STA 5107: Final Project
Reversible Jump Markov Chain Monte Carlo Algorithm for Model Selection in Linear Regression

Jingze Zhang
Department of Scientific Computing

Abstract

Linear regression is pretty useful in both research and industry. It is fast and simple, and quite power sometimes. It is a linear approach to modelling the relationship between a scaler output and some variables. In real application, the data we are given are usually more complicated. Some of the predictors are not useful, while some other important variables are unknown. So when applying linear regression, there could be hundreds and thousands of combinations of predictors and we have to select an appropriate combination to obtain the optimal model, which is called model selection. To get a Bayesian solution, I implement Reversible Jump Markov Chain Monte Carlo Algorithm and analyze its results using least square fitting and residual.

Introduction

Regression analysis is a set of statistical processes for estimating the relationships among variables. Linear regression is the simplest one. In multi-variable linear regression problems, the most important step is to select useful variables out of all the variables. In this report, I implement Reversible Jump Markov Chain Monte Carlo Algorithm and make experiments to test its performance on the accuracy of prediction and also compare it with least square fitting methods. Experiment results show that RJMCMC has some advantages and disadvantages.

Approaches

State the problem in statistic language, we want to find the coefficients for the model,

$$y = \sum_{i=1}^{n} x_i b_i + \epsilon$$

where $n < m$, x_is are the predictors, y is the response variable, and ϵ is the measurement noise. We are given k independent measurements, denoted in bold by $\boldsymbol{y}, \boldsymbol{X}$ and $\boldsymbol{\epsilon}$. We would like to find the optimal joint estimation of $\{n, b_1, b_2, \dots, b_n\}$

To set up a Bayesian formula, we need to define a joint posterior density:

$$f(n, \boldsymbol{b}_n, \boldsymbol{y}) \propto f(\boldsymbol{y}|n, \boldsymbol{b}_n) f(\boldsymbol{b}_n|n) f(n)$$

The terms on the right-hand side are:

- Likelihood function: $f(\boldsymbol{y}|n, \boldsymbol{b}_n) = (\frac{1}{\sqrt{2\pi{\sigma_0}^2}})^k e^{\frac{-1}{2\sigma_0^2}\|\boldsymbol{y}-\boldsymbol{X}_n\boldsymbol{b}_n\|^2}$
- Prior on b_n given n: $f(\boldsymbol{b}_n|n) = \left(\frac{1}{\sqrt{2\pi{\sigma_p}^2}}\right)^n e^{\frac{-1}{2\sigma_p^2}\|\boldsymbol{b}_n-\mu_b\|^2}$
- Prior on n: $f(n) = \frac{1}{m}$

The RJMCMC algorithm has the following steps:

(a) Select a candidate number n^* from the probability f(n)

(b) If $n^* \geq n$, generate a random vector $u \sim \mathrm{N}(0, \sigma_r I_{n^*})$.

The candidate coefficient vector is given by:

$$\boldsymbol{b}_{n^*} = \begin{bmatrix} \boldsymbol{b}_n \\ 0 \end{bmatrix} + \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{bmatrix}.$$

And the likelihoods are computed by:

$$h_1(\boldsymbol{u}) = \left(\frac{1}{\sqrt{2\pi{\sigma_r}^2}}\right)^{n^*} e^{\frac{-1}{2\sigma_r^2}\|\boldsymbol{u}\|^2}, \qquad h_2(\boldsymbol{u}_1) = (\frac{1}{\sqrt{2\pi{\sigma_r}^2}})^n e^{\frac{-1}{2\sigma_r^2}\|\boldsymbol{u}_1\|^2}$$

(c) If $n^* \leq n$, generate a random vector $u_1 \sim \mathrm{N}(0, \sigma_r I_{n^*})$. The candidate coefficient vector is given by:

$$\boldsymbol{b}_{n^*} = \boldsymbol{b}_n^1 + u_1$$

where $\boldsymbol{b}_n = \begin{bmatrix} \boldsymbol{b}_n^1 \\ \boldsymbol{b}_n^2 \end{bmatrix}$, $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{b}_n^2 \end{bmatrix}$.

And the likelihoods are computed by:

$$h_1(\boldsymbol{u}_1) = \left(\frac{1}{\sqrt{2\pi{\sigma_r}^2}}\right)^{n^*} e^{\frac{-1}{2\sigma_r^2}\|\boldsymbol{u}_1\|^2}, \quad h_2(\boldsymbol{u}) = (\frac{1}{\sqrt{2\pi{\sigma_r}^2}})^n e^{\frac{-1}{2\sigma_r^2}\|\boldsymbol{u}\|^2}$$

(d) Compute the acceptance-rejection function:

$$\rho = \min\left\{1, \frac{f(n^*, \boldsymbol{b}_{n^*}|\boldsymbol{y}) h_2}{f(n, \boldsymbol{b}_n|\boldsymbol{y}) h_1}\right\} = \min\left\{1, \frac{f(\boldsymbol{y}|n^*, \boldsymbol{b}_{n^*}) f(\boldsymbol{b}_{n^*}|n^*) h_2}{f(\boldsymbol{y}|n^*, \boldsymbol{b}_n) f(\boldsymbol{b}_n|n) h_1}\right\}$$

$$= \min\left\{1, \frac{e^{-(E_1-E_2)}\left(2\pi\sigma_p^2\right)^{\frac{n-n^*}{2}} e^{-\frac{1}{2\sigma_P^2}\left(\|\boldsymbol{b}_{n^*}-\mu_b\|^2 - \|\boldsymbol{b}_n-\mu_b\|^2\right)} h_2}{h_1}\right\}$$

where $E_1 = \frac{1}{2\sigma_0^2}\|\boldsymbol{y} - \boldsymbol{X}_{n^*}\boldsymbol{b}_{n^*}\|^2$ and $E_2 = \frac{1}{2\sigma_0^2}\|\boldsymbol{y} - \boldsymbol{X}_n\boldsymbol{b}_n\|^2$

(e) If $U \sim U[0,1]$ is less than ρ then set $(n, \boldsymbol{b}_n) = (n^*, \boldsymbol{b}_{n^*})$. Else, return to Step (a).

Experiments and Results

- Generate data

To simulate the algorithm, we generate toy data X and y. X contains 10 points and each point has 10 predictors. y is generated using first n predictors of X, where n is a random number between 1 and 10.

A brief look at data of one realization:

Table 1 sample data

X size(10*10)	n0	b_true size(4)	y size(10)
$\begin{bmatrix} 0.7762405 & 3.6664332 & \cdots & -0.9629241 \\ 7.0345276 & 2.8725370 & & -10.742857 \\ \vdots & & \ddots & \vdots \\ -0.470654 & -0.724895 & \cdots & -1.6370223 \end{bmatrix}$	4	$\begin{bmatrix} 2.2934485 \\ 2.2739783 \\ 1.8036615 \\ 1.6218531 \end{bmatrix}$	$\begin{bmatrix} 28.9905535 \\ 2.79143665 \\ \vdots \\ -15.071293 \end{bmatrix}$

We run RJMCMC for 100000 iterations on the data above and get the following result.

Table 2 result using RJMCMC

result	Run time	n	b
	42.048s	4	[2.27326457 2.31587494 1.80555815 1.56591108]

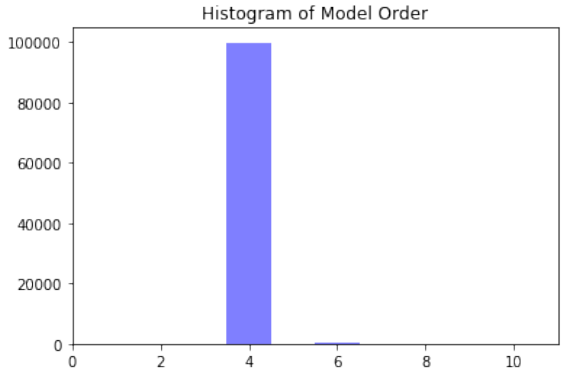


Figure 1 histogram of model order

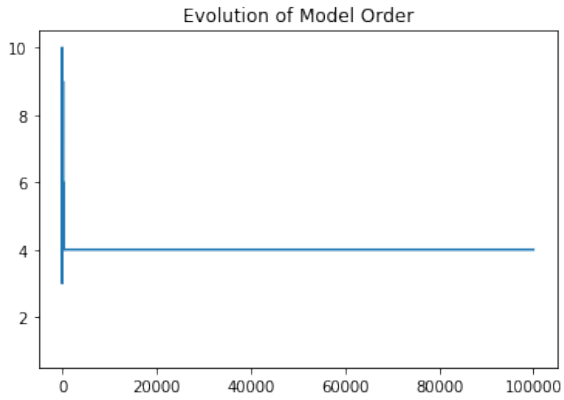


Figure 2 evolution of model order

Table 3 probability of n visiting different state

	1	2	3	4	5	6	7	8	9	10
probability	0	0	0.00005	0.99784	0.00002	0.00154	0.00015	0	0.00039	0.00001

Since we know that y is generated by using n0 = 4, we can simply tell that we get good answer because we get n = 4 = n0.

- Accuracy of getting true n

To evaluate the accuracy of this algorithm on find the true n, I repeat the above experiments 10 times and compare n with its true n0. To make difference, I generate data using different n0 range from 1 to 10. One of these 10 realizations give me the wrong answer. The histogram and evolution of 10 realizations are in appendix.

Table 4 comparation of n and n0 on 10 different realizations

Real n0	1	2	3	4	5	6	7	8	9	10
n	1	2	3	4	5	6	5	8	9	10

To better evaluate how good the result is, we introduce more methods to test how good is n and how good is the coefficients $\boldsymbol{b}$.

- Distance between b_ture and the b we get from RJMCMC

Because we know b_true, we can see how close between our result and b_ture. So in each iteration, I calculate their distance and plot as below.

Figure 3 evolution of distance between b and b_true

In the end, the distance is about 0.0405, which means b is very close to b_true. Compare with the evolution of model order, we can see that even if n is not changed, b is still being updated.

- Squared residual of prediction

The model we are trying to find is actually a regression plane in high dimensional space. A good regression plane will go across the point cloud in the middle. It should also can predict y given X. Based on this idea, I calculate the residual of prediction using

$$R = \sum\big(y - f(X)\big)$$

Because the residual can be positive or negative, to avoid the residuals cancel out, it is better to calculate the square of residual.

$$R^2 = \sum\big(y - f(X)\big)^2$$

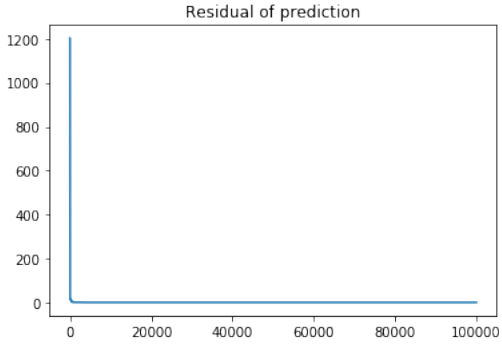


Figure 4 evolution of squared residual

In the end, square residual is 0.3419. This small residual tells that the model can predict well on X.

- Compare with least square fitting

All the test above are to test whether we get a good b. In order to test the performance of RJMCMC on finding a good n, I implement traditional least square fitting method and compare it with RJMCMC.

- Linear least squares fitting

Linear least squares fitting proceeds by minimizing the sum of squares of the vertical residuals,

$$minimize(R^2) = minimize(\sum_{i=1}^{k}[y_i - f(X_i, b_1, b_2, \dots, b_n)]^2)$$

where X_i is a n dimensional vector, k is the total number of points.

The condition for R^2 to be a minimum is that

$$\frac{\partial(R^2)}{\partial b_j} = 0$$

So

$$\frac{\partial(R^2)}{\partial b_1} = \sum_{i=1}^{k}[y_i - (b_1x_{1i} + b_2x_{2i} + \cdots + b_nx_{ni})]x_{1i} = 0$$

$$\frac{\partial(R^2)}{\partial b_2} = \sum_{i=1}^{k}[y_i - (b_1x_{1i} + b_2x_{2i} + \cdots + b_nx_{ni})]\,x_{2i} = 0$$

$$\vdots$$

$$\frac{\partial(R^2)}{\partial b_n} = \sum_{i=1}^{k}[y_i - (b_1x_{1i} + b_2x_{2i} + \cdots + b_nx_{ni})]\,x_{ni} = 0$$

In matrix form,

$$\begin{bmatrix} \sum x_{1i}^2 & \cdots & \sum x_{1i}x_{ni} \\ \vdots & \ddots & \vdots \\ \sum x_{1i}x_{ni} & \cdots & \sum x_{ni}^2 \end{bmatrix}\begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} \sum y_i x_{1i} \\ \vdots \\ \sum y_i x_{ni} \end{bmatrix}$$

Therefore, if we give a specific n, we can solve this linear system and we can get the coefficients b.

Applying to the example above, I get the following residuals.

Table 5 comparation of least square and RJMCMC

	Least square			RJMCMC
n	3	4	5	4
Squared residual	217.54	0.1068	0.1019	0.3419

Compare the result using least square method, when using n = 3, the residual is really big. So 3 predictors are not enough to predict correct y. When using n = 4, the residual 0.1068 is pretty small. So the 4th predictor is useful in predicting. When using 5 predictors, the residual is smaller than using 4 predictors. But the improvement is much smaller than from 3 predictors to 4 predictors. The 5th predictor doesn't bring much information into it. It may start learning the noise which will lead to overfitting. So 4 predictors is the optimal model. It can get good prediction without using redundant predictor. Also notice that the solution from RJMCMC is also n = 4. So RJMCMC found a good n.

Compare the residual of using least square and RJMCMC, least square method is 0.1, while RJMCMC gets 0.3. So RJMCMC can predict pretty accurate, but not as good as least square.

- Visualize both regression plane (when n0 = 2)

Another idea is to visualize the regression plane in feature space. Since it is in high dimensional space, we cannot visualize the plane. So I pick another realization where n0 = 2.

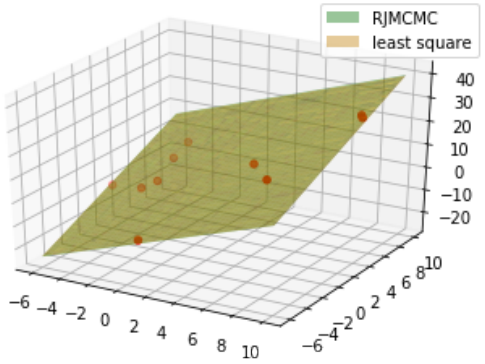


Figure 5 regression planes of RJMCMC and least square

The figure above shows that these 2 algorithms get almost the same plane in the feature space. Also notice that every point is on the plane, which means both methods are doing well on prediction.

Conclusion

RJMCMC algorithm can be used to find the best n and a relatively good set of b, which is a big innovation in feature selection. However, it also has some shortcoming. It is not always converging to the optimal solution even if using 100000 iterations. Also, it cannot select the most useful predictors by itself. The assumption that the optimal subset of predictors is simply the first n predictors reduces the possible number of models from 2^m to m. This algorithm is also time

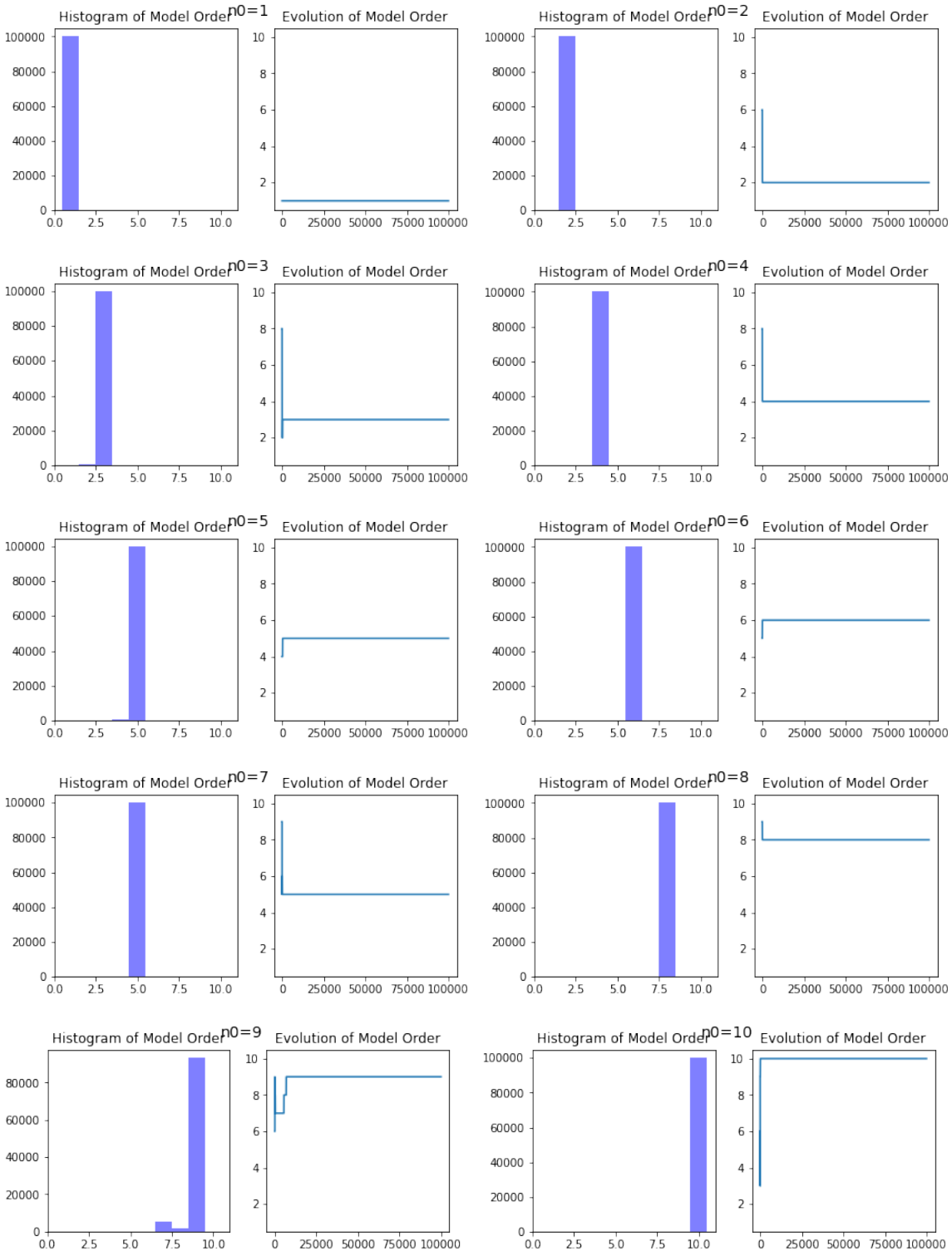
consuming. It takes 40s to find a good number of predictors from 10 possible solution. Without the assumption, RJMCMC will take much more time to select one from 2^{10} possible answers to find the optimal.

Bibliography

1. Green, P.J. (1995). "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination". Biometrika. 82 (4): 711–732.
2. A note on Reversible Jump Markov Chain Monte Carlo
3. David I. Hastie, Peter J. Green. Model Choice using Reversible Jump Markov Chain Monte Carlo
4. Farr W. M., Mandel I. and Stevens D. An efficient interpolation technique for jump proposals in reversible-jump Markov chain Monte Carlo calculations. 2. Royal Society Open Science
5. www.wikipedia.com
6. http://mathworld.wolfram.com/LeastSquaresFitting.html

Appendix:

- 10 different realizations start with different n0 range from 1 to 10



Most realizations give the correct n. However, the 7th realization gives a wrong answer. So RJMCMC is not always converge to the solution.

- Python code

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
import time
from mpl_toolkits.mplot3d import Axes3D

def PostF(y,X,b,sig0,mub,sigp):
    m = len(b)
    E = (np.linalg.norm(y -
X.dot(b))**2)/(2*sig0*sig0) +
(np.linalg.norm(b-mub)**2)/(2*sigp*sigp)
    E = E + 0.5*m*np.log(2*np.pi*sigp*sigp);
    return E

for n0 in range(1,11):
    m = 10
    print(n0)
    k = 10
    sig0 = 0.2
    sigp = 0.3
    sigr = 0.2
    mub = 2
    bT = mub + sigp*np.random.randn(n0)
    X = 5*np.random.randn(k,m)
    y = X[:,:n0].dot(bT) +
sig0*np.random.randn(k)
    nc = int(np.ceil(np.random.rand()*m))
    bc = mub + sigp*np.random.randn(nc)
    b0 = bc
    norm_dist = scipy.stats.norm(0,sigr)
    errors = []
    Rs = []
    N = 100000
    t1 = time.time()
    model = np.zeros(N)
    for it in range(N):
        n_p = int(np.ceil(np.random.rand()*m))
        model[it] = nc
        if n_p >= nc:
            U = sigr*np.random.randn(n_p)
            bp = np.concatenate((bc, np.zeros(n_p-
nc)), axis=0) + U
            hterm1 = np.prod(norm_dist.pdf(U))
            hterm2 = np.prod(norm_dist.pdf(U[:nc]))
        else:
            U1 = sigr*np.random.randn(n_p)
            U2 = bc[n_p+1:nc]
            bp = bc[:n_p] + U1
            U = np.concatenate((U1, U2), axis=0)
            hterm1 = np.prod(norm_dist.pdf(U))
            hterm2 = np.prod(norm_dist.pdf(U1))

        E1 = PostF(y,X[:,:n_p],bp,sig0,mub,sigp)
        E2 = PostF(y,X[:,:nc],bc,sig0,mub,sigp)
        rat  = np.exp(-E1+E2)*hterm2/hterm1
        uu = np.random.rand()
        if uu < np.min([1,rat]):
            bc = bp
            nc = n_p

errors.append(np.linalg.norm(np.concatenate((b
T,np.zeros(10-n0)), axis = 0)-
np.concatenate((bc,np.zeros(10-nc)), axis = 0)))
    R = np.sum((y-X[:,:len(bc)].dot(bc))**2)
    Rs.append(R)
print(nc)
print(time.time()-t1)
    plt.figure(figsize=(6.5,3))
    fig = plt.gcf()
    fig.suptitle('n0={}'.format(n0), fontsize=14)
    plt.subplot(1,2,1)
    plt.title('Histogram of Model Order')
    n, bins, patches = plt.hist(model, 10,
facecolor='blue', alpha=0.5, range=(0.5,10.5))
    print(n/N)
    plt.subplot(1,2,2)
    plt.plot(model)
    plt.title('Evolution of Model Order')
    plt.ylim(0.5,10.5)
    plt.show()
plt.title('Error between b_true and b')
plt.ylabel('$|bt-b|^2$')
plt.plot(errors)
plt.show()

plt.title('Residual of prediction')
plt.plot(Rs)
plt.show()

n = int(nc)
A = np.zeros((n,n))
B = np.zeros(n)
for i in range(n):
    for j in range(n):
        A[i,j] = X[:,i].dot(X[:,j])
    B[i] = y.dot(X[:,i])
b_ls = np.linalg.solve(A, B)
print(b_ls)
```

```
R_ls = np.sum((y-X[:,:n].dot(b_ls))**2)
print(R_ls)

xx1 = np.linspace(-6,10,100)
xx2 = np.linspace(-6,10,100)
X1, X2 = np.meshgrid(xx1,xx2)
yy = b_ls[0]*X1 + b_ls[1]*X2
yy2 = bc[0]*X1 + bc[1]*X2

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf1 = ax.plot_surface(X1, X2, yy2, alpha =
0.4, label = 'RJMCMC',color = 'green')
surf2 = ax.plot_surface(X1, X2, yy, alpha = 0.4,
label = 'least square',color = 'orange')
surf1._facecolors2d=surf1._facecolors3d
surf1._edgecolors2d=surf1._edgecolors3d
surf2._facecolors2d=surf2._facecolors3d
surf2._edgecolors2d=surf2._edgecolors3d
ax.scatter(X[:,0],X[:,1],y,c='r')
ax.legend()
plt.show()
```