实验报告——ARMA 模型预测和谱估计

Zhe Zhao
School of Integrated Circuits and Electronics, Beijing Institute of Technology
1120222262

1 实验目的

- 通过手搓 AR 模型来理解 ARMA 模型的含义
- 使用 AR 模型进行预测
- 使用 AR 模型进行谱估计

2 ARMA：滑动平均自回归模型

ARMA 模型的通式为：

$$x(n)+\sum_{i=1}^{p}a_i x(n-i)=e(n)+\sum_{j=1}^{q}b_j e(n-j) \tag{1}$$

其中，$x(n)$ 代表信号，$e(n)$ 代表白噪声，方差为 σ^2。如果某个信号的产生过程可以用该式描述，这个信号就被称作 ARMA 过程，该式称作信号的 ARMA(p,q) 模型。p 和 q 是 ARMA 模型的阶数。

接下来解释一下什么是 ARMA。左边 p 项 $x(n)$ 的延迟的加权求和称作 “自回归” (Auto-Regressive, AR)，右边 q 项 $e(n)$ 的加权求和叫做 “滑动平均” (Moving-Average, MA)。自回归指的是，用自身过去的数据来建立自身的回归模型。滑动平均指的是，用同一组系数在噪声序列上 “滑动”，用这组系数来求对应序列的加权平均。

我们可以用系统的观点来看待这个模型。把 arma 模型本身看作一个线性时不变系统，则代表信号 $x(n)$ 是该系统在白噪声 $e(n)$ 激励下的输出。对它做 z 变换：

$$A(z)X(z)=B(z)E(z) \tag{2}$$

系统函数为：

$$H(z)=\frac{B(z)}{A(z)}=\frac{1+\sum_{i=1}^{p}a_i z^{-i}}{1+\sum_{j=1}^{q}b_j z^{-j}} \tag{3}$$

Contributions are same. Names not listed in contribution order.

我们可以对分子分母进行多项式展开得到系统的零点和极点，这部分的理论和离散时间系统完全相同。如果只有零点，则系统相当于一个 FIR 滤波器，它的冲激响应必然是有限长的。此时的模型叫做 MA 模型：

$$x(n) = e(n) + \sum_{j=1}^{q} b_j e(n-j) \tag{4}$$

如果系统既有零点又有极点，则系统相当于一个 IIR 滤波器，它的冲激响应是无限长的。极点相当于在系统内引入了反馈。此时的模型叫做 AR 模型：

$$x(n) + \sum_{i=1}^{p} a_i x(n-i) = e(n) \tag{5}$$

基于自激振荡正反馈的正弦波发生器都可以用 AR 模型来描述，这相当于极点在单位圆上。如果用傅里叶展开的思想，把信号看作一系列正弦波的叠加，那仅用 AR 模型就可以描述相当广泛的一类信号，只要阶数足够高。所以，工程上基本都会使用 AR 模型来进行拟合，从而用于高精度功率谱估计、时间序列预测等任务。相对的，MA 模型则是在学习预测误差的特征。本文接下来的部分主要聚焦于 AR 模型。

3 AR 模型参数的确定

假定我们知道 AR 模型的阶数 p，我们接下来首先要想办法计算系数 $\{a_i\}$。常用的有最小二乘法、自相关法。这里介绍自相关法。

AR 模型可以写成这样的形式：

$$x(n) = \sum_{i=1}^{p} a_i x(n-i) + e(n) \tag{6}$$

注意，这里的 a_i 和上面的 a_i 是相反数的关系。这样就成了一个常见的线性回归模型。我们在左右两边乘上 τ 的延迟部分：

$$x(n)x(n-\tau) = (\sum_{i=1}^{p} a_i x(n-i) + e(n))x(n-\tau) \tag{7}$$

然后在两边取期望。如果我们假设信号是平稳的，那么有 $\mathbf{E}(x(n)x(n-\tau)) = R(\tau)$，以及信号 $x(n)$ 和噪声 $e(n)$ 应当在任何时刻都是不相关的，我们有：

$$R(\tau) = \sum_{i=1}^{p} a_i R(\tau - i) \tag{8}$$

因为模型是 p 阶的，所以我们一共需要 p 个方程才能解出所有的系数。令 τ 取遍 1 到 p，就能得到 p 个方程。这个方程组可以用矩阵的形式来表示：

$$\mathbf{Ra} = \mathbf{r} \tag{9}$$

其中，$\mathbf{R}$ 是 $x(n)$ 的自相关矩阵：

$$\mathbf{R} = \mathbf{E}(\mathbf{x}\mathbf{x}^T) = \begin{pmatrix} R(0) & R(1) & \cdots & R(p-1) \\ R(1) & R(0) & \cdots & R(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{pmatrix} \tag{10}$$

$\mathbf{x}$ 是数据矢量：

$$\mathbf{x} = \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(n-1) \end{pmatrix} \tag{11}$$

$\mathbf{a}$ 是 AR 系数：

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} \tag{12}$$

列矢量 $\mathbf{r}$ 可以写成这样的形式：

$$\mathbf{r} = \begin{pmatrix} R(0) \\ R(1) \\ \vdots \\ R(n-1) \end{pmatrix} \tag{13}$$

式 (9) 被称为 Yule-Walker 方程。它是假定信号平稳的前提下，AR 系数满足的方程。如果要求解这组系数，只需要：

$$\mathbf{a} = \mathbf{R}^{-1}\mathbf{r} \tag{14}$$

即可。求解这个方程需要算矩阵逆。对于一般的矩阵的逆，复杂性过高，而对于这个矩阵来说，可以利用矩阵的 Toeplitz 性和 Hermite 性，构造出递推的方式来求解方程，这就是大名鼎鼎的 levinson 算法。

接下来是我的计算 AR 系数的代码部分，这个代码是我自己手搓的，没有调用过于高级的库函数：

```
clear;
N = 100;
F1 = 0.05;  % 第一个正弦波频率
F2 = 0.08; % 第二个正弦波频率
n = 0:N-1;

theta1 = 2*pi*rand();
theta2 = 2*pi*rand();

x = sin(2*pi*F1*n+theta1) + 1*sin(2*pi*F2*n+theta2); % 两个正弦波叠加
noisestd = 0.05;  % 高斯白噪声标准差
```

```
x = x + noisestd * randn(1, N); % 添加高斯白噪声

% x = ar_generate(N,3,[0.3,-0.14,0.11],0.02);

p = 35;

% 接下来计算p阶自相关
% 先划定训练集
train_ratio = 0.5;
train_data_size = floor(N*train_ratio);
train_data = x(1:train_data_size);

% 然后在训练集上计算p阶自相关
acf = xcorr(train_data,'biased');
corr_porder = acf(train_data_size-p:train_data_size+p);
% 构造p阶自相关矩阵
T = toeplitz(acf(train_data_size:train_data_size+p));
corr_mat = T(1:p,1:p);
% for k = 0:p-1
%     corr_mat_k = circshift(corr_porder,-k);
%     corr_mat(k+1,:) = fliplr(corr_mat_k(2:p+1));
% end
% 构造yule方程，求解ar系数
R = T(2:end,1);
ar_coeff = (corr_mat\R)';
```

然后可以利用计算出的系数和原数据不断递推 $x(n)$，从而实现预测：

```
y_pred = zeros(1,N);
y_pred(1:train_data_size) = train_data;

for k = train_data_size + 1: N
    y_pred(k) = ar_coeff * flip((y_pred(k-p:k-1)))';
end

figure;
plot(y_pred,'g','linewidth',2);
hold on;
plot(x,'r');
plot(train_data_size,y_pred(train_data_size),'r*','linewidth',4);
figure;
error = y_pred - x;
plot(error);
hold on;
plot(train_data_size,error(train_data_size),'r*');
```

然后绘制出预测结果如图所示。

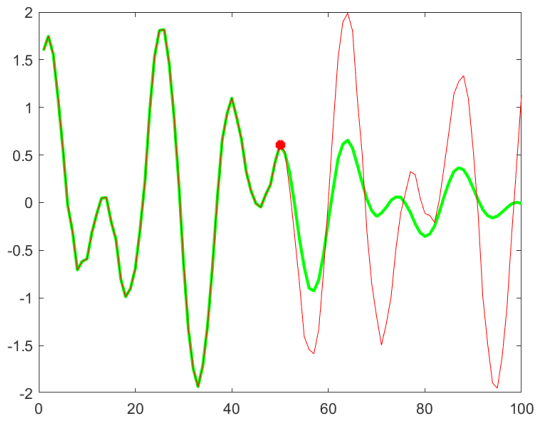

图 1: AR 模型预测结果

可以看到，AR 模型可以做到预测短程的趋势，但是预测长时间的序列便无能为力。我查阅了相关的资料：这是因为自相关矩阵 **R** 是正定的，正定性会使得求解出的模型的极点都在单位圆内，使得系统稳定。系统稳定就是指有界输入必为有界输出。所以按照这个方法求解出的系数一定最终是趋近于 0 的。

程序里还有个问题，计算 $x(n)$ 的自相关使用的 xcorr 函数，它有 biased 和 unbiased 两个选项。这里使用的是 biased，即有偏估计。按照我先前的认识，本来以为无偏估计肯定会更好，可是这里如果使用无偏估计，则会出现预测值发散的情形，即系统不稳定：

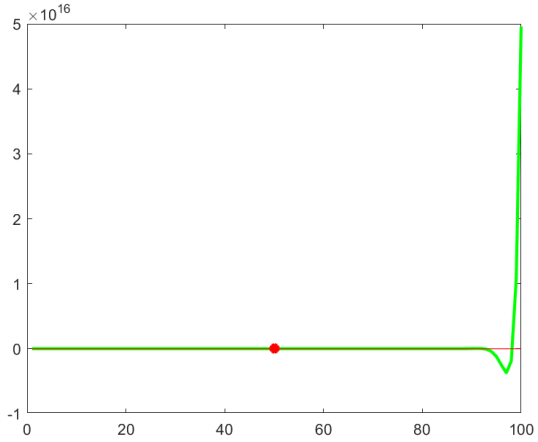

图 2: 发散的 ARMA 预测结果

这是因为，在平稳性的假设下，再假设信号有各态历经性，我们就能用时间平均代替统计平均：有偏估计的估计子是：

$$R(\tau) = \frac{1}{N} \sum_{n=1}^{N-\tau-1} x(n)x(n+\tau) \tag{15}$$

改估计子是有偏的，我们可以把它改进成无偏的：

$$R(\tau) = \frac{1}{N-\tau} \sum_{n=1}^{N-\tau-1} x(n)x(n+\tau) \tag{16}$$

按照教材《现代信号处理》的说法，一般都会采用有偏估计。首先，该估计子虽然有偏，但是是渐进无偏的。其次，该估计子虽然有偏，但是能保证是正定的，而无偏估计子失去了正定性，也就使得系统失去了稳定性。为了验证这一点，我们先后绘制了有偏估计求解的系统零极点和无偏估计下的系统零极点示意图。

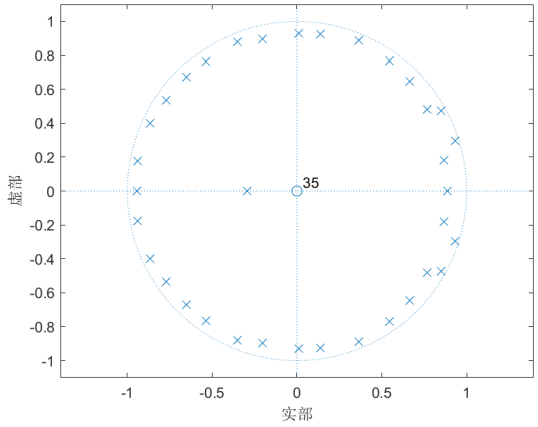

图 3: 有偏估计子的零极点图

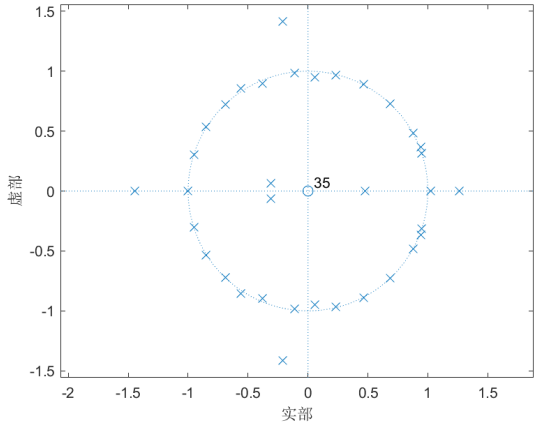

图 4: 无偏估计子的零极点图

结果显示，前者的极点都位于单位圆内，系统稳定，而后者的极点有些跑到了单位圆外，系统不稳定。

接下来探讨不同情形下的模型的预测能力，信号使用的两个正弦波的线性组合，并且添加了随机相位从而把正弦波调制成平稳随机过程，然后添加了加性高斯白噪声。

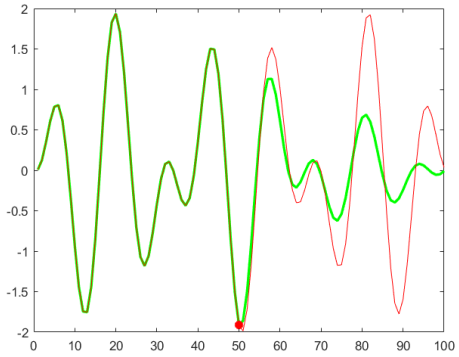

图 5: 噪声方差为 0 时的预测

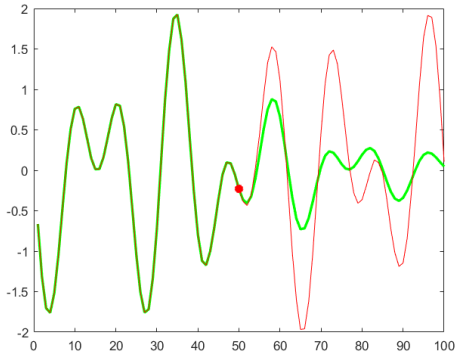

图 6: 噪声方差为 0.01 时的预测

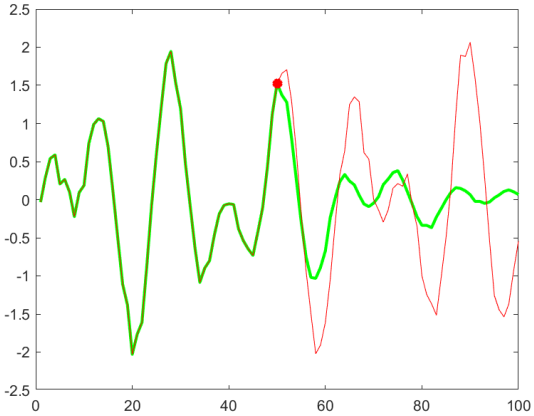

图 7: 噪声方差为 0.10 时的预测

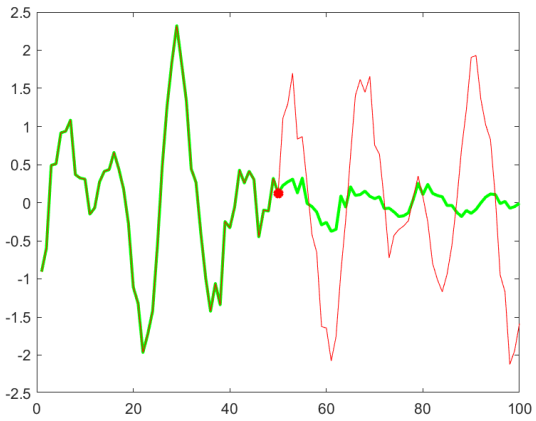

图 8: 噪声方差为 0.20 时的预测

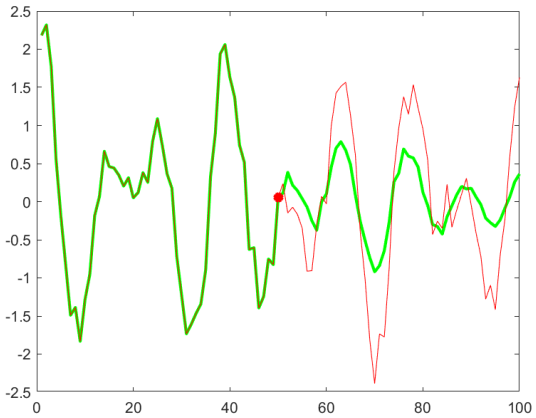

图 9: 噪声方差为 0.25 时的预测

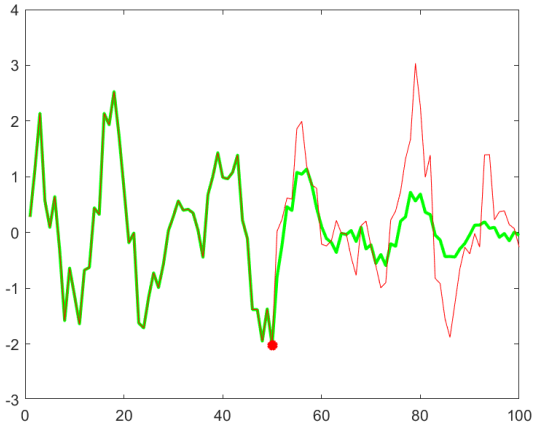

图 10: 噪声方差为 0.50 时的预测

可以看到，在不同的信噪比下，AR 模型基本上都具备短程预测能力，能预测出短期内数据的上升下降的趋势，但是都不能预测太长的数据。

4 AR 谱估计

AR 模型求解出的系数可以用于高分辨率谱估计。功率谱估计有两大类方法：非参数化方法、参数化方法。非参数化方法指的是周期图法，即通过 fft 来实现。因为 fft 等价于把信号视作周期的，所以叫周期图法。该方法固有的缺点是频率分辨率有限，最小为采样频率的 $\frac{1}{N}$。如果通过加窗来减少频谱泄露，则会导致频率分辨率进一步降低。所以基于 fft 的方法的天然缺陷就是频率分辨率有限。如果使用基于 AR 模型的参数化谱估计，则可以克服这一点。参数化的方法也可以理解为在原来模型的基础上添加了先验知识，自然可以让分辨率更高。

一个随机过程通过线性系统，则它的功率谱密度会发生变化：

$$S_Y(j\omega) = |H(j\omega)|^2 S_X(j\omega) \tag{17}$$

而在 ARMA 模型中，我们把白噪声视作输入，把我们的信号视作输出。因为白噪声的功率谱密度是常数，所以上式可以改写作：

$$S_X(j\omega) = \sigma^2 |H(j\omega)|^2 \tag{18}$$

而在 AR 模型中，$H(j\omega)$ 只有极点，所以有：

$$S_X(j\omega) = \sigma^2 \frac{1}{|1 - \sum_{i=1}^{p} a_i \mathrm{e}^{j\omega}|^2} \tag{19}$$

所以只要求出了 AR 系数，就可以获得 AR 功率谱。下面是加了噪声的两个不同频率正弦信号的叠加的功率谱密度，两个正弦信号的频率是 0.15 和 0.18，信号长度为 50，代码如下：

```
sigma2 = acf(train_data_size) - sum(ar_coeff .*
    acf(train_data_size+1:train_data_size+p)); % 预测误差方差

% 计算 AR 模型的功率谱密度
freqs = linspace(0, 1, 1000);  % 归一化频率范围 [0, 0.5]
psd_ar = zeros(size(freqs));
for i = 1:length(freqs)
    f = freqs(i);
    H = 1 ./ (1 - sum(ar_coeff .* exp(-1j * 2 * pi * f * (1:p)))); % 传递函数
    psd_ar(i) = sigma2 * abs(H)^2;  % PSD 公式
end

% 绘制功率谱
figure;
% subplot(311);
plot(freqs, (psd_ar), 'b', 'LineWidth', 1.5);
% subplot(312);
figure;
win = hamming(length(train_data));
plot((abs(fft(train_data .* win', 10*train_data_size)).^2+0.01)/train_data_size,
    'b', 'LineWidth', 1.5);
% subplot(313);
```

功率谱分别用 fft 和 AR 模型求得，将结果绘制如下：

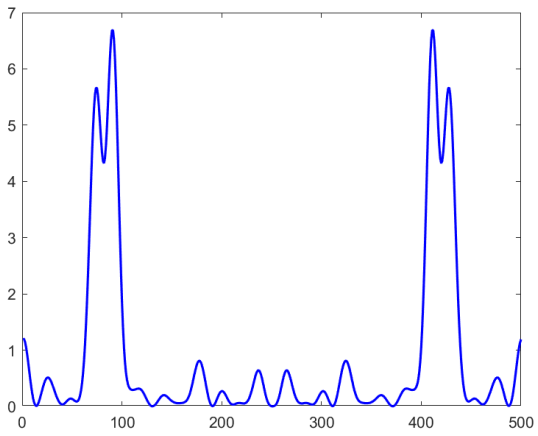

图 11: 周期图功率谱

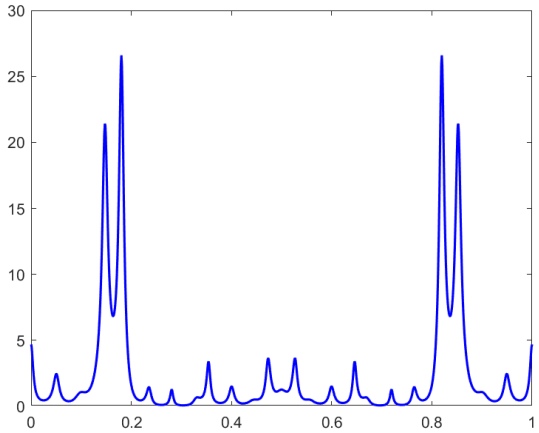

图 12: AR 功率谱

可以看到，AR 功率谱的谱峰更窄。这也意味着它更高的分别率。接下来，我们把两路正弦信号的频率改为分别为 0.15 和 0.16，继续绘制两种功率谱：

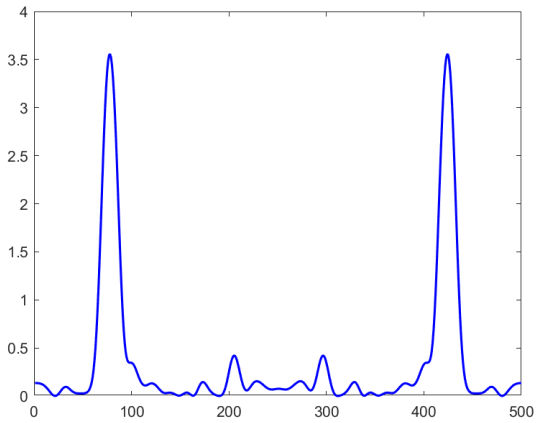

图 13: 周期图功率谱

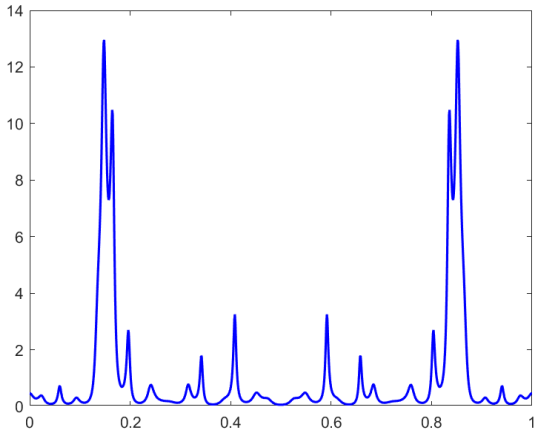

图 14: AR 功率谱

可以看到，前者已经无法将很接近的两个频率分开，而后者仍然能清晰地辨别出两个不同的频率。说明基于 AR 模型的功率谱密度的分辨率确实会更高。

5 附录

整个程序的代码：

```
clear;
N = 100;
F1 = 0.15;  % 第一个正弦波频率
F2 = 0.16; % 第二个正弦波频率
n = 0:N-1;

theta1 = 2*pi*rand();
theta2 = 2*pi*rand();

x = sin(2*pi*F1*n+theta1) + 1*sin(2*pi*F2*n+theta2); % 两个正弦波叠加
noisestd = 0.5;  % 高斯白噪声标准差
x = x + noisestd * randn(1, N); % 添加高斯白噪声
```

```
% x = ar_generate(N,3,[0.3,-0.14,0.11],0.02);

p = 35;

% 接下来计算p阶自相关
% 先划定训练集
train_ratio = 0.5;
train_data_size = floor(N*train_ratio);
train_data = x(1:train_data_size);

% 然后在训练集上计算p阶自相关
acf = xcorr(train_data,'biased');
corr_porder = acf(train_data_size-p:train_data_size+p);
% 构造p阶自相关矩阵
T = toeplitz(acf(train_data_size:train_data_size+p));
corr_mat = T(1:p,1:p);
% for k = 0:p-1
%     corr_mat_k = circshift(corr_porder,-k);
%     corr_mat(k+1,:) = fliplr(corr_mat_k(2:p+1));
% end
% 构造yule方程，求解ar系数
R = T(2:end,1);
ar_coeff = (corr_mat\R)';

y_pred = zeros(1,N);
y_pred(1:train_data_size) = train_data;

for k = train_data_size + 1: N
    y_pred(k) = ar_coeff * flip((y_pred(k-p:k-1)))';
end

figure;
plot(y_pred,'g','linewidth',2);
hold on;
plot(x,'r');
plot(train_data_size,y_pred(train_data_size),'r*','linewidth',4);
figure;
error = y_pred - x;
plot(error);
hold on;
plot(train_data_size,error(train_data_size),'r*');

sigma2 = acf(train_data_size) - sum(ar_coeff .*
    acf(train_data_size+1:train_data_size+p)); % 预测误差方差

% 计算 AR 模型的功率谱密度
freqs = linspace(0, 1, 1000);  % 归一化频率范围 [0, 0.5]
psd_ar = zeros(size(freqs));
for i = 1:length(freqs)
    f = freqs(i);
    H = 1 ./ (1 - sum(ar_coeff .* exp(-1j * 2 * pi * f * (1:p)))); % 传递函数
    psd_ar(i) = sigma2 * abs(H)^2;  % PSD 公式
end
```

```
% 绘制功率谱
figure;
% subplot(311);
plot(freqs, (psd_ar), 'b', 'LineWidth', 1.5);
% subplot(312);
figure;
win = hamming(length(train_data));
plot((abs(fft(train_data .* win', 10*train_data_size)).^2+0.01)/train_data_size,
    'b', 'LineWidth', 1.5);
% subplot(313);

% plot(abs(fft(acf, 10*train_data_size)), 'b', 'LineWidth', 1.5);

figure;
% 绘制零极点
ar_poly = [1, -ar_coeff];
zplane(1, ar_poly);
```