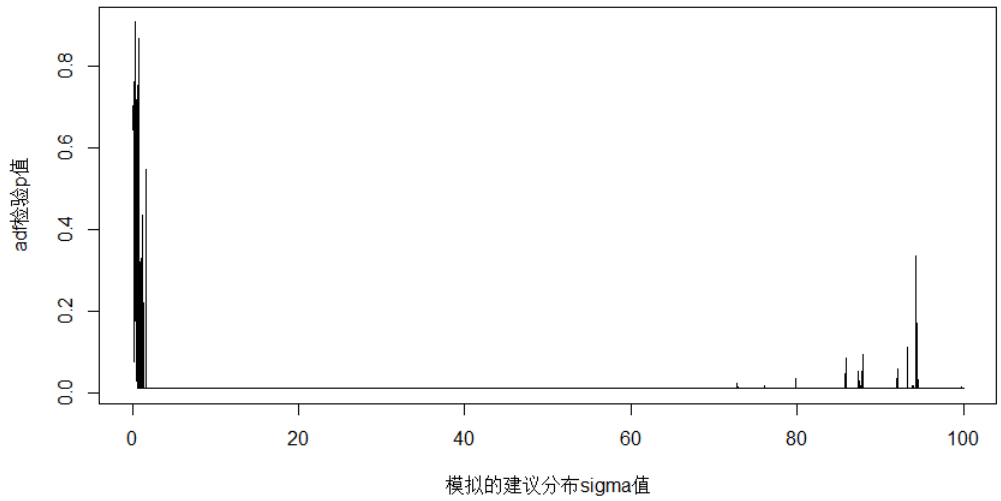


Figure 18: 随机游走 Metropolis-Hastings 实验二 adf 单位根检验 p 值

可以看到此时的图像的横纵坐标就没有特别大的关系了。当我们取建议分布的 σ 值位于 3-80 之间时，几乎都是 p 值小于检验显著水平的情况，都需要拒绝原假设，即此时的数据链不存在单位根，数据链本身已经是平稳的。

综上，我们得知，随着建议分布的 σ 值的增大，在总迭代次数中得到的“候选值”被拒绝的次数会逐次增多，而 Monte-Carlo 误差值会在前段逐渐减小，而后保持一个近乎平稳的值上下波动。依据这里的三幅图像，可以看到选择建议分布的 σ 值在 5-10 之间是较为合理的，因为此时的 Monte-Carlo 误差值刚刚达到平稳值 (也是近似的最小值)，adf 单位根检验得到的结果是平稳序列，且“候选值”被拒绝次数也没有特别多以至于抽样方法的有效性无法得到保证。

4.2.4 实验三：“遍历性”实验——目标后验分布自由度

在探究完建议分布的 σ 值对于随机游走 Metropolis-Hastings 算法性能的影响后，我们自然而然地会考虑除了建议分布，目标后验分布本身对于随机游走 Metropolis-Hastings 算法性能会不会产生影响。因此对于这一问题，我设计了如下实验：

在这里，对于目标后验分布本身，我主要探究这里的目标后验分布 (t 分布) 的自由度对于随机游走 Metropolis-Hastings 性能的影响。对于作为后验分布的 t 分布，我选择了自由度的变动区间为 1-200 进行实验。这样我一共需要进行 200 次实验，每次实验迭代 2000 次。还是用之前的方法得到图像进行分析。

首先是 200 次实验中，不同的建议分布的标准差值选择与 Monte-Carlo 误差之间的关系：

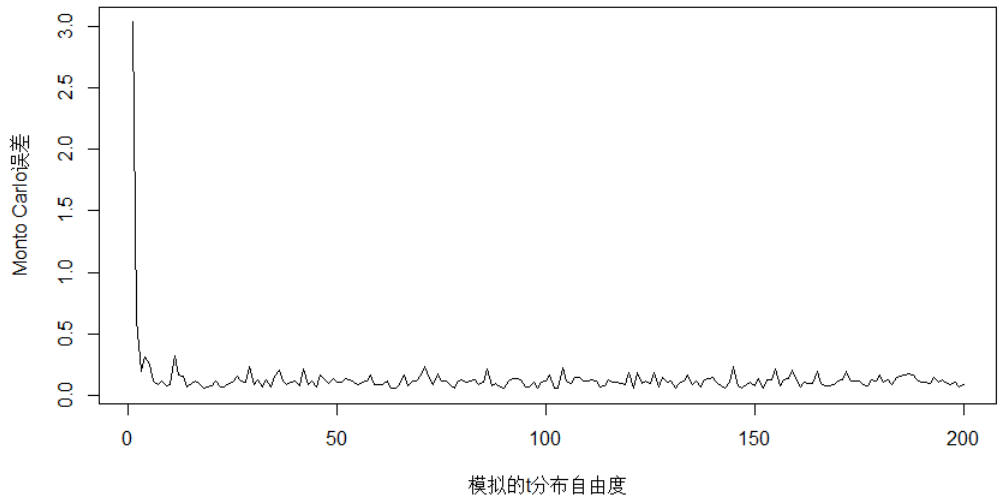


Figure 19: 随机游走 Metropolis-Hastings 实验三 Monte-Carlo 误差值

通过图像可以明显地发现，随着后验分布的自由度的不断增大，当自由度位于 0 到 5 之间时，Monte-Carlo 误差值不断减小。但继续随着自由度值的不断增大，Monte-Carlo 误差值就趋于平稳了，不会产生特别大的波动。

接着是 200 次实验中，不同的建议分布的标准差值选择与 2000 次迭代中“候选值”被拒绝次数之间的关系：

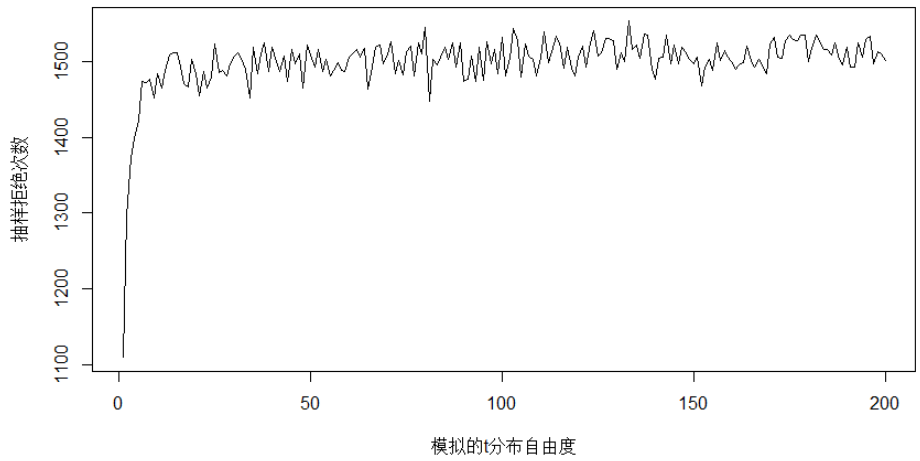


Figure 20: 随机游走 Metropolis-Hastings 实验三“候选值”被拒绝次数

通过图像，我们发现，随着每次模拟实验的后验分布的自由度不断增大，在每次实验中的 2000 次迭代中“候选值”被拒绝次数逐渐增加。但是在自由度达到 10 左右时，抽样拒绝次数就达到一个较大值，且后续围绕这个值进行波动。因此，“候选值”被拒绝的次数是随着 σ 的值不断增大而不断增大的。

最后我们给出随着模拟实验的后验分布的自由度不断增大实验得到样本链的平稳情况的图像。在这里，我们基于之前实验二的经验，我选择了每次实验的 acf 图像的 lag=20 的值进行记录，进而将其体现在图像上，如下所示：

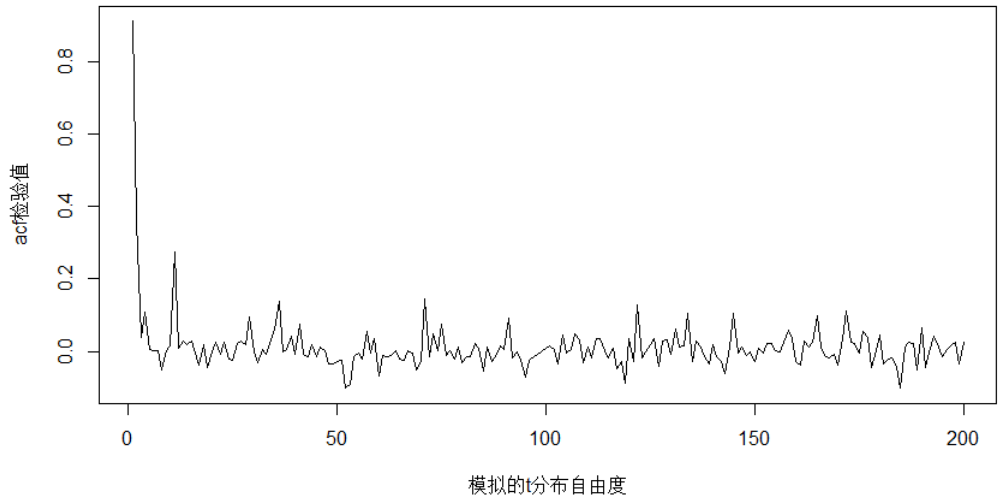


Figure 21: 随机游走 Metropolis-Hastings 实验三每次实验 acf 值

可以看到当自由度达到 5 左右时，acf 检验值就依据趋近于 0，且后续一直围绕其进行波动。这简介说明当自由度较大时，可以得到一条相对稳定的样本数据链。

综上，我们不难看出，此处作为后验分布的 t 分布的自由度选择为 5-10 左右是较好的选择。当自由度过大时，各项指标的变动实际不大，主要是围绕一个固定值进行上下波动。而且自由度在 5-10 之间时，迭代中“候选值”被拒绝次数并不会特别大，这可以说明方法的有效性，其对于参数空间的探索是足够的。

总之，随机游走 Metropolis-Hastings 算法作为一种较为基础且简便的 Metropolis-Hastings 算法，在现实中得到了广泛应用。

4.3 独立抽样 Metropolis-Hastings 算法

前文中我们已经系统地介绍并研究了随机游走 Metropolis-Hastings 算法。该算法的优势在于，在计算接受概率时，对于计算过程进行了一定程度上的简化。主要在于其建议分布的选择上，选择了一些具有良好性质和特征的分布，在一定范围内能够满足对称性质，因此此时的接受概率值主要是依赖于目标后验分布本身。

但是简单的计算也势必会带来一些问题。随机游走 Metropolis-Hastings 算法的接受概率完全依赖于目标后验分布，这会某种程度上增加抽样的偶然性，这使得抽样结果的精度与参数估计的准确性都成为了需要考虑的问题。

于是，统计学家们引入了一种独立抽样的 Metropolis-Hastings 算法。该种方法的建

议分布生成的样本不依赖于之前产生的抽样结果。此时建议分布的参数可以通过近似方法或者此前的经验得到。

在独立抽样 Metropolis-Hastings 算法中，我们常用的建议分布为多元正态分布

$$\theta^* \sim N_d(\bar{\theta}, \bar{s_\theta}^2) \tag{21}$$

事实上，当建议分布 q(θ) 能有效地近似目标后验分布 f(θ|y) 时的独立抽样是较为有效的。因此，一个比较好的获得建议分布的方法是采用 Laplace 近似方法获得建议分布的密度函数，且此时的建议分布主要是多元正态分布。

Laplace 近似方法主要是用于估计分布的方差。我们需要构造精度矩阵

$$H(\tilde{\theta}) = -\frac{\partial^2 log f(\theta|y)}{\partial\theta_i \partial\theta_j}|_{\tilde{\theta}} \tag{22}$$

其中得到 $\tilde{\theta}$ 维后验众数，也是建议分布的均值。则我们可以得到一个有效的建议分布为

$$q(\theta) = N_d(\tilde{\theta}, [-\frac{\partial^2 log f(\theta|y)}{\partial\theta_i \partial\theta_j}|_{\tilde{\theta}}]^{-1}) \tag{23}$$

故当我们进行每一步的迭代，将参数进行更新，可以得到接受概率为

$$\alpha = min(1, \frac{f(\theta = \theta^*|y)q(\theta = \theta^{(t-1)})}{f(\theta = \theta^{(t-1)}|y)q(\theta = \theta^*)}) \tag{24}$$

在实际分析中，通常选择建议分布的厚尾性要高于目标后验分布，且二者可以达到一定程度上的接近。此时的方法会更有效一些。

4.3.1 实验：利用独立抽样 Metropolis-Hastings 算法估计

独立抽样 Metropolis-Hastings 算法相较于随机游走 Metropolis-Hastings 算法的一大优势在于可以作用于更为复杂的分布，对于一些复杂分布也有一些较好的模拟。在这里，我们采用一个由两个正态分布 N(μ_1,σ_1) 与 N(μ_2,σ_2) 进行混合得到的混合正态分布，进行随机抽样。混合分布的形式为

$$pN(\mu_1, \sigma_1^2) + (1-p)N(\mu_2, \sigma_2^2) \tag{25}$$

在这里，我们可以使用独立抽样 Metropolis-Hastings 算法估计混合参数 p 的值。其主要做法在于选定建议分布后，依据给定分布生成观察值，生成独立样本链。接着依据建议分布生成候选值，再依据目标后验分布与建议分布生成接受概率；再依据该接受概率生成伯努利分布进行抽样，决定下一步迭代值为保持原值不变还是选择候选值。对此，进行多步迭代后，可以得到参数的估计值。

在此我采用独立抽样 Metropolis-Hastings 算法，对于缺乏参数 p 的已知信息，可以采用参数均为 1 的 beta 分布作为建议分布，即 0 到 1 之间的均匀分布。由建议分布 q 生成的样本值的接受概率为

$$\alpha(X_t, Y) = min(1, \frac{p(\theta = \theta^*|\theta^{(t-1)})q(\theta = \theta^{(t-1)})}{p(\theta = \theta^{t-1}|\theta^{(*)})q(\theta = \theta^*)}) \tag{26}$$

在此处，若选择 beta(a,b) 作为建议分布，接受概率的表达式为

$$\begin{aligned}\alpha = &\frac{p(\theta = \theta^*|\theta^{(t-1)})q(\theta = \theta^{(t-1)})}{p(\theta = \theta^{t-1}|\theta^{(*)})q(\theta = \theta^*)} = \\ &\frac{(\theta^{(t-1)})^{a-1}(1-(\theta^{(t-1)})^{(b-1)}\prod_{j=1}^{n}[\theta^* f_1(z_j) + (1-\theta^*)f_2(z_j)]}{(\theta^*)^{a-1}(1-\theta^*)^{b-1}\prod_{j=1}^{n}[\theta^{(t-1)} f_1(z_j) + (1-\theta^{(t-1)})f_2(z_j)]}\end{aligned} \tag{27}$$

其中的 $f_1(z_j)$ 与 $f_2(z_j)$ 分别为两个正态分布的密度函数。在此处，我们选择的目标后验分布为 $0.2N(0,1) + 0.8N(5,1)$，此处的估计混合参数 p 的值取为 0.2. 进行 20000 次迭代算法，得到的 p 值的每步迭代值如下所示：

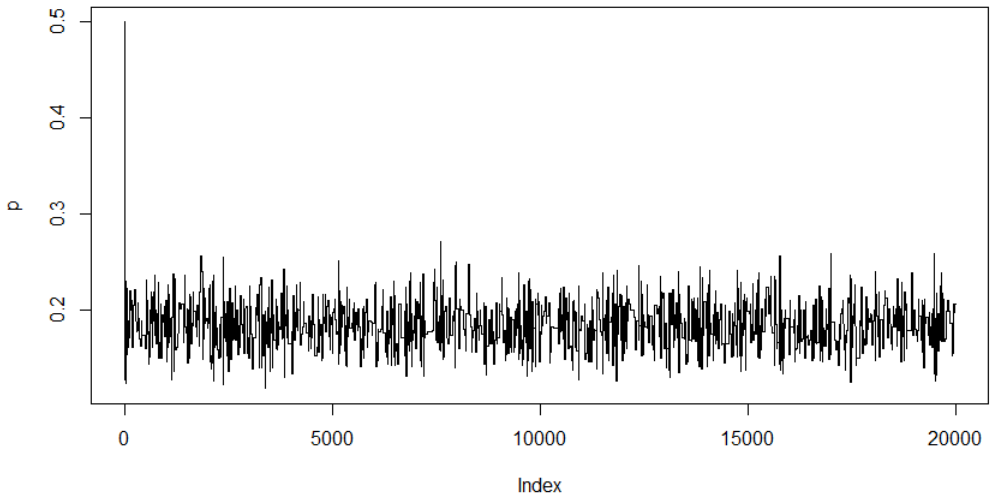


Figure 22: 独立抽样 Metropolis-Hastings 对混合分布 p 值估计

可以看到 p 值几乎都在 0.2 附近波动。我燃烧掉前 1000 次迭代结果，对于剩下的 19000 次迭代结果取均值，得到的数值估计结果如下所示：

```
> print(mean(xt[1001:m]))
[1] 0.1893693
```

Figure 23: 独立抽样 Metropolis-Hastings 对混合分布 p 值估计

可以看到估计效果还是不错的。

我尝试对于 p 值取 0.01-0.99 的序列值，依次进行实验，共进行 100 次实验，每次实验采用独立抽样 Metropolis-Hastings 算法进行 20000 次迭代，对于估计得到的 p 值与真实值进行比较，求解出每次实验的估计 MSE. 对此，我得到的实验结果如下所示：

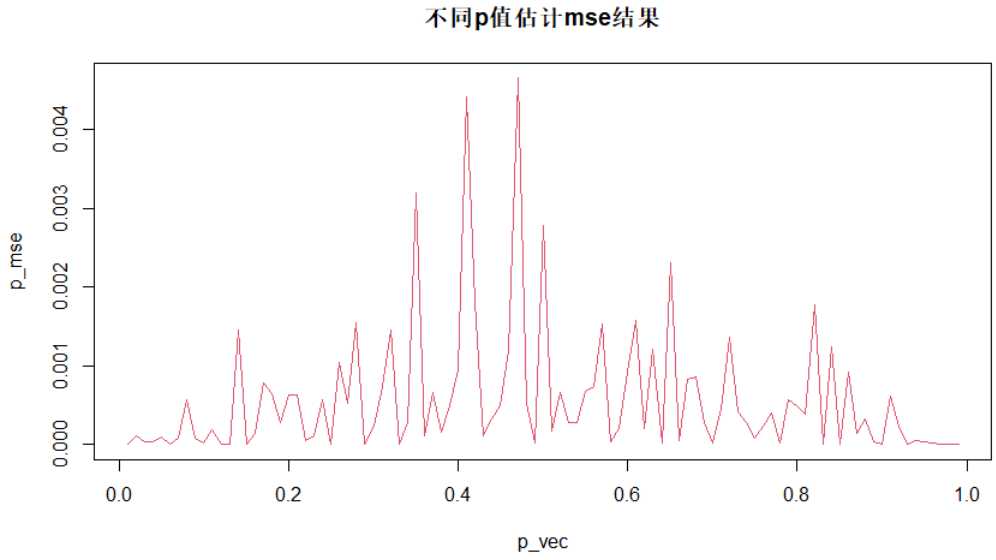


Figure 24: 独立抽样 Metropolis-Hastings 对混合分布 p 值估计 MSE(变化)

由该图可以看出，MSE 结在 p 位于 0.3-0.6 之间时较大，而对于位于边缘的部分，则误差 MSE 结果较小。事实上，这一点是可以被理解的。因为在位于区间的边缘处，p 的值或者很大或者很小，此时的混合分布愈发接近于一个单纯的正态分布的形式，故在一定程度上降低了参数估计的难度。显然对于单一的分布，估计的结果是更为精确的，故此时的误差 MSE 值也较小。

4.3.2 对比试验：后验分布为二项分布

在此处用一个实例对随机游走 Metropolis-Hastings 与独立抽样 Metropolis-Hastings 算法进行对比。在这里，我主要采用了一个二项分布的形式进行实验，用于预测球队在一系列比赛中的胜率情况。

若我们假定球队的获胜胜率为 π，则获胜次数服从二项分布，取为先验

$$f(y|\pi) = C_N^y \pi^y (1-\pi)^{N-y} \tag{28}$$

此处为了便于参数数值的选取，可以对 π 进行变换，令 $\theta = log[\frac{\pi}{1-\pi}]$, 则 $\pi = \frac{e^\theta}{1+e^\theta}$. 则获胜次数的二项分布为

$$\begin{aligned} f(y|\theta) &= C_N^y (\frac{exp(\theta)}{1+exp(\theta)})^y (\frac{1}{1+exp(\theta)})^{N-y} = C_N^y \frac{exp(\theta y)}{(1+exp(\theta))^N} \\ &= exp(logC_N^y + \theta y - Nlog(1+exp(\theta))) \end{aligned} \tag{29}$$

而对于参数 θ，我们还是考虑建议分布为一正态分布 $N($ $mu_\theta, \sigma_\theta^2)$. 故可得目标后验分布为以下形式：

$$f(\theta|y) \propto \frac{exp(\theta y)}{(1+exp(\theta))^N} exp[-\frac{1}{2}(\frac{\theta-\mu_\theta}{\sigma_\theta})^2] \tag{30}$$

综上，对于随机游走 Metropolis-Hastings 算法，迭代步骤如下所示：

(1) 设定迭代初始值为 $\theta^{(0)}$.

(2) 对于迭代步数 t=1,2,$\cdots$,T, 重复以下步骤：

设定 $\theta = \theta^{(t-1)}$

从建议分布正态分布 N(θ,$\bar{S_\theta}^2$) 生成新的备选参数值为 θ^*.

计算接受概率的对数值为 $log(\alpha) = min(0, A)$, 且

$$A = log\frac{f(y|\theta^*)f(\theta^*)}{f(y|\theta)f(\theta)} = (\theta^*-\theta)(y - \frac{\theta^*+\theta-2{\mu_\theta}^2}{2\sigma_\theta}) - Nlog\frac{1+exp(\theta^*)}{1+exp(\theta)} \tag{31}$$

以接受概率 α 选取“候选值” θ^*, 而以 1-α 的概率值接受上一步迭代值 $\theta^{(t-1)}$. 即完成一次迭代。

在这里，我对这个二项分布的情况用随机游走 Metropolis-Hastings 算法进行了一次尝试。假定一共进行 1000 场比赛中，球队胜利 600 场。由此构造出二项分布进行实验。得到的参数 π 与 θ 的迹图如下所示：

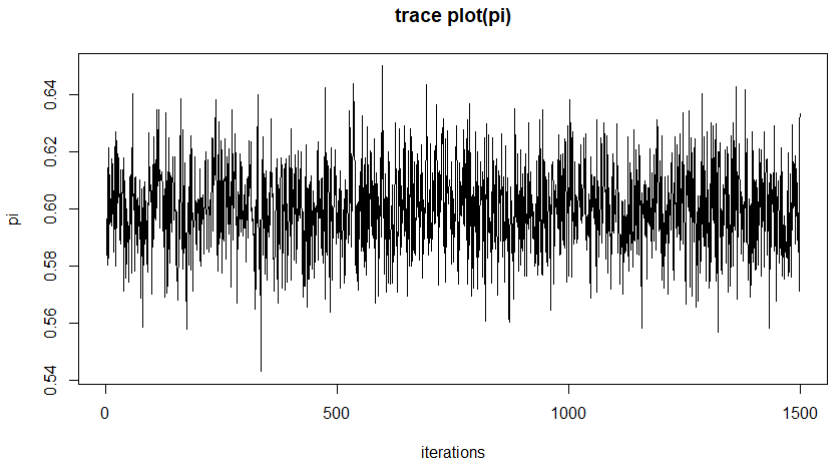


Figure 25: 随机游走 Metropolis-Hastings 作用于二项分布 π 的迹图

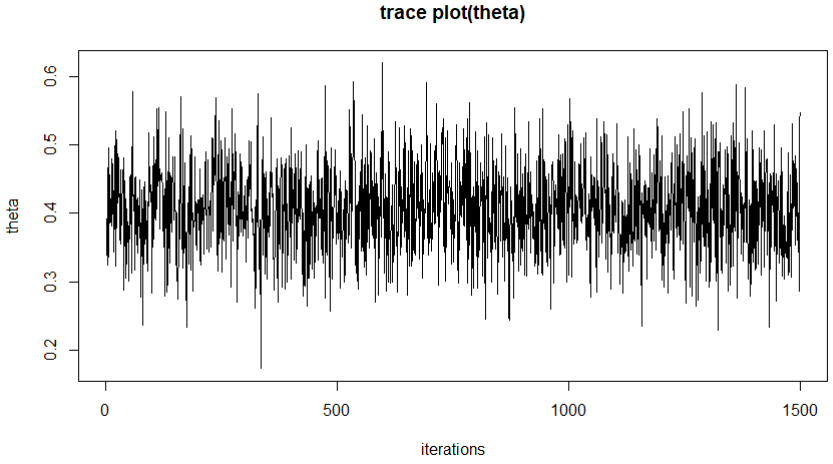


Figure 26: 随机游走 Metropolis-Hastings 作用于二项分布 θ 的迹图

可以看到胜率 π 的模拟值一直在 0.6 附近波动。

接着我们用独立抽样 Metropolis-Hastings 算法对该问题进行求解。在此处我们构建一个专门用于该问题的建议分布。参照前文中对于先验分布取对数求二阶导构造建议分布的方法，设总场次为 N，此时根据先验为二项分布，可得

$$\frac{\partial log f(\theta|y)}{\partial \theta} = y - N\frac{exp(\theta)}{1+exp(\theta)} - \frac{\theta-\mu_\theta}{\sigma_\theta^2} \tag{32}$$

而建议分布均值的估计值为

$$\bar{\mu_\theta} = \hat{\theta} = log[\frac{y}{N-y}] \tag{33}$$

建议分布方差的估计值为

$$\begin{aligned} \bar{s_\theta}^2 = \tilde{\sigma_\theta}^2 &\approx [-\frac{\partial^2 log f(\theta|y)}{\partial\theta_i \partial\theta_j}|_{\tilde{\theta}}]^{-1} = [\frac{y(N-y)}{N} + \frac{1}{\sigma_\theta^2}]^{-1} \\ &= (\frac{1}{y} + \frac{1}{N-y})[1 + \frac{(1/y + 1/(N-y))}{\sigma_\theta^2}]^{-1} \end{aligned} \tag{34}$$

由此得建议分布 $N(\bar{\mu_\theta}, \bar{S_\theta}^2)$.

则仿照之前的随机游走 Metropolis-Hastings 算法，可得算法的迭代过程如下所示：

(1) 设定迭代初值为 $\theta = \theta^{(t-1)}$.

(2) 从建议分布 $N(\bar{\mu_\theta}, \bar{S_\theta}^2)$ 生成参数的迭代候选值 θ^*。

(3) 得到接受概率的对数值为 $log(\alpha) = min(0, A)$. 其中

$$\begin{aligned} A &= log\frac{f(y|\theta^*)f(\theta^*)}{f(y|\theta)f(\theta)} + log\frac{f_N(\theta;\bar{\theta},\bar{s_\theta}^2)}{f_N(\theta^*;\bar{\theta},\bar{s_\theta}^2)} \\ &= (\theta^*-\theta)(y - \frac{\theta^*+\theta-2\mu_\theta}{2\sigma_\theta^2} + \frac{\theta^*+\theta-2\bar{\mu_\theta}}{2\bar{s_\theta}^2}) - Nlog\frac{1+exp(\theta^*)}{1+exp(\theta)} \end{aligned} \tag{35}$$

(4) 以接受概率 α 选取“候选值” θ^*, 而以 1-α 的概率值接受上一步迭代值 $\theta^{(t-1)}$. 即完成一次迭代。

此时，我们用独立抽样 Metropolis-Hastings 算法得到的参数 π 与 θ 的迹图如下所示：

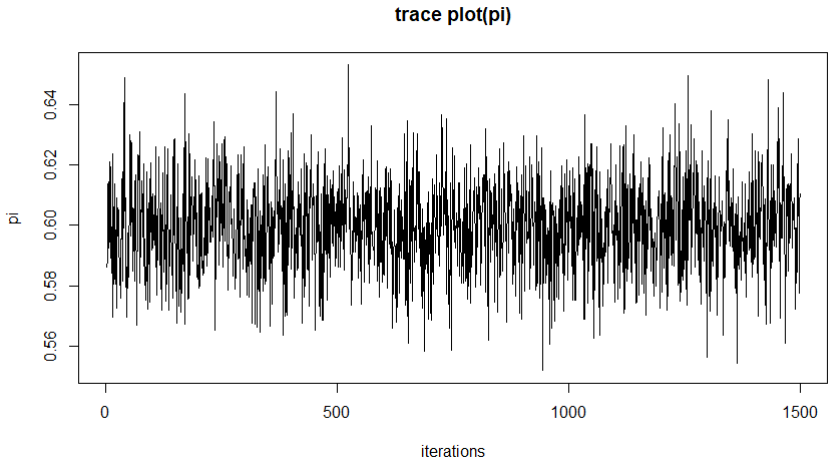


Figure 27: 独立抽样 Metropolis-Hastings 作用于二项分布 π 的迹图

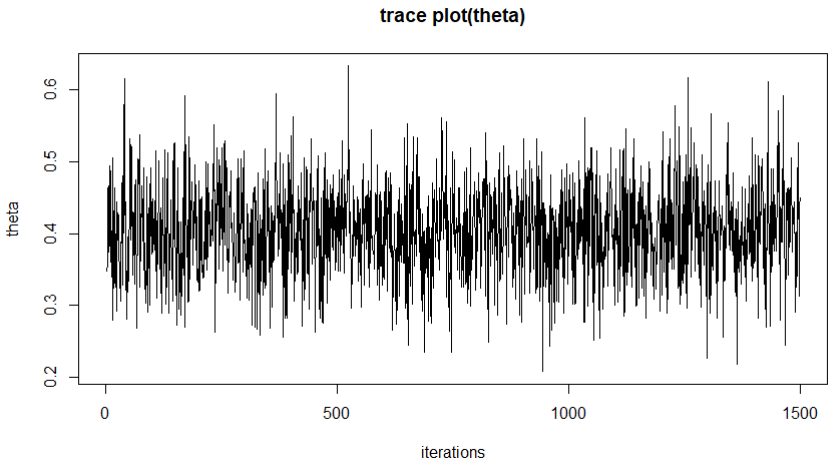


Figure 28: 独立抽样 Metropolis-Hastings 作用于二项分布 θ 的迹图

若将二者的数值有关结果放到一起比较，如下两张图所示：

```
> summary(theta)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1728  0.3617  0.4066  0.4062  0.4510  0.6194
> summary(p)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5431  0.5895  0.6003  0.6001  0.6109  0.6501
> sd(theta)
[1] 0.06546194
> sd(p)
[1] 0.01568905
> quantile(theta,c(0.025,0.975))
     2.5%     97.5%
0.2809761 0.5335475
> quantile(p,c(0.025,0.975))
     2.5%     97.5%
0.5697855 0.6303101
```

(a) 随机游走 Metropolis-Hastings 参数估计数值结果

```
> summary(theta)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2079  0.3615  0.4034  0.4048  0.4480  0.6329
> summary(p)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5518  0.5894  0.5995  0.5997  0.6102  0.6531
> sd(theta)
[1] 0.06465668
> sd(p)
[1] 0.01550226
> quantile(theta,c(0.025,0.975))
     2.5%     97.5%
0.2772424 0.5302906
> quantile(p,c(0.025,0.975))
     2.5%     97.5%
0.5688700 0.6295509
```

(b) 独立抽样 Metropolis-Hastings 参数估计数值结果

Figure 29: 两种 Metropolis-Hastings 参数估计数值结果比较

可以看到两参数的估计结果差不多，估计效果相当。但是独立抽样 Metropolis-Hastings 参数估计数值结果的标准差要少一些,且分布结果厚尾性没有随机游走 Metropolis-Hastings 参数估计数值结果明显。

但是总得来说，一次实验的结果不具备代表性。因此，我将球队的胜场次数构造出一个序列值进行实验，基于 Monte-Carlo 误差、参数计算的标准差以及算法迭代过程中“候选值”被接受的次数为评判标准，对两种算法对于这一二项分布模型的运用结果进行评判，比较这两种算法的优劣。

在此，我设定 1000 场中的胜场数从 50 开始，依次增加 10 场，直到 950 场，由此形成一个胜场次数的等差序列。可以看到，我舍弃了胜场次数过少与胜场次数过多的两种情况，这是为了防止边界值对实验结果产生巧合性质的影响。接着对于这其中的每一种胜场情况，我都用两种算法进行了实验，且每进行完一次实验，都计算出比较两种算法的评判标准参数，并将其可视化到图像上。

首先是 Monte-Carlo 误差值的结果。我做出了 θ 与 π 两个参数每次估计过程中的 Monte-Carlo 误差值结果，如下图所示：

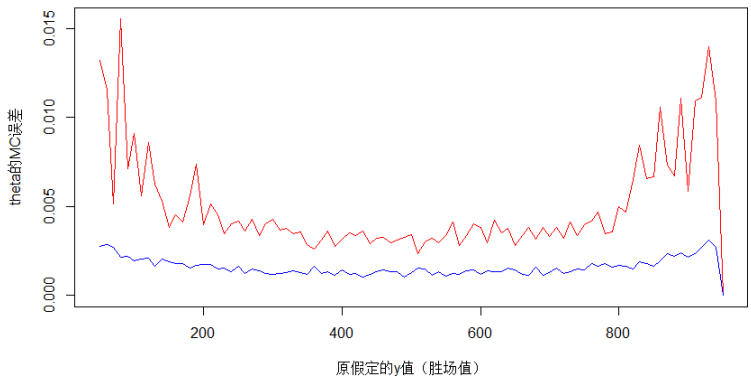


Figure 30: 两种 Metropolis-Hastings 算法实验 θMonte-Carlo 误差值比较

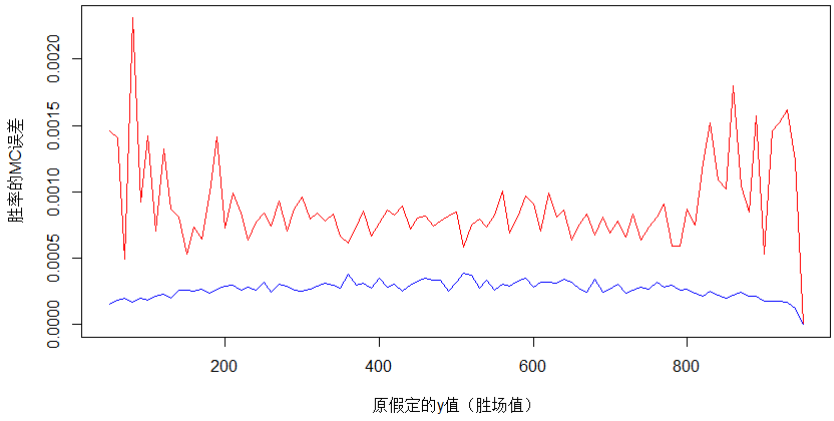


Figure 31: 两种 Metropolis-Hastings 算法实验胜率 πMonte-Carlo 误差值比较

由这两幅图[4]，我们可以很明显地看到，红线几乎对于序列中的所有值均位于蓝线的上方，这说明随机游走 Metropolis-Hastings 算法结果的 Monte-Carlo 误差要明显大于独立抽样 Metropolis-Hastings 算法结果的误差。从这一角度可以说明对于该二项分布模型，独立抽样 Metropolis-Hastings 算法是更好的算法。

接着是参数标准差的结果。我做出了 θ 与 π 两个参数每次估计过程中的标准差值结果，如下图所示：

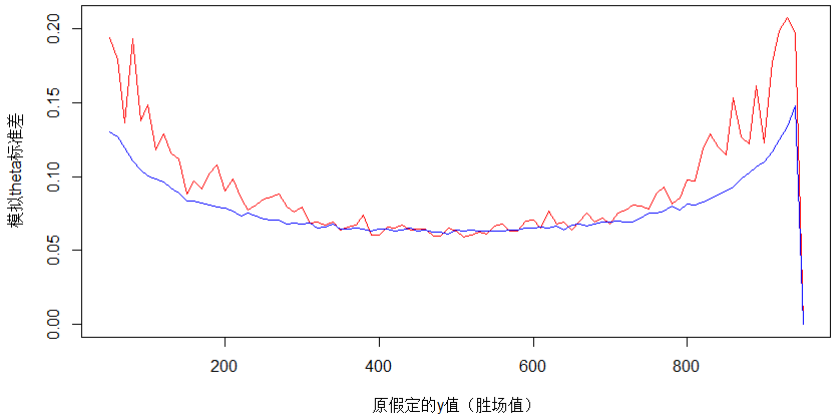


Figure 32: 两种 Metropolis-Hastings 算法实验 θ 标准差比较

[4]该实验中所有图中红线为随机游走 Metropolis-Hastings 算法结果，蓝线为独立抽样 Metropolis-Hastings 算法结果

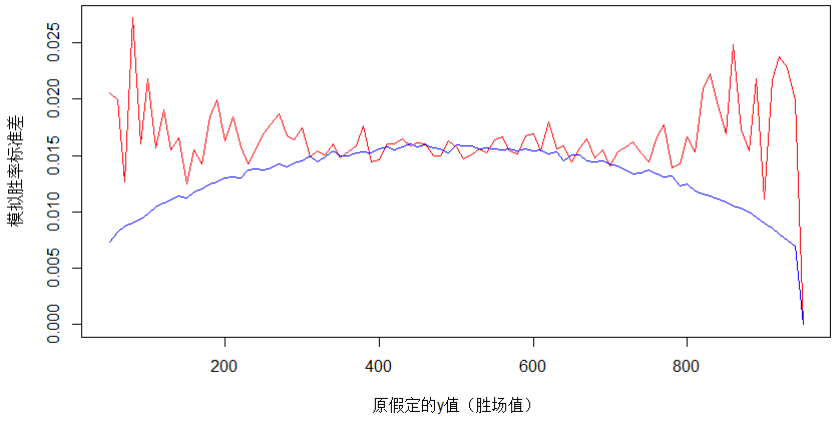


Figure 33: 两种 Metropolis-Hastings 算法实验胜率 π 标准差比较

由标准差的结果，我们发现对于序列中较为边缘的情况，红线明显位于蓝线的上方，这说明对于胜场次数较少或者胜场次数较多的情况，随机游走 Metropolis-Hastings 算法的标准差结果明显大于独立抽样 Metropolis-Hastings 算法的标准差结果。而对于胜场次数位于中段的情况，二者的标准差结果十分接近。

标准差这一参数可以侧面反映估计的稳定性程度。这说明对于胜场次数较少或者胜场次数较多的情况，随机游走 Metropolis-Hastings 算法的估计稳定性明显差于独立抽样 Metropolis-Hastings 算法的估计稳定性。而对于胜场序列中段的胜场次数，两种算法的稳定性相当。从这一角度出发，也是独立抽样 Metropolis-Hastings 算法的估计稳定性更高一些。

最后是迭代过程中候选值被接受次数的结果。两种算法的结果分别如下所示：

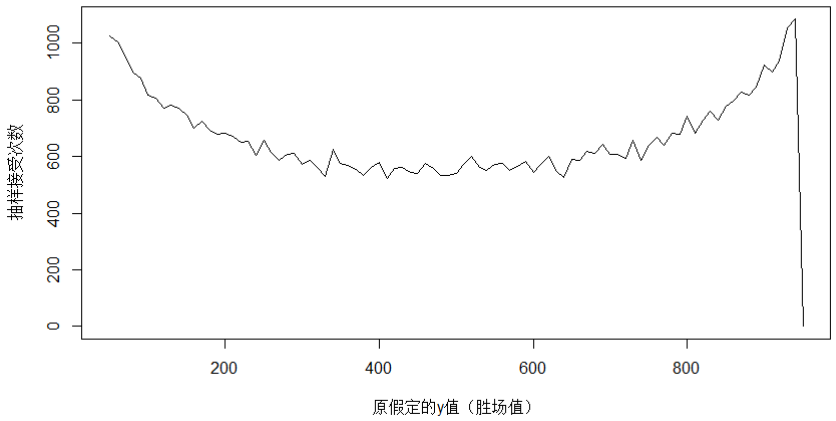


Figure 34: 随机游走 Metropolis-Hastings 算法实验迭代被拒绝次数

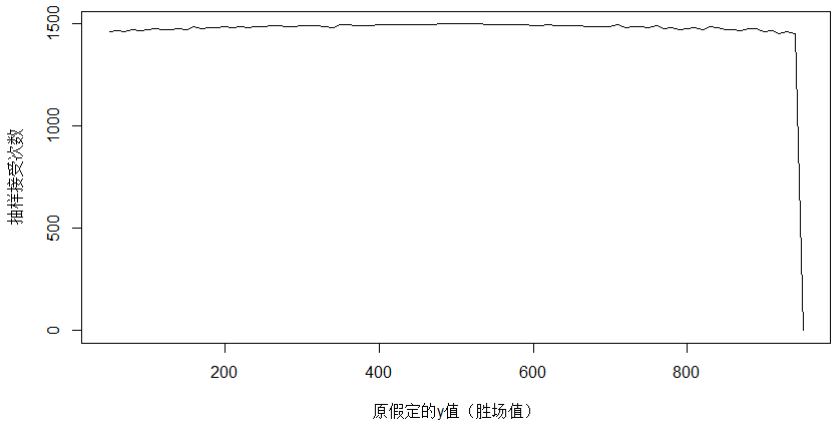


Figure 35: 独立抽样 Metropolis-Hastings 算法实验迭代被拒绝次数

可以看到此处独立抽样 Metropolis-Hastings 算法实验迭代被拒绝次数要远远大于随机游走 Metropolis-Hastings 算法实验迭代被拒绝次数 (也正是如此，我将结果分为两张图呈现，否则必定有一种方法的图像呈现不出来)。这说明独立抽样 Metropolis-Hastings 算法是一种更为有效的迭代算法。其对于参数空间的探索程度更高。

综合以上所有的图像与实验结果，我们可以知道对于这一二项分布模型，独立抽样 Metropolis-Hastings 算法的估计要明显优于随机游走 Metropolis-Hastings 算法。事实上，独立抽样 Metropolis-Hastings 算法的精确度往往更高，但是在计算复杂度上也要远高于随机游走 Metropolis-Hastings 算法。因此对于实际问题的解决，往往需要我们综合考虑多方面的问题，以选择最佳的算法进行求解。

5 MCMC 方法时间序列

在前文中，我已经介绍了 Gibbs 抽样、随机游走 Metropolis-Hastings 算法以及独立抽样 Metropolis-Hastings 算法。这些算法都是 MCMC 中的经典算法，主要采用构造建议分布采样出迭代候选值，进而生成后验分布，得到接受概率进而决定下一步迭代值，逐渐将结果向真实值逼近的算法。

这种时刻将先验分布考虑进去的算法也可以被广泛应用到时间序列分析中。在时间序列中，一个最常为研究的对象就是金融时间序列数据，例如股票的每日对数收益率数据等等。通常时间序列数据的分析最大的难度在于其时序性。数据随时间波动的性质会使得我们较难提取数据中所蕴含的特征。

MCMC 时刻考虑分布的性质在时间序列的分析与预测中可以取到较好的效果。在 R 语言中，我主要调用 bsts 这一专门用于 Bayes 时间序列分析的包进行实验。这个包中最主要的工作函数即为 bsts 函数，其功能极其强大，可以在综合考虑数据集，指数族方法以及先验的情况下，对于时间序列数据进行残差分离，模型建立，模型拟合以及数

据预测等等。下面我们就用这个包中的函数对于一组时间序列数据进行分析。

5.1 MCMC 时间序列分析 S&P500 数据

在此处，我们用时间序列股票数据中较为经典的标普 S&P500 数据集进行分析。我选择的数据跨度为 2016 年 1 月 4 日到 2020 年 12 月 28 日的 5 年内的所有交易日统计得到的 261 个周数据进行时间序列的建模。对于得到的每周的收盘价利用公式 $r_t = ln(\frac{P_t}{P_{t-1}})$ 进行取对数收益率的处理，得到具体对数收益率。选择周数据的收盘价与得到的对数收益率数据可视化后如下所示：

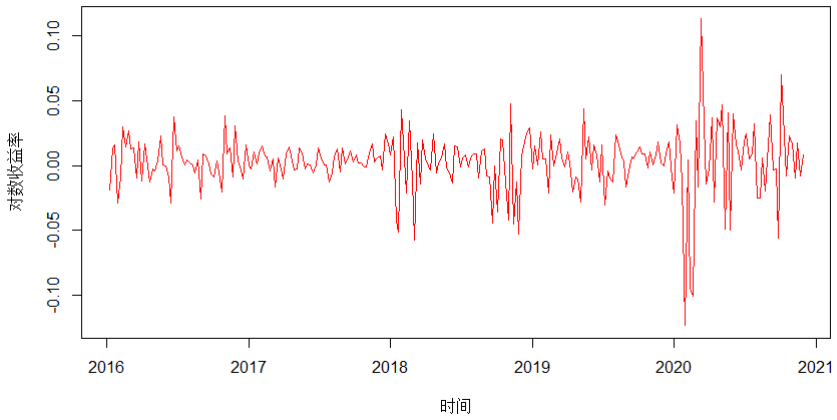


Figure 36: 2016-2020SP500 周数据对数收益率

在此处，用 bsts 函数作用完后，我可视化出了该对数收益率的“状态”(R 语言中意为 state)，也就是大致的分布点情况以及其大致的分布情况，如下图所示：

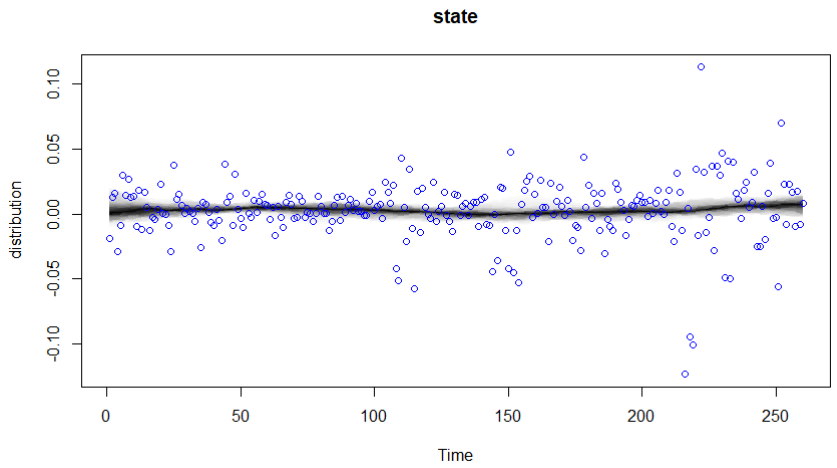


Figure 37: 2016-2020SP500 周数据对数收益率 State

可以看到，此时得到的这条分布曲线 (曲线较粗是因为加上了残差) 显然对于数据的拟合效果很差。此时可以分离出数据中的残差。我们得到数据点以及其残差的上下波动情况如下所示：

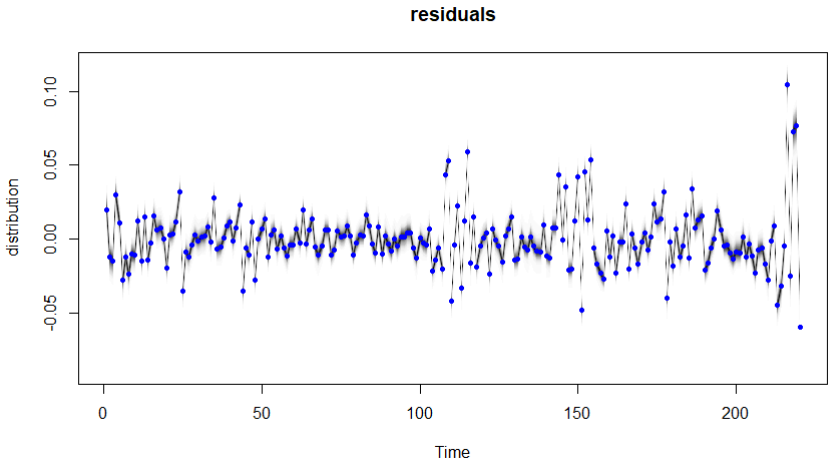


Figure 38: 2016-2020SP500 周数据对数收益率残差

该图显示出了每个数据点及其上下的残差情况。该图的意义在于，我们对于数据点的分布拟合应该合理位于残差所显示的范围内。

接着用 bsts 包自带的预测函数进行预测，由此可以得到预测分布的图像如下所示：

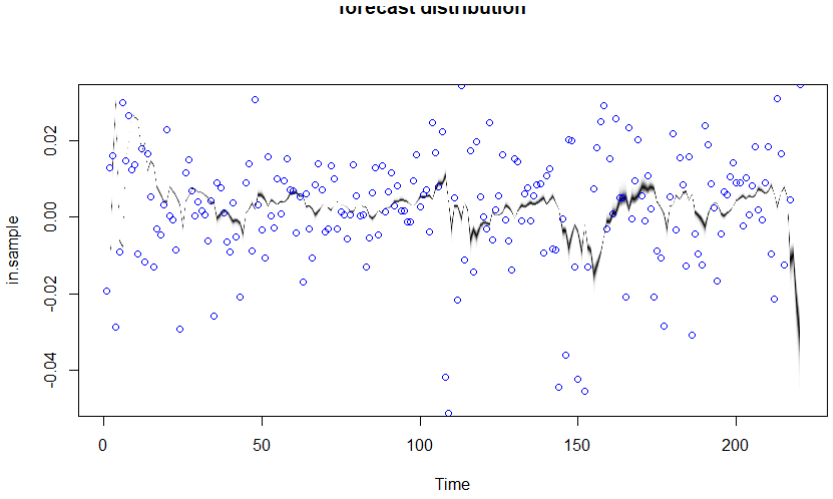


Figure 39: 2016-2020SP500 周数据对数收益率预测分布

注意到该图的数据点进行了一定程度上的修改。事实上，这是结合之前的残差图进行分析的结果。对于图像的原数据集，进行了某种程度上的修正，将一些极端值 (例如 2020 年 3-5 月因为新冠疫情影响而产生的一些极端值) 依照残差的波动进行了数据点的修正。此时拟合得到的分布 (上图中的黑线) 效果要远好于之前 State 中的分布拟合情况。而且此时的预测分布的过拟合情况也体现得不明显。

最后可以利用 bsts 包给出每个数据点处的预测误差分布图，如下所示：

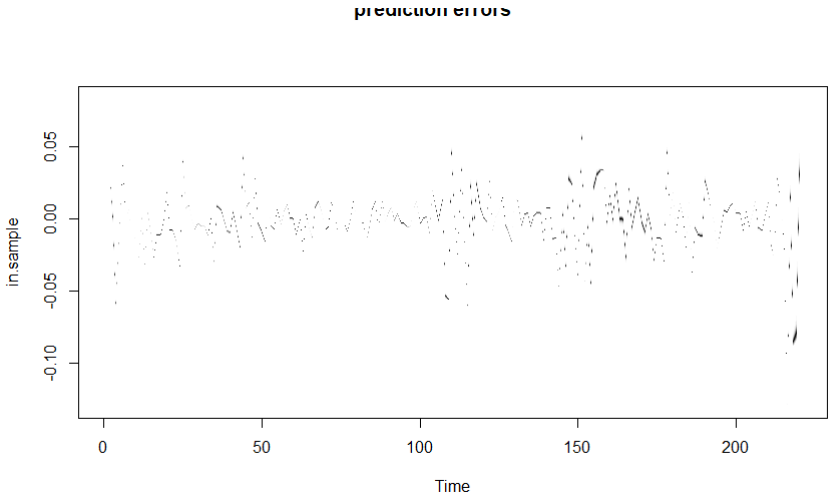


Figure 40: 2016-2020SP500 周数据对数收益率预测误差情况

我们可以明显地看出图中的虚线的波动情况在 2020 年达到了最大值，但是最大值也始终维持在 +/-0.1 之间。这说明预测的结果还是较好的，尤其是对于 2020 年出现疫情的极端值情况。

5.2 MCMC 时间序列方法的预测效果

前文中提到了 MCMC 时间序列分析的预测分布。但是此时的预测分布是基于已有的数据集与分布，对于已有值的拟合预测结果。这使我不禁对该方法对于未来数据集的预测结果产生了一些疑问。

在这里，我主要还是用 bsts 包中的预测函数，对于时间序列的未来值进行预测，进而计算预测误差，对于其预测方法进行分析。我选择了 S&P500 数据集 (2016 年 1 月到 2020 年 12 月数据) 与 GOOGLE 公司的股票数据集 (2007 年 1 月到 2012 年 12 月数据) 进行预测未来值效果的实验。主要将 MCMC 的预测未来值的效果与 R 语言中较为常用的时间序列方法的预测未来值方法的预测效果进行比对。

此处我主要进行了这样的操作。对于有接近 300 个数据的数据集，我从第 60 个数据集开始，第一次取前 60 个数据作为基础的分布，以此对第 61-72 个数据进行预测，并与真实值比较求解预测的 MSE. 接着取前 61 个数据作为基础的数据分布，对 62-73 个数据进行预测，计算 MSE; 以此类推，直到可以预测出数据集中的最后 12 个数据为止。由此可以将 MSE 可视化在图像上，比较两种方法的预测效果。

首先，用 S&P500 数据集进行实验。得到的 MCMC 的预测未来值的 MSE 与 R 语言中较为常用的时间序列方法的 MSE 如下图所示:(红线为 MCMC 的预测未来值的 MSE，蓝线为 R 语言中较为常用的时间序列方法的 MSE)

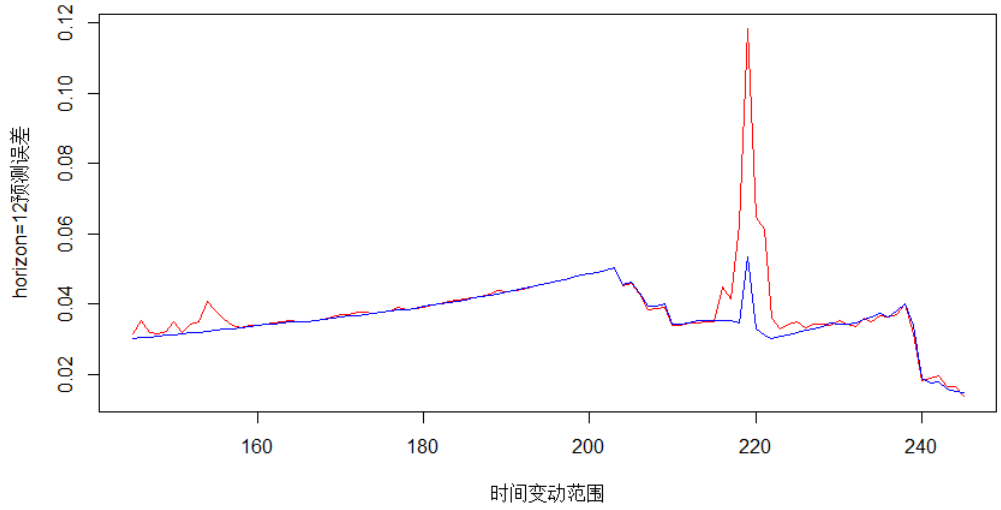


Figure 41: 两种时间序列分析方法预测未来值 MSE(S&P500)

可以看到，在 210-230 之外的时间变动范围下，二者的预测 MSE 几乎相同，比对不出孰优孰劣。但是对于 210-230 之间的时间变动范围，发现 MCMC 预测未来值的 MSE 要明显大于常用的时间序列方法预测未来值的 MSE. 在我看来这是因为此段时间由于疫情而出现极端值，因此采用 MCMC 方法预测未来值需要依赖于已有数据的分布；但是鉴于突然出现的极端值，其对总体分布的改变较为有限，会被总体分布作为分布的极端值处理，因此其对于极端值的反应较差，此时的预测 MSE 就非常大。

采用 GOOGLE 公司的股票数据集进行实验，得到的结果如下图所示：

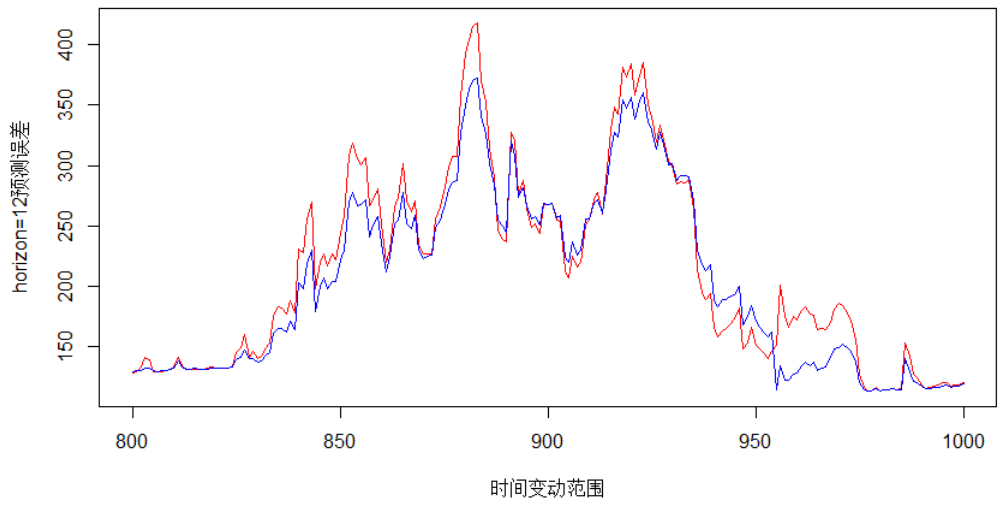


Figure 42: 两种时间序列分析方法预测未来值 MSE(GOOGLE)

可以看到此时的极端值出现情况较少，因此二者的预测 MSE 差不多，且在一些中间值位置，MCMC 时间序列的预测效果要更好一些。

该实验可以变相说明 MCMC 方法预测未来值对于极端值出现的情况下，预测的效果较差。因此对于数据集而言，去除异常值、极端值以及选择方法的效果就非常重要了。

5.3 MCMC 时间序列方法对于季节性数据预测的优势

季节性是时间序列数据中一个较为重要的性质。往往具有季节性的数据在分析方面会具有一定的便利。值得一提的是，MCMC 时间序列方法对于具有季节性质的数据集预测效果十分好。以如下实验结果为例：

此处我采用的数据集为 R 语言中自带的 AirPassengers 数据集。该数据集反映了 1949-1960 年间某航空公司的某架客机的接待乘客数。这是 R 语言中自带的一个极为经典的季节性质较为明显的数据集。

该数据集的数据可视化结果如下所示：

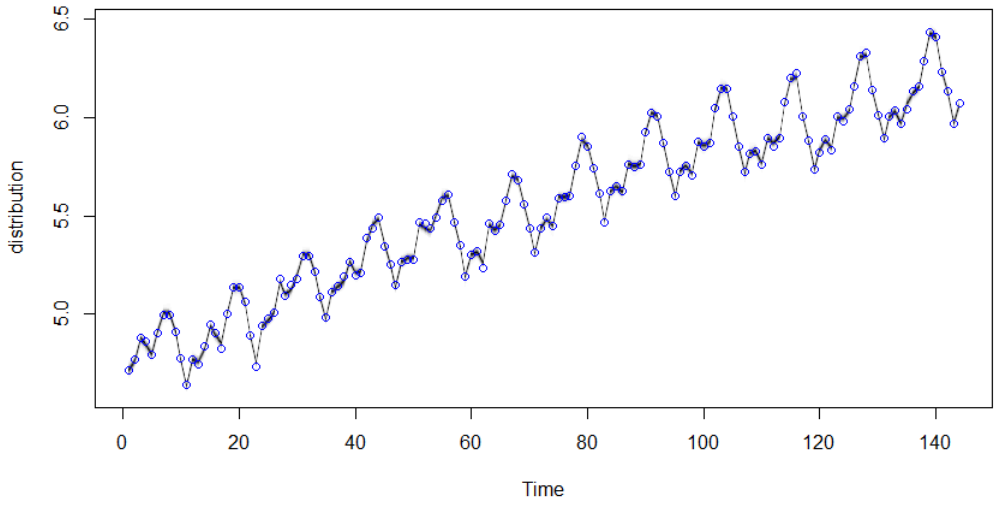


Figure 43: 时间序列 AirPassenger 数据集

在这里我们可以很明显地发现数据集的季节性质，且该性质主要以年为单位体现出来。

我对于原数据集去除掉 1960 年的数据，仅仅利用 1949-1959 年的数据，对 1960 年的数据进行未来值预测。

此时得到的预测未来值的结果如下图所示：

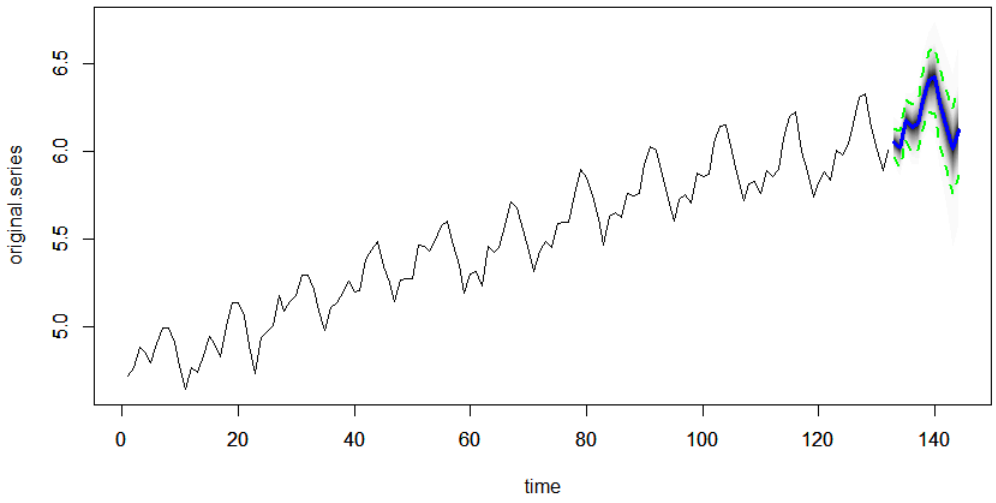


Figure 44: 时间序列 AirPassenger 数据集预测未来值结果

而可以得到真实值与预测值的数值结果如下所示：

```
> pred$mean
 [1] 6.051714 6.017989 6.175570 6.132370 6.153883 6.284521 6.397898 6.411522 6.250113 6.129426
[11] 6.009574 6.121436
> y_ori[133:144]
 [1] 6.033086 5.968708 6.037871 6.133398 6.156979 6.282267 6.432940 6.406880 6.230481 6.133398
[11] 5.966147 6.068426
```

Figure 45: 时间序列 AirPassenger 数据集预测未来值数值结果

无论是图像结果函数数值结果，都可以说明 MCMC 时间序列方法对于季节性数据的未来值预测效果十分优秀，MSE 值很小。

6 总结

本报告主要围绕 MCMC 算法展开，主要介绍了 MCMC 算法的基本原理，简述了几种极为经典的 MCMC 算法，例如 Gibbs 抽样方法、随机游走 Metropolis-Hastings 算法、独立抽样 Metropolis-Hastings 算法等，并对于这些算法的参数估计、抽样性能进行了探究与比较。与此同时，尝试了利用 MCMC 方法对时间序列数据分析，发现了 MCMC 方法对于时间序列数据存在的优缺点。总之，MCMC 方法因为其方法的特殊性以及技巧性，已经为越来越多的领域得到应用。

A 相关代码

Gibbs抽样一维正态分布：

```
y <- rnorm(100,mean = 100,sd=0.1)
bary <- mean(y)
iterations <- 50000
mu_0 <- 0
s_0 <- 100
a_0 <- 0.001
b_0 <- 0.001
theta <- matrix(nrow = iterations ,ncol = 2)
cur.mu <- 0
cur.tau <- 4
cur.s <- sqrt(1/cur.tau)
for(t in 1:iterations){
        w <- s_0^2/(cur.s^2/n+s_0^2)
        m <- w*bary+(1-w)*mu_0
        s <- sqrt(w/n)*cur.s
        cur.mu <- rnorm(1,m,s)
        a <- a_0+0.5*n
        b <- b_0+0.5*sum((y-cur.mu)^2)
        cur.tau <- rgamma(1,a,b)
        cur.s <- sqrt(1/cur.tau)
        theta[t,] <- c(cur.mu,cur.s)
}
plot(c(1:iterations),theta[,1],xlab = "迭代次数",ylab = "均值迭
    代结果",type="l")
plot(c(1:iterations),theta[,2],xlab = "迭代次数",ylab = "标准差
    迭代结果",type="l")
mean(theta[,1])
mean(theta[,2])
```

Gibbs抽样一维影响因素：

```
sdq <- seq(from=0.1,to = 10,by= 0.1)
meanres <- rep(0,length(sdq))
sdres <- rep(0,length(sdq))
meanmse <- rep(0,length(sdq))
sdmse <- rep(0,length(sdq))
M<- 100

for(i in 1:length(sdq)){
        y <- rnorm(100,mean = 100,sd=sdq[i])
        bary <- mean(y)
        iterations <- 10000
        mu_0 <- 0
        s_0 <- 10
        a_0 <- 0.001
        b_0 <- 0.001
        theta <- matrix(nrow = iterations,ncol = 2)
        cur.mu <- 0
        cur.tau <- 2
        cur.s <- sqrt(1/cur.tau)
        for(t in 1:iterations){
                w <- s_0^2/(cur.s^2/n+s_0^2)
                m <- w*bary+(1-w)*mu_0
                s <- sqrt(w/n)*cur.s
                cur.mu <- rnorm(1,m,s)
                a <- a_0+0.5*n
                b <- b_0+0.5*sum((y-cur.mu)^2)
                cur.tau <- rgamma(1,a,b)
                cur.s <- sqrt(1/cur.tau)
                theta[t,] <- c(cur.mu,cur.s)
        }
        meanres[i] <- mean(theta[,1])
        sdres[i] <- mean(theta[,2])
}

for(j in 1:length(meanres)){
```

```
        meanmse[j] <- (meanres[j]-M)^2
        sdmse[j] <- (sdres[j]-sdq[j])^2
}

plot(sdq,sdmse,col=2,type = "l")
plot(sdq,meanmse,col=2,type = "l")
```

Gibbs抽样二维正态分布：

```
N <- 5000
burn <- 1000
X <- matrix(0,N,2)
rho <- -.75
mu1 <- 0
mu2 <- 2
sigma1 <- 1
sigma2 <- .5
s1 <- sqrt(1-rho^2)*sigma1
s2 <- sqrt(1-rho^2)*sigma2
X[1,] <- c(mu1,mu2)
for(i in 2:N){
        x2 <- X[i-1,2]
        m1 <- mu1+rho*(x2-mu2)*sigma1/sigma2
        X[i,1] <- rnorm(1,m1,s1)
        x1 <- X[i,1]
        m2 <- mu2+rho*(x1-mu1)*sigma2/sigma1
        X[i,2] <- rnorm(1,m2,s2)
}
b <- burn+1
x <- X[b:N,]
colMeans(x)
cov(x)
cor(x)
```

```
plot(x,main="",cex=.5,xlab = bquote(X[1]),ylab = bquote(X[2]),
    ylim = range(x[,2]))



n <- 5000
burn.in <- 2500
X <- matrix(0,n,2)
mu1 <- 1
mu2 <- -1
sigma1 <- 1
sigma2 <- 2
rho <- .5
s1.c <- sqrt(1-rho^2)*sigma1
s2.c <- sqrt(1-rho^2)*sigma2
X[1,] <- c(mu1,mu2)
for(i in 2:n){
          x2 <- X[i-1,2]
          m1.c <- mu1+rho*(x2-mu2)*sigma1/sigma2
          X[i,1] <- rnorm(1,m1.c,s1.c)
          x1 <- X[i,1]
          m2.c <- mu2+rho*(x1-mu1)*sigma2/sigma1
          X[i,2] <- rnorm(1,m2.c,s2.c)
}
b <- burn.in+1
x.mcmc <- X[b:n,]
library(MASS)
MVN.kdensity <- kde2d(x.mcmc[,1],x.mcmc[,2],h=5)
plot(x.mcmc,col="blue",xlab="X1",ylab="X2")
contour(MVN.kdensity,add=TRUE)
hist(x.mcmc[,1])
library(stats)
acf(x.mcmc[,1])
index <- 1:n
plot(index,X[,1],type="l")
```

```
n <- 5000
burn.in <- 2500
X <- matrix(0,n,2)
mu1 <- 1
mu2 <- -1
sigma1 <- 1
sigma2 <- 2
rho <- .5
s1.c <- sqrt(1-rho^2)*sigma1
s2.c <- sqrt(1-rho^2)*sigma2
X[1,] <- c(mu1,mu2)
for(i in 2:n){
        x2 <- X[i-1,2]
        m1.c <- mu1+rho*(x2-mu2)*sigma1/sigma2
        X[i,1] <- rnorm(1,m1.c,s1.c)
        x1 <- X[i,1]
        m2.c <- mu2+rho*(x1-mu1)*sigma2/sigma1
        X[i,2] <- rnorm(1,m2.c,s2.c)
}
b <- burn.in+1
x.mcmc <- X[b:n,]
library(MASS)
MVN.kdensity <- kde2d(x.mcmc[,1],x.mcmc[,2],h=5)
plot(x.mcmc,col="blue",xlab="X1",ylab="X2")
contour(MVN.kdensity,add=TRUE)
hist(x.mcmc[,1])
library(stats)
acf(x.mcmc[,1])
index <- 1:n
plot(index,X[,1],type="l")
```

Monte–Carlo 误差计算：

```
mcerror.batch <- function(x,batches){##batch mean批均值法
        iter <- length(x)
        batleng <- ceiling(iter/batches)
        bats <- sort(rep(seq(from=1,to=batches,by=1),batleng))
        bats <- bats[1:iter]
        batmeans <- tapply(x, bats, mean)
        mc.error <- sd(batmeans)/sqrt(batches-1)
        return(mc.error)
}
```

随机游走Metropolis-Hastings算法函数：

```
rw.Metropolis <- function(n,sigma,x0,N){
        x <- numeric(N)
        x[1] <- x0
        u <- runif(N)
        k <- 0
        for(i in 2:N){
                y <- rnorm(1,x[i-1],sigma)
                if(u[i]<=(dt(y,n)/dt(x[i-1],n)))
                x[i] <- y
                else{
                        x[i] <- x[i-1]
                        k <- k+1
                }
        }
        return(list(x=x,k=k))
}
```

随机游走Metropolis-Hastings算法影响因素（4次实验）：

```
##
library(tseries)
path="C:\\Users\\user\\Documents\\统计实践与高维\\统计咨询与实
    践\\大报告\\code"
setwd(path)
source('随机游走Metropolis.R')
source('批均值法计算Monte Carlo误差.R')
n <- 5#设定t分布自由度
N <- 10000#抽样次数为2000次
sigma <- c(0.05,0.5,4,20)
x0 <- 25#设定数据链的初值
rw1 <- rw.Metropolis(n,sigma[1],x0,N)
rw2 <- rw.Metropolis(n,sigma[2],x0,N)
rw3 <- rw.Metropolis(n,sigma[3],x0,N)
rw4 <- rw.Metropolis(n,sigma[4],x0,N)
print(c(rw1$k,rw2$k,rw3$k,rw4$k))
index <- 1:10000
y1 <- rw1$x[index]
y2 <- rw2$x[index]
y3 <- rw3$x[index]
y4 <- rw4$x[index]
plot(index,y1,type="l",xlab = bquote(sigma*"=0.05"),main = "",
    ylab = "x")
plot(index,y2,type="l",xlab = bquote(sigma*"=0.5"),main = "",
    ylab = "x")
plot(index,y3,type="l",xlab = bquote(sigma*"=4"),main = "",ylab
    = "x")
plot(index,y4,type="l",xlab = bquote(sigma*"=20"),main = "",
    ylab = "x")
mcerror.batch(y1,batches=50)
mcerror.batch(y2,batches=50)
mcerror.batch(y3,batches=50)
mcerror.batch(y4,batches=50)
acf(y1)
acf(y2)
acf(y3)
acf(y4)
```

```
##发现sigma过小，图形为随机游走，数据链10000次迭代不收敛
##但是sigma大了，虽然收敛，但拒绝次数过多
```

随机游走Metropolis–Hastings算法建议分布sigma影响：

```
path="C:\\Users\\user\\Documents\\统计实践与高维\\统计咨询与实践\\大报告\\code"
setwd(path)
library(tseries)
source('随机游走Metropolis.R')
source('批均值法计算Monte Carlo误差.R')
sigma_max <- 100##设定建议分布sigma值范围为1-100
sigma_trans <- seq(from=0.05,to=sigma_max,by=0.05)##考虑建议分布sigma值的影响
rwreject <- rep(0,length(sigma_trans))
mcerrorlist <- rep(0,length(sigma_trans))
adflist <- rep(0,length(sigma_trans))
n <- 4
N <- 2000#抽样次数为2000次
x0 <- 25#设定数据链的初值
index <- 1:2000
for (i in 1:length(sigma_trans)) {
        rw <- rw.Metropolis(n,sigma_trans[i],x0,N)
        rwreject[i] <- rw$k
        y <- rw$x[index]
        mcerrorlist[i] <- mcerror.batch(y,batches = 50)
        adflist[i] <- adf.test(y)$p.value
}
plot(sigma_trans,rwreject,type="l",xlab = "模拟的建议分布sigma值",ylab = "抽样拒绝次数")
plot(sigma_trans,mcerrorlist,type="l",xlab = "模拟的建议分布sigma值",ylab = "Monto Carlo误差")
```

```
plot(sigma_trans,adflist,type="l",xlab = "模拟的建议分布sigma值
    ",ylab = "adf检验p值")
```

随机游走Metropolis–Hastings算法目标分布自由度影响：

```
path="C:\\Users\\user\\Documents\\统计实践与高维\\统计咨询与实
    践\\大报告\\code"
setwd(path)
library(tseries)
source('随机游走Metropolis.R')
source('批均值法计算Monte␣Carlo误差.R')
n_max <- 200##设定t分布的自由度范围为1-200
n_trans <- seq(from=1,to=n_max,by=1)##考虑t分布自由度的影响
rwreject <- rep(0,length(n_trans))
mcerrorlist <- rep(0,length(n_trans))
acflist <- rep(0,length(n_trans))
adflist <- rep(0,length(n_trans))
N <- 2000#抽样次数为2000次
sigma <- 5
x0 <- 25#设定数据链的初值
index <- 1:2000
for (i in 1:length(n_trans)) {
        rw <- rw.Metropolis(n_trans[i],sigma,x0,N)
        rwreject[i] <- rw$k
        y <- rw$x[index]
        mcerrorlist[i] <- mcerror.batch(y,batches = 50)
        acflist[i] <- acf(y)[20]
        adflist[i] <- adf.test(y)$p.value
}
plot(n_trans,rwreject,type="l",xlab = "模拟的t分布自由度",ylab
    = "抽样拒绝次数")
plot(n_trans,mcerrorlist,type="l",xlab = "模拟的t分布自由度",
    ylab = "Monto␣Carlo误差")
```

```
plot(n_trans,acflist,type="l",xlab = "模拟的t分布自由度",ylab =
    "acf检验值")
plot(n_trans,adflist,type="l",xlab = "模拟的t分布自由度",ylab =
    "adf检验p值")
```

独立抽样Metropolis−Hastings算法：

```
m <- 20000##样本链长度
xt <- numeric(m)
a <- 1
b <- 1##建议分布Beta(a,b)的参数
p <- 0.2##混合权重
n <- 300##样本量
mu <- c(0,5)
sigma <- c(1,1)##正态分布参数，用来形成混合分布
##生成观察值
i <- sample(1:2,size = n,replace = TRUE,prob = c(p,1-p))
x <- rnorm(n,mu[i],sigma[i])
##生成独立样本链
u <- runif(m)
y <- rbeta(m,a,b)##形成建议分布的抽样
xt[1] <- 0.5
for(i in 2:m){
        fy <- y[i]*dnorm(x,mu[1],sigma[1])+(1-y[i])*dnorm(x,mu
            [2],sigma[2])
        fx <- xt[i-1]*dnorm(x,mu[1],sigma[1])+(1-xt[i-1])*dnorm
            (x,mu[2],sigma[2])
        r <- prod(fy/fx)*(xt[i-1]^(a-1)*(1-xt[i-1])^(b-1))/(y[i
            ]^(a-1)*(1-y[i])^(b-1))
        if(u[i]<=r)
        xt[i] <- y[i]
        else
        xt[i] <- xt[i-1]
}
```

```
plot(xt,type = "l",ylab = "p")
hist(xt[1001:m],main = "",xlab = "p",prob = TRUE)
print(mean(xt[1001:m]))
```

对p的参数估计研究：

```
m <- 20000##样本链长度
xt <- numeric(m)
a <- 1
b <- 1##建议分布Beta(a,b)的参数
p_vec <- seq(from=0.01,to=0.99,by=0.01)
p_pre <- rep(0,length(p_vec))
p_mse <- rep(0,length(p_vec))
n <- 300##样本量
mu <- c(0,5)
sigma <- c(1,1)##正态分布参数，用来形成混合分布
##生成观察值
for(j in 1:length(p_vec)){
        i <- sample(1:2,size = n,replace = TRUE,prob = c(p_vec[
            j],1-p_vec[j]))
        x <- rnorm(n,mu[i],sigma[i])
        ##生成独立样本链
        u <- runif(m)
        y <- rbeta(m,a,b)##形成建议分布的抽样
        xt[1] <- 0.5
        for(i in 2:m){
                fy <- y[i]*dnorm(x,mu[1],sigma[1])+(1-y[i])*
                    dnorm(x,mu[2],sigma[2])
                fx <- xt[i-1]*dnorm(x,mu[1],sigma[1])+(1-xt[i
                    -1])*dnorm(x,mu[2],sigma[2])
                r <- prod(fy/fx)*(xt[i-1]^(a-1)*(1-xt[i-1])^(b
                    -1))/(y[i]^(a-1)*(1-y[i])^(b-1))
                if(u[i]<=r)
                xt[i] <- y[i]
```

```
            else
            xt[i] <- xt[i-1]
        }
        p_pre[j] <- mean(xt[1001:m])
        p_mse[j] <- (p_pre[j]-p_vec[j])^2
}
plot(p_vec,p_mse,col=2,main="不同p值估计mse结果",type="l")
```

两种Metropolis-Hastings算法对比：

```
path="C:\\Users\\user\\Documents\\统计实践与高维\\统计咨询与实
    践\\大报告\\code"
setwd(path)
source('批均值法计算Monte␣Carlo误差.R')
y_vec1 <- seq(from = 50, to = 950, by = 10)
y_vec2 <- seq(from = 50, to = 950, by = 10)
N <- 1000
iterations <- 2500
acc.problist1 <- rep(0,length(y_vec))
acc.problist2 <- rep(0,length(y_vec))
sdplist1 <- rep(0,length(y_vec))
sdplist2 <- rep(0,length(y_vec))
sdthetalist1 <- rep(0,length(y_vec))
sdthetalist2 <- rep(0,length(y_vec))
mcerrorplist1 <- rep(0,length(y_vec))
mcerrorplist2 <- rep(0,length(y_vec))
mcerrorthetalist1 <- rep(0,length(y_vec))
mcerrorthetalist2 <- rep(0,length(y_vec))
for(y in y_vec1){
        mu.theta <- 0
        s.theta <- 100
        prop.s <- 0.35
        theta <- numeric(iterations)
        current.theta <- 0
```

```
        acc.prob <- 0
        for(t in 1:iterations){
                prop.theta <- rnorm(1,current.theta,prop.s)
                loga <- ((prop.theta*y-N*log(1+exp(prop.theta))
                    )
                -(current.theta*y-N*log(1+exp(current.theta)))
                +dnorm(prop.theta,mu.theta,s.theta,log=TRUE)
                -dnorm(current.theta,mu.theta,s.theta,log =
                    TRUE))
                u <- runif(1)
                u <- log(u)
                if(u<loga){
                        current.theta <- prop.theta
                        acc.prob <- acc.prob+1
                }
                theta[t] <- current.theta
        }
        p <- exp(theta)/(1+exp(theta))
        acc.problist1[round((y-50)/10)] <- acc.prob
        sdplist1[round((y-50)/10)] <- sd(p)
        sdthetalist1[round((y-50)/10)] <- sd(theta)
        mcerrorplist1[round((y-50)/10)] <- mcerror.batch(p,
            batches = 50)
        mcerrorthetalist1[round((y-50)/10)] <- mcerror.batch(
            theta,batches = 50)
}
for(y in y_vec2){
        mu.theta <- 0
        s.theta <- 100
        prop.mu <- log(y/(N-y))
        mle.var <- 1/y+1/(N-y)
        w <- 1/(1+mle.var/s.theta^2)
        prop.s <- sqrt(mle.var*w)
        theta <- numeric(iterations)
        acc.prob <- 0
        current.theta <- 0
        for(t in 1:iterations){
```

```
                prop.theta <- rnorm(1,prop.mu,prop.s)
                loga <- ((prop.theta*y-N*log(1+exp(prop.theta))
                    )
                -(current.theta*y-N*log(1+exp(current.theta)))
                +dnorm(prop.theta,mu.theta,s.theta,log = TRUE)
                -dnorm(current.theta,mu.theta,s.theta,log =
                    TRUE)
                +dnorm(current.theta,prop.mu,prop.s,log = TRUE)
                -dnorm(prop.theta,prop.mu,prop.s,log = TRUE))
                u <- runif(1)
                u <- log(u)
                if(u<loga){
                        current.theta <- prop.theta
                        acc.prob <- acc.prob+1
                }
                theta[t] <- current.theta
        }
        p <- exp(theta)/(1+exp(theta))
        acc.problist2[round((y-50)/10)] <- acc.prob
        sdplist2[round((y-50)/10)] <- sd(p)
        sdthetalist2[round((y-50)/10)] <- sd(theta)
        mcerrorplist2[round((y-50)/10)] <- mcerror.batch(p,
            batches = 50)
        mcerrorthetalist2[round((y-50)/10)] <- mcerror.batch(
            theta,batches = 50)
}
plot(y_vec1,acc.problist1,type="l",xlab = "原假定的y值（胜场
    值）",ylab = "抽样接受次数",col="red")
lines(y_vec2,acc.problist2,type="l",xlab = "原假定的y值（胜场
    值）",ylab = "抽样接受次数",col="blue")
plot(y_vec1,sdplist1,type="l",xlab = "原假定的y值（胜场值）",
    ylab = "模拟胜率标准差",col="red")
lines(y_vec2,sdplist2,type="l",xlab = "原假定的y值（胜场值）",
    ylab = "模拟胜率标准差",col="blue")
plot(y_vec1,sdthetalist1,type="l",xlab = "原假定的y值（胜场值）
    ",ylab = "模拟theta标准差",col="red")
```

```
lines(y_vec2, sdthetalist2, type="l", xlab = "原假定的y值（胜场
    值）", ylab = "模拟theta标准差", col="blue")
plot(y_vec1, mcerrorplist1, type="l", xlab = "原假定的y值（胜场
    值）", ylab = "胜率的MC误差", col="red")
lines(y_vec2, mcerrorplist2, type="l", xlab = "原假定的y值（胜场
    值）", ylab = "胜率的MC误差", col="blue")
plot(y_vec1, mcerrorthetalist1, type="l", xlab = "原假定的y值（胜
    场值）", ylab = "theta的MC误差", col="red")
lines(y_vec2, mcerrorthetalist2, type="l", xlab = "原假定的y值（胜
    场值）", ylab = "theta的MC误差", col="blue")
```

时间序列Makeplots函数：

```
MakePlots <- function(model, ask = TRUE) {
        ## Make all the plots callable by plot.bsts.
        opar <- par(ask = ask)
        on.exit(par(opar))
        plot.types <- c("state", "components", "residuals",
        "prediction.errors", "forecast.distribution")
        for (plot.type in plot.types) {
                plot(model, plot.type, main = plot.type)
        }
        if (model$has.regression) {
                regression.plot.types <- c("coefficients", "
                   predictors", "size")
                for (plot.type in regression.plot.types) {
                        plot(model, plot.type)
                }
        }
}
```

SP500bsts函数：

```
library(bsts)
path="C:\\Users\\user\\Documents\\统计实践与高维\\统计咨询与实践\\大报告\\code"
setwd(path)
source("Makeplots函数.R")
#数据导入
data<-read.csv("D:/Rsoftware/时间序列与金融统计/非线性模型作业/data/S&P.csv")
d<-ts(data,frequency = 53,start = c(2016))
#数据处理
#figure1:导入收盘价数据
plot(d[,5])
x <- d[,5]
#figure2:转对数收益率
logd<-diff(log(d[,5]))
plot(logd,col="red",xlab="时间",ylab="对数收益率")
y_ori <- logd
y <- y_ori[1:220]
ss <- AddLocalLinearTrend(list(), y)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(model)
plot(pred)
plot(model, "state",main = "state")
plot(model, "components",main = "components")
plot(model, "residuals",main = "residuals")
plot(model, "prediction.errors",main = "prediction errors")
plot(model, "forecast.distribution",main = "forecast distribution")
#MakePlots(model)

ss2 <- AddSemilocalLinearTrend(list(), x)
model2 <- bsts(x, state.specification = ss2, niter = 500)
pred2 <- predict(model2, horizon = 12, burn = 100)
plot(model2)
```

```
plot(pred2)
plot(model2, "state",main = "state")
plot(model2, "components",main = "components")
plot(model2, "residuals",main = "residuals")
plot(model2, "prediction.errors",main = "prediction errors")
plot(model2, "forecast.distribution",main = "forecast
    distribution")
#MakePlots(model2)
```

SP500后项预测误差比较：

```
library(bsts)
path="C:\\Users\\user\\Documents\\统计实践与高维\\统计咨询与实
    践\\大报告\\code"
setwd(path)
source("Makeplots函数.R")
#数据导入
data<-read.csv("D:/Rsoftware/时间序列与金融统计/非线性模型作业/
    data/S&P.csv")
d<-ts(data,frequency = 53,start = c(2016))
#数据处理
#figure1:导入收盘价数据
plot(d[,5])
x <- d[,5]
#figure2:转对数收益率
logd<-diff(log(d[,5]))
plot(logd,col="red",xlab="时间",ylab="对数收益率")
y_ori <- logd
y_range <- as.vector(seq(from = 145, to=245, by = 1))
pre_error1 <- rep(0,length(y_range))
pre_error2 <- rep(0,length(y_range))
for(i in y_range){
        y <- y_ori[1:i]
        ss <- AddLocalLinearTrend(list(), y)
        model <- bsts(y, state.specification = ss, niter = 500)
```

```
        pred <- predict(model, horizon = 12, burn = 100)
        error <- dist(rbind(pred$mean,y_ori[i+1:i+12]),method =
            "euclidean")/12
        pre_error1[i-144] <- error
}
pre_error1
for(i in y_range){
        y <- y_ori[1:i]
        pre <- forecast(y,12)
        error <- dist(rbind(pre$mean,y_ori[i+1:i+12]),method =
            "euclidean")/12
        pre_error2[i-144] <- error
}
pre_error2
plot(y_range,pre_error1,xlab="时间变动范围",ylab="horizon=12预
    测误差",col="red",type="l")
lines(y_range,pre_error2,xlab="时间变动范围",ylab="horizon=12预
    测误差",col="blue",type="l")
```

Google公司股票数据后项误差预测比较：

```
data(goog)
d<-ts(goog,frequency = 260,start = c(2007))
y_ori <- d
ss <- AddSemilocalLinearTrend(list(), d)
model <- bsts(goog, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
y_range <- as.vector(seq(from = 800, to=1000, by = 1))
pre_error1 <- rep(0,length(y_range))
pre_error2 <- rep(0,length(y_range))
for(i in y_range){
        y <- y_ori[1:i]
        ss <- AddLocalLinearTrend(list(), y)
        model <- bsts(y, state.specification = ss, niter = 500)
```

```
        pred <- predict(model, horizon = 12, burn = 100)
        error <- dist(rbind(pred$mean,y_ori[i+1:i+12]),method =
            "euclidean")/12
        pre_error1[i-799] <- error
}
pre_error1
for(i in y_range){
        y <- y_ori[1:i]
        pre <- forecast(y,12)
        error <- dist(rbind(pre$mean,y_ori[i+1:i+12]),method =
            "euclidean")/12
        pre_error2[i-799] <- error
}
pre_error2
plot(y_range,pre_error1,xlab="时间变动范围",ylab="horizon=12预
    测误差",col="red",type="l")
lines(y_range,pre_error2,xlab="时间变动范围",ylab="horizon=12预
    测误差",col="blue",type="l")
```

作用于季节性数据AirPassenger:

```
data(AirPassengers)
y_ori <- log(AirPassengers)
y <- log(AirPassengers)[1:132]
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(model)
plot(pred)
pred$mean
y_ori[133:144]
```