EE 779 Advanced topics in signal processing
Assignment 2 simulations

Swrangsar Basumatary Roll 09d07040

Dated: Sep/10/2012

First part using AR process

The codes for the AR process using Levinson-Durbin algorithm:
The function using the Levinson-Durbin algorithm:

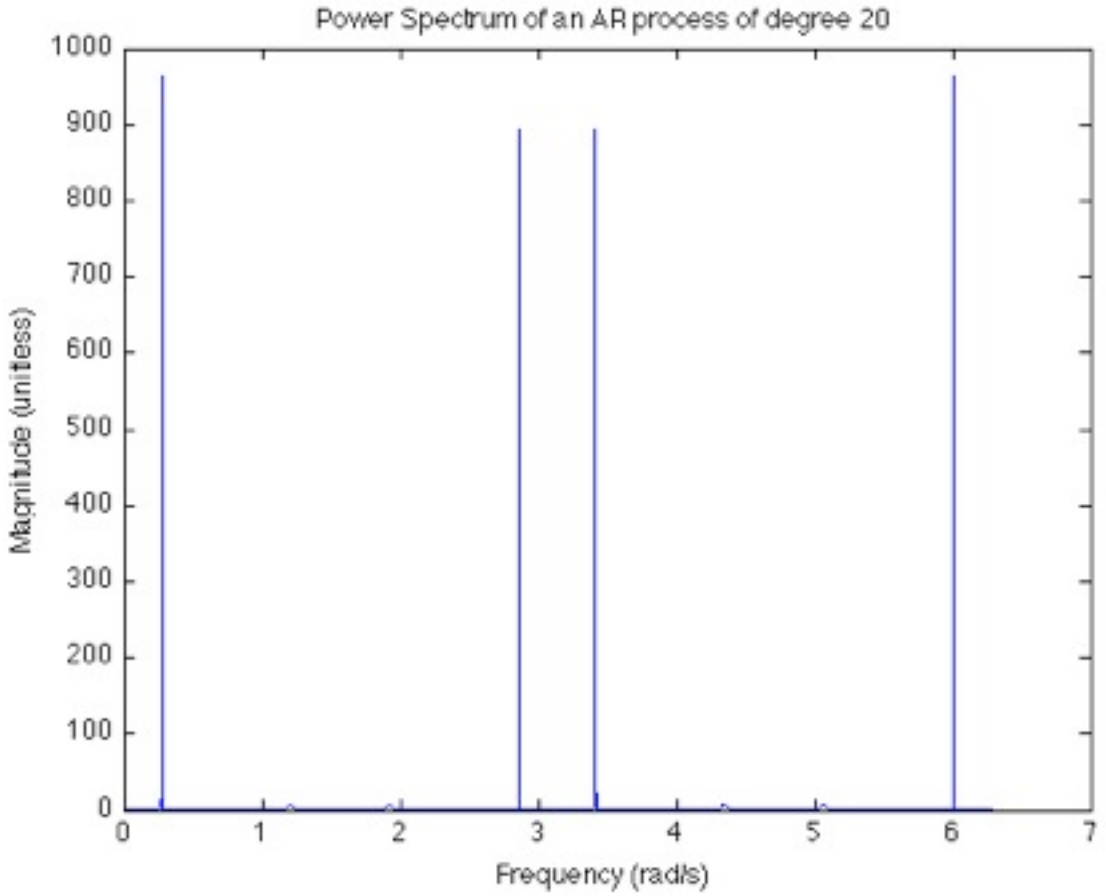
```
function [ coeff, b0 ] = levinsonDurbin( data , p)
%LEVINSONDURBIN A function to estimate parameters of a AR
process using
%     Levinson - Durbin algorithm. 'data' is the input data or
samples available
%     to us. 'p' is essentially the degree of the denominator
polynomial in
%     (1/z). Written by Swrangsar Basumatary.


rx = autocorr(data, p);
a = zeros(p+1);
e = zeros(1, p+1);
G = zeros(1, p+1);
reflected = zeros(1, p+1);
a(1, 1) = 1; e(1) = rx(1);

for j = 1:p;
    G(j) = rx(j+1);
    sum = 0;
    for i = 2:j;
        sum = sum + (a(j, i)*rx(j-i+1));
    end
    G(j) = G(j) + sum;
    reflected(j+i) = -G(j)/e(j);

    for i  = 2:j;
            a(j+1, i) = a(j, i) + (reflected(j+1)*a(j, j-i+1));
    end

    a(j+1, j+1) = reflected(j+1);
    e(j+1) = e(j) * (1-(abs(reflected(j+1))^2));
```

```
end
b0 = sqrt(e(p+1)); % the b(0) that we require in the numerator
of the transfer function
coeff = a(p+1, 2:end);  % coeff is the array of a(p)
coefficients
% length(coeff)

% the code below forms the equation of the power spectrum and
then plots it
res = 100000; % resolution some kind of
w = 0:res;
w = w.*2*pi*(1/res);
denominator = 1;
for k = 1:length(coeff);
    denominator = denominator + (coeff(k) * exp(-1i*k.*w));
end

numerator = abs(b0)^2;
denominator = abs(denominator).^2;
Px = numerator./denominator;

figure, plot (w, Px);
title(['Power Spectrum of an AR process of degree ',
num2str(p)]);
xlabel('Frequency (rad/s)');
ylabel('Magnitude (unitless)');
```

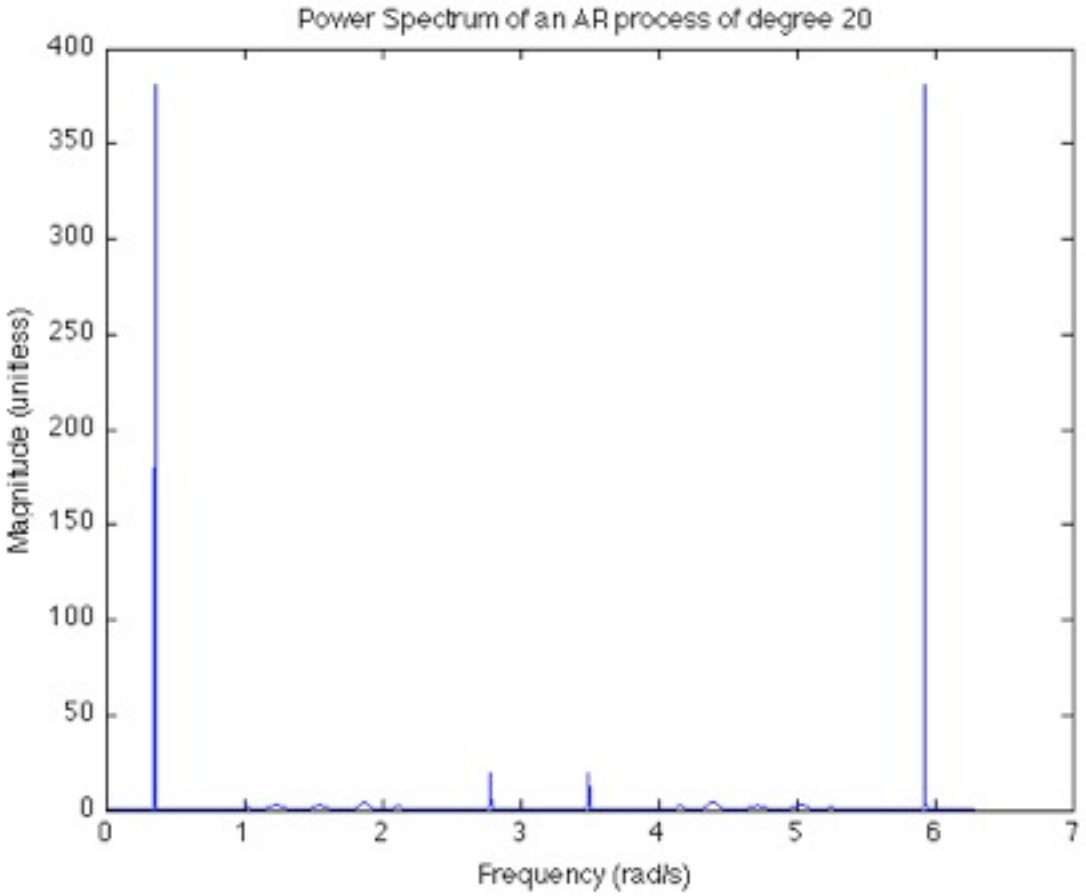
The script for the AR process simulation:

```
clear all; close all;

load 'sunspotdata';
levinsonDurbin(sunspot, 20);

clear all;
load 'lynxdata';
levinsonDurbin(lynx, 20);
levinsonDurbin(loglynx, 20);
```

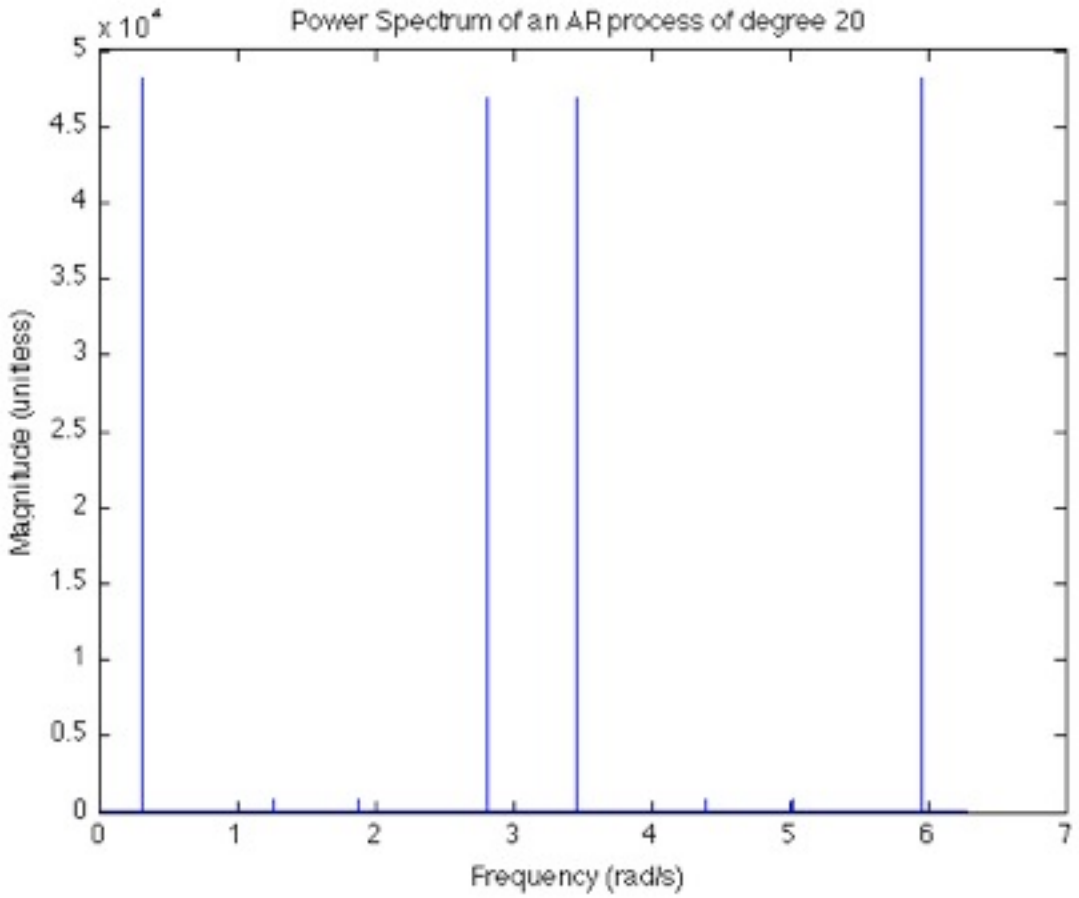
The plot for the ‘sunspot’ data:



The plot for the ‘lynx’ data:



The plot for the ‘loglynx’ data:



The log transformed data shows more sinusoidal frequencies in a better way because it reduces relative magnitudes of frequency components. The plots are for 0 to 2pi rad/s. They show components near 0.2, 2.4, 3.5 and 6 rad/s.

2nd part

The MA process using Levinson-Durbin method. We use polyfit() to estimate an all-zero filter from the all-pole filter that we get assuming AR process.

The code of the MA process function:

```
function [newCoeff] = levinsonDurbinMA( data , q)
%LEVINSONDURBINMA A function to estimate parameters of a MA
process using
%     Levinson - Durbin algorithm. 'data' is the input data or
samples available
%     to us. 'q' is essentially the order of the MA process. We
first find
```

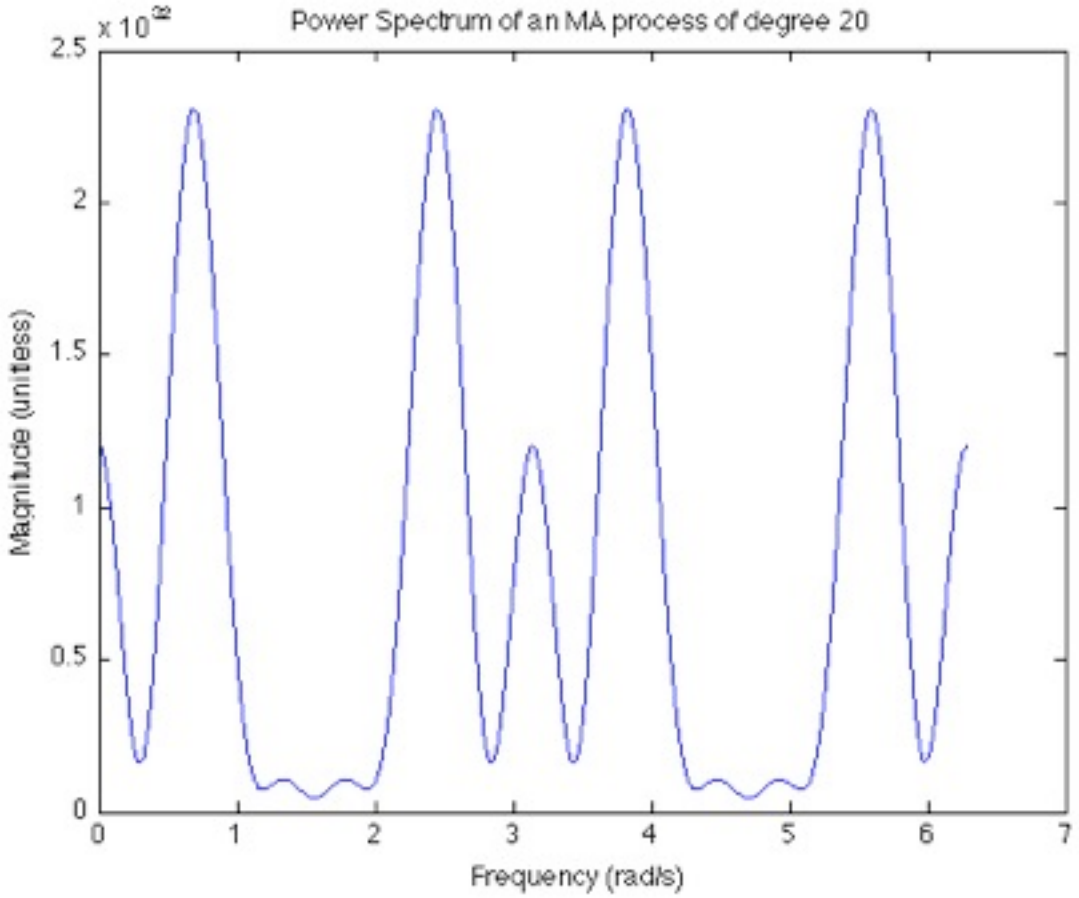
```
%     the coefficients for an AR model of the given data. We
then find a
%     fitting polynomial for the AR model. The coefficients of
this fitting
%     polynomial are the coefficients for the MA process.

p = min(q + 100, length(data)-1); % set 'p' of the AR model to
be larger than the 'q' of the MA model.
rx = autocorr(data, p);
a = zeros(p+1);
e = zeros(1, p+1);
G = zeros(1, p+1);
reflected = zeros(1, p+1);
a(1, 1) = 1; e(1) = rx(1);

for j = 1:p;
    G(j) = rx(j+1);
    sum = 0;
    for i = 2:j;
        sum = sum + (a(j, i)*rx(j-i+1));
    end
    G(j) = G(j) + sum;
    reflected(j+i) = -G(j)/e(j);

    for i  = 2:j;
            a(j+1, i) = a(j, i) + (reflected(j+1)*a(j, j-i+1));
    end

    a(j+1, j+1) = reflected(j+1);
    e(j+1) = e(j) * (1-(abs(reflected(j+1))^2));

end
b0 = sqrt(e(p+1))+eps; % the b(0) that we require in the
numerator of the transfer function
coeff = a(p+1, 2:end);  % coeff is the array of a(p)
coefficients
% length(coeff)
% plot(coeff);

% we have the a(p) coefficients. Now we find an inverse
polynomial of
% degree 'q' that fits the AR model.
x = -p:p;
numerator = b0;
denominator = 1;
for k = 1:length(coeff);
```

```
    denominator = denominator + (coeff(k) * (x.^k));
end
y = numerator./denominator;
% figure,plot(x,y);

newCoeff = polyfit(x, y, q);
newCoeff = flipdim(newCoeff, 2);


% the code below forms the equation of the power spectrum and
then plots it
res = 100000; % resolution some kind of
w = 0:res;
w = w.*2*pi*(1/res);
numerator = 0;
for k = 1:length(newCoeff);
    numerator = numerator + (coeff(k) * exp(-1i*k.*w));
end

numerator = abs(numerator).^2;
Px = numerator;

figure, plot (w, Px);
title(['Power Spectrum of an MA process of degree ',
num2str(q)]);
xlabel('Frequency (rad/s)');
ylabel('Magnitude (unitless)');
```

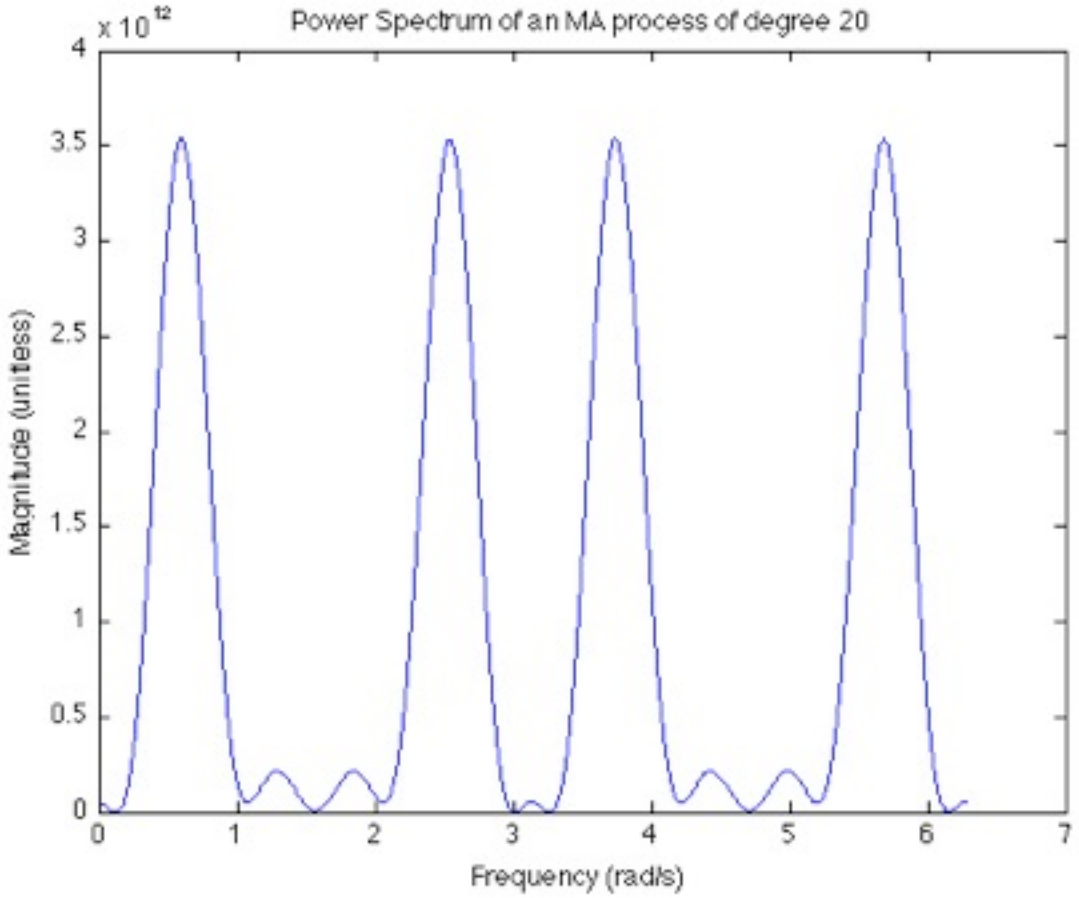
The script for the MA process:

```
clear all; close all;

load 'sunspotdata';
levinsonDurbinMA(sunspot, 20);

clear all;
load 'lynxdata';
levinsonDurbinMA(lynx, 20);
levinsonDurbinMA(loglynx, 20);
```

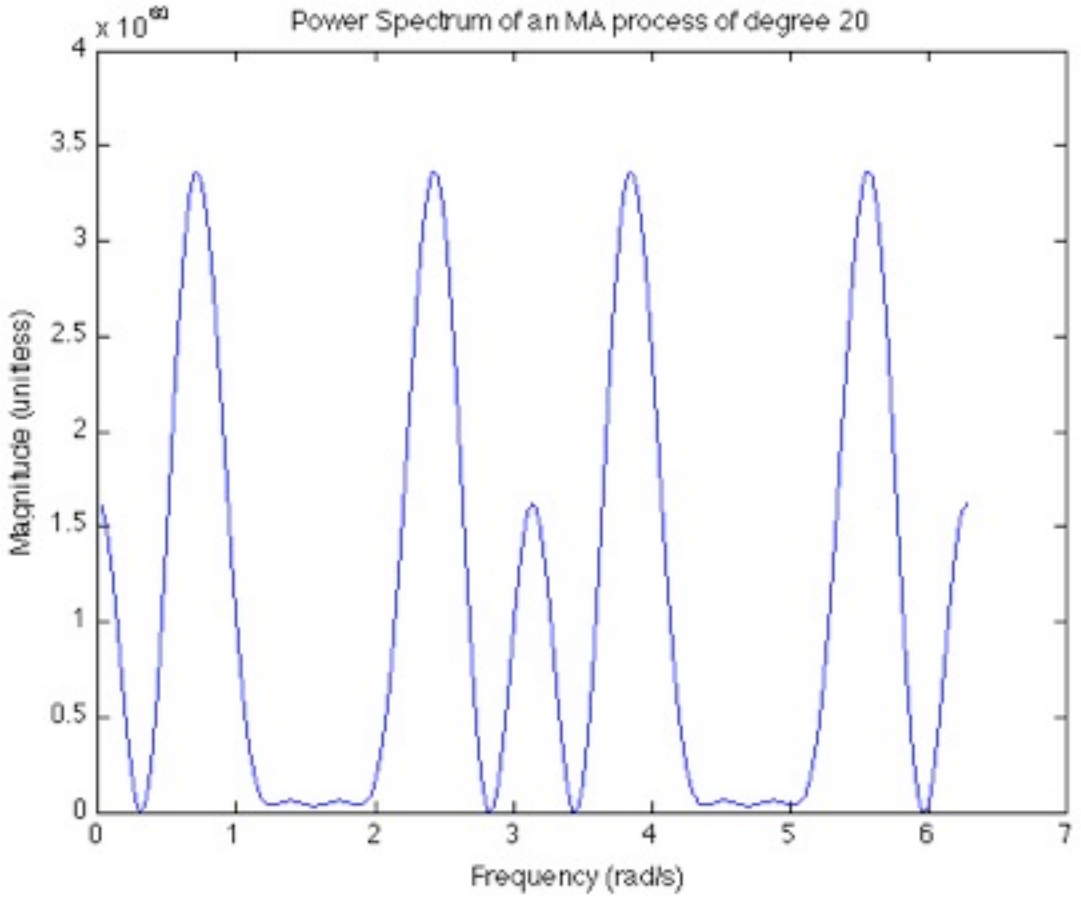
The plot for the ‘sunspot’ data:



The plot for the ‘lynx’ data:



The plot for the ‘loglynx’ data:



The peaks we get using MA process are not as sharp as the ones using AR process.

The ARMA part

The code using the ARMA process function:

```
function [ ap, bq ] = sbARMAModel( data, p, q )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

L = p+q;
if (L > (length(data) - 1))
    error('Not enough data for that order ARMA model.');
end
```

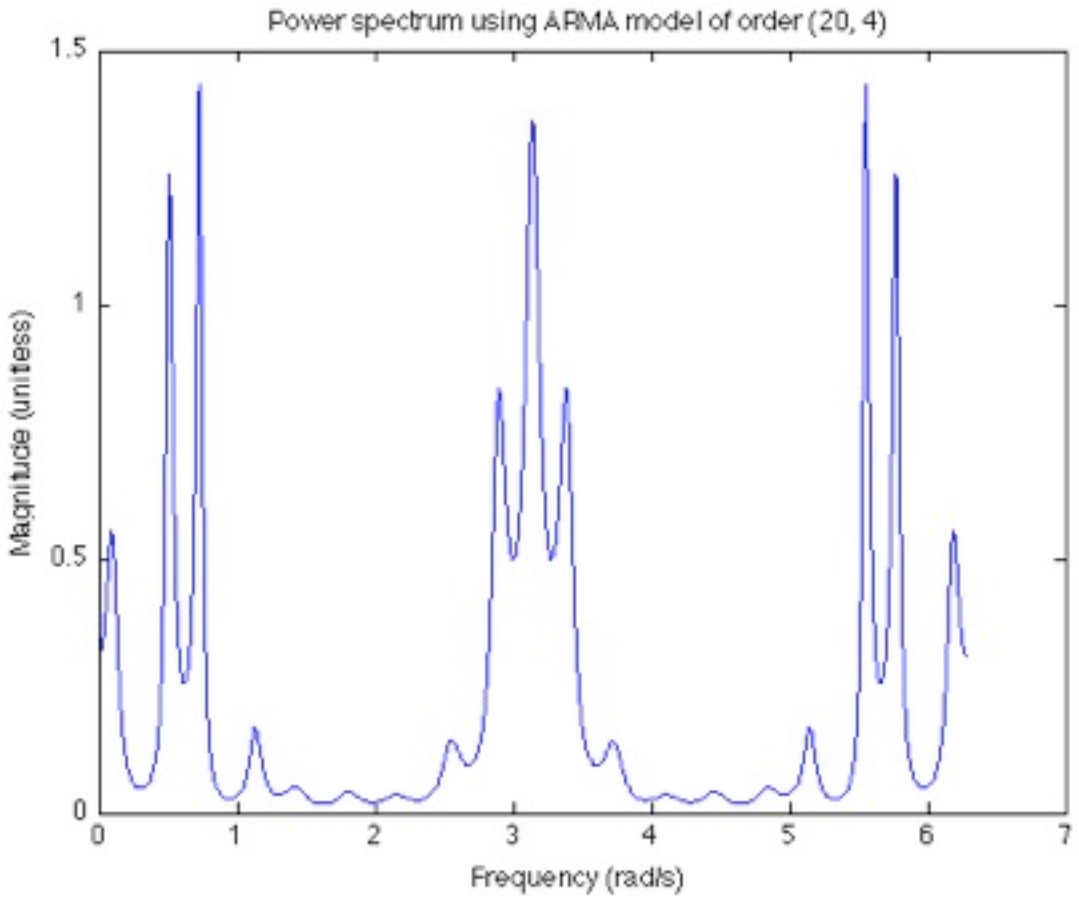
```
rx = autocorr(data, length(data)-1); % calculating the
autocorrelation values for different 'k's
rxmat = zeros(p); % the loop below builds the autocorrelation
matrix on the LHS of MYWE.
for k = 1:p
    for l = 1:p;
        if (q+k-l) >= 0
            rxmat(k, l) = rx(q+k-l+1);
        else
            rxmat(k, l) = rx(-(q+k-l)+1);
        end
    end
end

rxvect = rx(q+2:q+p+1)';% the vector at the RHS of Modified Yule
Walker equation

ap = (-rxvect)/rxmat; % the a(p) coefficients from index 1 to p.
ap1 = 1;
ap = cat(2, ap1, ap); % we concatenate a(0)=1 to the a(p) vector
so that we can use the vector for fft
dataLength = length(data);
xz = fft(data, dataLength)';% the fft of x(n) i.e. data
apz = fft(ap, dataLength); % the fft of the a(p) coefficients

% size(xz)
% size(apz)
yz = xz.*apz; % the fft of y(n) = x(n) passed through AR filter
of order 'p'.
yn = ifft(yz); % the y(n)

% y is the new data
ry = autocorr(yn, q+1);
z = tf('z');
BzBzinv = ry(1);
for k=1:q+1
    BzBzinv = BzBzinv + (2*ry(k+1)*(1/(z^k)));
end

[roots] = zpkdata(BzBzinv);
rootArray = roots{1};
indices = find(abs(rootArray) < 1);
newRootArray = zeros(1, length(indices));
for k = 1:length(indices)
    newRootArray = rootArray(indices(k));
end
```

```
s=tf('s');   % since the tf object can't be a string we have to
take 's' as 'z^-1'

Bz = 1;
for k = 1:length(newRootArray)
    Bz = Bz * (1-(newRootArray(k)*s));
end
[bznum] = tfdata(Bz);
bznum = bznum{1};
bznum = flipdim(bznum, 2);
bqdash = zeros(1, q+1);
for k=1:length(bznum)
    bqdash = bznum(k);
end
g = ry(1);
bqdashSq = bqdash.^2;
g = g/sum(bqdashSq);
bq = g.*bqdash;

res = 100000; % some kind of frequency resolution
w = 0:res;
w = w*(2*pi/res);
Py = 0;
for k=1:length(bq)
    Py = Py + (bq(k) * exp(-1i .* w .* (k-1)));
end
Py = abs(Py).^2;

Px = 0;
for k = 1:length(ap)
    Px = Px + (ap(k) * exp(-1i .* w .* (k-1)));
end
Px = abs(Px).^2;

Pw = Py./Px;
figure, plot(w, Pw);
title(['Power spectrum using ARMA model of order (', num2str(p),
', ', num2str(q), ')']);
xlabel('Frequency (rad/s)');
ylabel('Magnitude (unitless)');



end
```

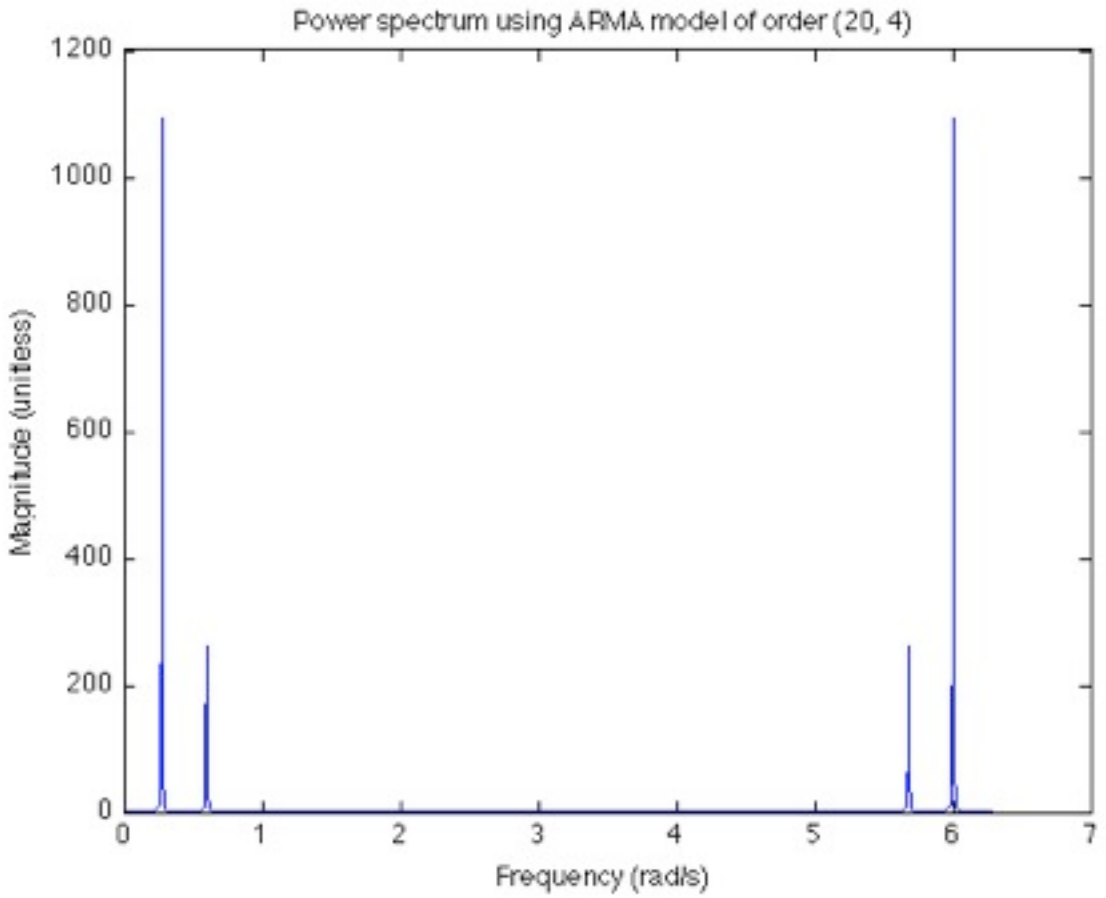
The script for the ARMA process simulation:

```
clear all; close all;

load 'sunspotdata';
sbARMAModel(sunspot, 20, 4);

clear all;
load 'lynxdata';
sbARMAModel(lynx, 20, 4);
sbARMAModel(loglynx, 20, 4);
```

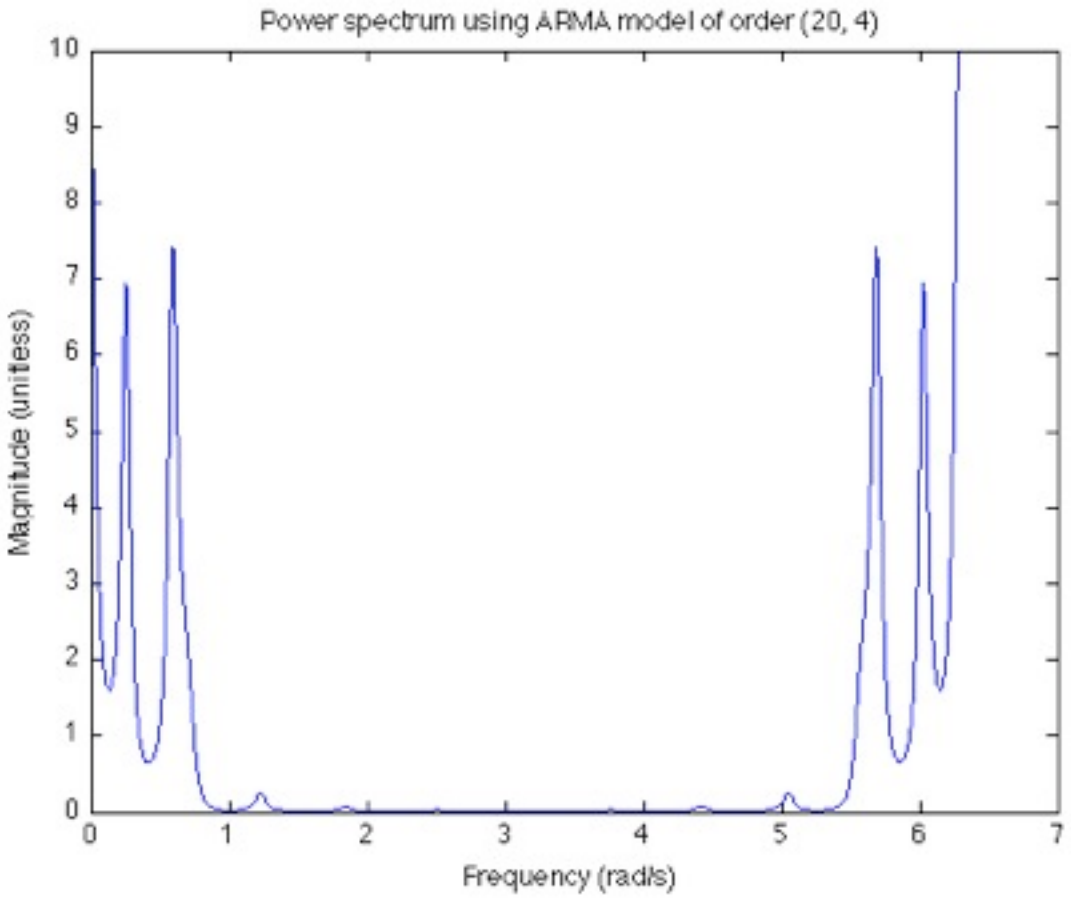
The plot for the ‘sunspot’ data:



The plot for the ‘lynx’ data:



The plot for the ‘loglynx’ data:



Just as in other processes the transformed data reveals more components. The ARMA processes are missing out some frequency components around 3 rad/s for both the ‘lynx’ and the ‘loglynx’ data.