STOCHASTIC SIGNALS AND SYSTEMS

WS 2018

ASSIGNMENT 3

LEAST - SQUARES ESTIMATION

Date: 08/01/2019

by

Shinde Mrinal Vinayak (Matriculation No.: 5021349)
Yama Sanath Kumar(Matriculation No.: 5021387)

guided by

Prof. Dr. -Ing. Dieter Kraus

Contents

List of Figures

Chapter 1

Functions of random variables

1.1 Polynomial model

Task: The signal model is given by

$$Y_i = a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p + Z_i$$

Let the observation $(x_i, y_i) : i = 1, 2, ..., N$ be given. Derive the LS-Estimate for the unknown parameters $a_i : i = 0, 1, ..., p$ and the estimate for the variance of the noise. How large has the number N of observations to be at least? Express with regard to a MATLAB implementation the problem in vector notation.

Solution:

We have the signal model equation to be,

$$Y_i = a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p + Z_i$$

The signal Y_i is a random variable deterministic with noise. p is the degree of the polynomials. In-order to determine this signal, we need to find its coefficients i.e., $a_0, a_1, ..., a_p$.

Let us assume that the noise Z_i is normally distributed and independent, i.e. $Z_i \sim N(o, {\sigma_Z}^2)$ with mean as 0 and variance, ${\sigma_Z}^2$. The noise in a signal is given as,

$$Z_i = Y_i - (a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p)$$

Therefore, our signal $Z_i \sim N(a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p, {\sigma_Z}^2)$ has a different mean and hence it is not identically distributed. Hence, there is a change in the distribution. Density function for this case is,

$$fY_i(y_i) = \frac{1}{\sqrt{2\pi {\sigma_Z}^2}} \exp\left\{ \frac{-1}{2{\sigma_Z}^2} (Y_i - (a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p)) \right\}$$

Since $(x_i, y_i) : i = 1, 2, ..., N$, the product of all the terms gives us the density function. i.e.,

$$fY_1...Y_N(y_1...y_N) = (2\pi {\sigma_Z}^2)^{-N/2} \exp\left\{ \frac{-1}{2{\sigma_Z}^2} \sum_{i=1}^{N} (Y_i - (a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p)) \right\}$$

We have to maximize the likelihood of its estimation to get the coefficient for the polynomial. We need to adjust the coefficient in such that the density function is maximum. Since x_1, y_i are known, we maximise with respect to vector a and ${\sigma_Z}^2$ to get the estimation.

Our maximizer estimator is,

$$(\underline{\hat{a}}, {\sigma_Z}^2) = \text{argmax}(a, {\sigma_Z}^2)$$

If we want to maximize the function, it is the same as minimizing the polynomial. Thus we define,

$$q(a_0, ..a_p) = \sum_{i=1}^{N} (Y_i - (a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p)^2$$

$$q(a_0, ..a_p) = \sum_{i=1}^{N} {Z_i}^2$$

This shows that we have to minimize the noise. We know,

$$Z_i = Y_i - (a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p)$$

The above equation when taken the individual terms for, $i = 1, 2, ..., N$ can be expressed in vector form like,

$$\begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{bmatrix} - \begin{bmatrix} 1 & x_1 & x_1^2 & . & . & . & x_1^p \\ 1 & x_2 & x_2^2 & . & . & . & x_2^p \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 1 & x_n & x_n^2 & . & . & . & x_n^p \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ . \\ . \\ . \\ a_p \end{bmatrix}$$

In matrix domain it is given as,

$$\underline{z} = \underline{y} - \underline{\underline{H}} \cdot \underline{a}$$

where $\underline{z}, \underline{a}$ and $\underline{y}$ are vectors and $\underline{\underline{H}}$ is a matrix of signal x_i.

If the number of independent equations are greater than the number of unknowns, we can determine the unknowns. Hence, the number of the observations should be greater than the number of the parameters that are to be found.

$$n \geq p + 1$$

The equation $q(a)$ is define as to be equal to,

$$q(a) = \sum_{i=0}^{N} (y_i - (a_0 + a_1 x_1 + a_2 x_1^2 + ... + a_p x_i^p)) = \sum_{i=1}^{N} z_i^2$$

where $q(a) = q(a_0, ..a_p)$ To find the sum, we have the property,

$$\sum_{i=1}^{N} z_i^2 = \underline{Z}^T \cdot \underline{Z} = \begin{bmatrix} Z_1 ... Z_N \end{bmatrix} \begin{bmatrix} Z_1 \\ . \\ . \\ . \\ Z_N \end{bmatrix}$$

Therefore we get,

$$q(a) = \underline{Z}^T \cdot \underline{Z}$$

After substituting the value for Z we get,

$$q(\underline{a}) = (\underline{y} - \underline{\underline{H}} \cdot \underline{a})^T (\underline{y} - \underline{\underline{H}} \cdot \underline{a})$$

In-order to minimise, we need to find the unknown parameters in a that satisfy,

$$\underline{\hat{a}} = \text{arg min } q(\underline{a})$$

Partially if we differentiate the equation with respect to a_i and equate them to 0, we get the minimum.

Now,

$$\frac{\partial q(a)}{\partial a_i} = \frac{\partial}{\partial a_i} (\underline{y} - \underline{\underline{H}} \cdot \underline{a})^T (\underline{y} - \underline{\underline{H}} \cdot \underline{a}) \quad \text{where,} \quad i = 0, 1, 2....p$$

$$= \frac{\partial}{\partial a_i} (\underline{y}^T \, \underline{y} - 2\underline{a}^T \underline{\underline{H}}^T \, \underline{y} + \underline{a}^T \, \underline{\underline{H}}^T \, \underline{\underline{H}} \, \underline{a})$$

Since,

$$(\underline{y}^T \, \underline{\underline{H}} \, \underline{a})^T = \underline{a}^T \, \underline{\underline{H}}^T \, \underline{y}$$

Therefore,

$$\frac{\partial q(a)}{\partial a_i} = 0 - 2\ \underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{y} + \underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{a} + \underline{a}^T\ \underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{e_i}$$

where e_i is a derivative of vector a with respect to i. The vector e_i will be an identity matrix as the coefficient of i's will have derivative as 1. Then,

$$\frac{\partial q(a)}{\partial a_i} = -2\ \underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{y} + 2\underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{a}$$

$$0 = -2\ \underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{y} + 2\underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{a}$$

$$\underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{y} = \underline{e_i}^T\ \underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{a}$$

If we take the individual terms, i.e., $i = 0, 1, 2....p$, we can express in matrix domain and we get,

$$\begin{bmatrix} {e_1}^T \\ {e_2}^T \\ . \\ . \\ . \\ {e_p}^T \end{bmatrix} \underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{a} = \begin{bmatrix} {e_1}^T \\ {e_2}^T \\ . \\ . \\ . \\ {e_p}^T \end{bmatrix} \underline{\underline{H}}^T\ \underline{y}$$

The two column matrices are identity matrices. Therefore,

$$\underline{\underline{H}}^T\ \underline{\underline{H}}\ \underline{a} = \underline{\underline{H}}^T\ \underline{y}$$

Now since $n \geq p + 1$ we get,

$$\underline{\hat{a}} = (\underline{\underline{H}}^T \cdot \underline{\underline{H}})^{-1} \underline{\underline{H}}^T\ \underline{y}$$

We can get to know the coefficient of the polynomial through the above equation. Thus the least square error is given by,

$$q(\underline{\hat{a}}) = (\underline{y} - \underline{\underline{H}} \cdot \underline{\hat{a}})^T(\underline{y} - \underline{\underline{H}} \cdot \underline{\hat{a}})$$

$$q(\underline{\hat{a}}) = (\underline{y} - \underline{\underline{H}} \cdot (\underline{\underline{H}}^T \cdot \underline{\underline{H}})^{-1}\underline{\underline{H}}^T \cdot \underline{y})^T(\underline{y} - \underline{\underline{H}} \cdot (\underline{\underline{H}}^T \cdot \underline{\underline{H}})^{-1}\underline{\underline{H}}^T \cdot \underline{y})$$

$$q(\underline{\hat{a}}) = (\underline{y} - \underline{\underline{P}} \cdot \underline{y})^T(\underline{y} - \underline{\underline{P}} \cdot \underline{y})$$

where P is a projection matrix having idempotent property.

$$\underline{\underline{P}} \cdot \underline{\underline{P}} = \underline{\underline{P}}$$

$$\underline{\underline{P^\perp}} \cdot \underline{\underline{P^\perp}} = \underline{\underline{P^\perp}}$$

where $\underline{\underline{P}}^\perp = (\underline{\underline{I}} - \underline{\underline{P}})$ Therefore we get the equation as follows,

$$q(\underline{\hat{a}}) = ((\underline{\underline{I}} - \underline{\underline{P}})\underline{y})^T((\underline{\underline{I}} - \underline{\underline{P}})\underline{y})$$

$$q(\underline{\hat{a}}) = \underline{y}^T(\underline{\underline{I}} - \underline{\underline{P}})^T(\underline{\underline{I}} - \underline{\underline{P}})\underline{y}$$

$$q(\underline{\hat{a}}) = \underline{y}^T(\underline{\underline{I}} - \underline{\underline{P}})\underline{y}$$

Therefore,

$$q(\underline{\hat{a}}) = \underline{y}^T\underline{\underline{P}}^\perp\underline{y}$$

Estimate for variance of noise:

SInce $Z_i \sim N(o, {\sigma_Z}^2)$, estimate of variance, ${\sigma_Z}^2$ is,

$${\sigma_Z}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(Z_i - \mu_Z)^2$$

Our signal is $Y_i \sim N(a_0 + a_1 x_i + a_2 {x_i}^2 + ... + a_p {x_i}^p, {\sigma_Z}^2)$ Therefore,

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}\sum_{i=1}^{N}(y_i - (\hat{a}_0 + \hat{a}_1 x_i + \hat{a}_2 {x_i}^2 + ... + \hat{a}_p {x_i}^p))^2$$

The term $\frac{1}{N-(P+1)}$ is used because we have the same degree as the number of coefficient causing an unbiased estimator.

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}(\underline{y} - \underline{\underline{H}} \cdot \underline{\hat{a}})^T(\underline{y} - \underline{\underline{H}} \cdot \underline{\hat{a}})$$

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}(\underline{y} - \underline{\underline{H}} \cdot (\underline{\underline{H}}^T \cdot \underline{\underline{H}})^{-1}\underline{\underline{H}}^T \cdot \underline{y})^T(\underline{y} - \underline{\underline{H}} \cdot (\underline{\underline{H}}^T \cdot \underline{\underline{H}})^{-1}\underline{\underline{H}}^T \cdot \underline{y})$$

Substituting the value of $\underline{\hat{a}}$ and P we get,

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}(\underline{y} - \underline{\underline{P}}\,\underline{y})^T\ (\underline{y} - \underline{\underline{P}}\,\underline{y})$$

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}((I - \underline{\underline{P}})\ \underline{y})^T\ ((I - \underline{\underline{P}})\ \underline{y}$$

where $(I - P) = \underline{\underline{P}}^{\perp}$ therefore,

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}(\underline{\underline{P}}^{\perp}\ \underline{y})^T\ (\underline{\underline{P}}^{\perp}\ \underline{y})$$

$${\sigma_Z}^2 = \frac{1}{N-(P+1)}\ \underline{y}^T\ \underline{\underline{P}}^{\perp}\ \underline{\underline{P}}^{\perp}\ \underline{y}$$

Therefore,

$$\sigma_Z{}^2 = \frac{1}{N-(P+1)}\,\underline{y}^T\,\underline{\underline{P}}^{\perp}\,\underline{y}$$

1.2 Power model and exponential model

Task: The power model is given by

$$Y_i = a \cdot x_i{}^b \cdot Z_i$$

Linearise the model assuming $(a > 0, x_i > 0)$ and $Z_i > 0$. Why are the measuring errors assumed to be multiplicative? Estimate the parameters a and b as well as the variance of the measurement error after linearization. How large has to be the number N of observations at least? The exponential model is defined by

$$Y_i = a \cdot b^{x_i} \cdot Z_i$$

Carry out all the tasks mentioned above also for this model.

Solution:

The power model is given by,

$$Y_i = a \cdot x_i{}^b \cdot Z_i$$

To get the value of the coefficients, we can define $q(a, b)$ where,

$$q(a,b) = \sum_{i=1}^{N}(Y_i - a\;x_i{}^b)^2$$

This equation leads to a non - linear solution because of the power b. Therefore to get the iterative solution, we can linearize the power model by taking logarithms on both sides of the equation.

Therefore,

$$\log Y_i = \log\, a + b \log\, x_i + \log\, Z_i$$

Hence, we get a linear model of the form,

$$\tilde{Y}_i = \tilde{a} + b\tilde{x}_i + \tilde{Z}_i, \quad i = 1, 2, ..., N$$

where, $\log\, Y_i = \tilde{Y}_i,\ \log\, x_i = \tilde{x}_i,\ \log\, a_i = \tilde{a}_i,$ and $\log\, Z_i = \tilde{Z}_i$ are all positive.

The measuring errors are assumed to be multiplicative so that after linearization they become additive. And that assumption can be made because a bias in the estimation of the parameters a and b result in a multiplicative part in the error of the measurement dependent on x_i. In matrix notation, we can represent as,

$$\begin{bmatrix} \tilde{Y}_1 \\ \tilde{Y}_2 \\ . \\ . \\ . \\ \tilde{Y}_N \end{bmatrix} = \begin{bmatrix} 1 & \tilde{x}_1 \\ 1 & \tilde{x}_2 \\ . & . \\ . & . \\ . & . \\ 1 & \tilde{x}_N \end{bmatrix} \begin{bmatrix} \tilde{a} \\ b \end{bmatrix} + \begin{bmatrix} \tilde{Z}_1 \\ \tilde{Z}_2 \\ . \\ . \\ . \\ \tilde{Z}_N \end{bmatrix}$$

The matrix can be written as $\underline{\tilde{Y}} = \underline{\underline{H}} \cdot \underline{\tilde{a}} + \underline{\tilde{Z}}$. This is the same linear model as in 1.1 and we can apply the concluded results. We can determine the estimates for the parameters a, b from the following equations,

$$\hat{\underline{\tilde{a}}} = (\underline{\underline{\tilde{H}}}^T \cdot \underline{\underline{\tilde{H}}})^{-1} \cdot \underline{\underline{\tilde{H}}}^T \cdot \underline{\tilde{y}}$$

Thus we have,

$$q(\underline{\tilde{a}}) = (\underline{\tilde{y}} - \underline{\underline{\tilde{H}}}\, \underline{\tilde{a}})^T \cdot (\underline{\tilde{y}} - \underline{\underline{\tilde{H}}}\, \underline{\tilde{a}})$$

The variance of the measurement error after linearization is given by,

$$\hat{\sigma_z^2} = \frac{\underset{\sim}{\underline{y}}^T \cdot \underline{\underline{p}}^{\perp} \cdot \underline{y}}{(N-(p+1))}$$

In this model, $(p+1) = 2$ number of observations as two parameters are unknown, we need to determine two equations, i.e., $N > P+1$.

$$\hat{\sigma_z^2} = \frac{\underset{\sim}{\underline{y}}^T \cdot \underline{\underline{p}}^{\perp} \cdot \underline{y}}{(N-2)}$$

where,

$$\underline{\underline{p}}^{\perp} = I - \tilde{\underline{\underline{H}}}(\tilde{\underline{\underline{H}}}^T \cdot \tilde{\underline{\underline{H}}})^{-1} \cdot \tilde{\underline{\underline{H}}}^T$$

Thus we have an accurate realization.

The exponential model is given as,

$$Y_i = a \cdot b^{x_i} \cdot Z_i$$

By taking the logarithms on both sides we get,

$$\log Y_i = \log a + x_i \log b + \log Z_i$$

Let $\log Y_i = \tilde{Y_i}$, $\log x_i = \tilde{x_i}$, $\log a_i = \tilde{a_i}$, and $\log Z_i = \tilde{Z_i}$ Then we can write the equation in matrix domain and we get,

$$\begin{bmatrix} \overset{\sim}{Y_1} \\ \overset{\sim}{Y_2} \\ . \\ . \\ . \\ \overset{\sim}{Y_N} \end{bmatrix} = \begin{bmatrix} \log Y_1 \\ \log Y_2 \\ . \\ . \\ . \\ \log Y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ . & . \\ . & . \\ . & . \\ 1 & x_N \end{bmatrix} \begin{bmatrix} \tilde{a} \\ \tilde{b} \end{bmatrix} + \begin{bmatrix} \overset{\sim}{Z_1} \\ \overset{\sim}{Z_2} \\ . \\ . \\ . \\ \overset{\sim}{Z_N} \end{bmatrix}$$

The matrix can be written as $\underset{-}{\tilde{y}} = \underline{\underline{H}} \cdot \underset{-}{\tilde{a}} + \underset{-}{\tilde{Z}}$. This is the same model as in section 1.1. Therefore we can determine the coefficients in the same way as we did in the previous case. Thus the variance of the measurement error after linearization is given by,

$$\hat{\sigma_z^2} = \frac{\underset{-}{\tilde{y}}^T \cdot \underline{\underline{p}}^{\perp} \cdot \underset{-}{\tilde{y}}}{(N-2)}$$

where,

$$\underline{\underline{p}}^{\perp} = I - \overset{\sim}{\underline{\underline{H}}}(\overset{\sim}{\underline{\underline{H}}}^T \cdot \overset{\sim}{\underline{\underline{H}}})^{-1} \cdot \overset{\sim}{\underline{\underline{H}}}^T$$

The number of observations should also be at least 2 in order to determine the parameter. Thus we have an accurate realization.

Chapter 2

Exercises with Matlab

2.1 Standard normal distribution

Task: Now data for four different signal models have to be created. Generate therefore an on [0, 1] uniformly distributed random sequence `x1` and a standard normal distributed random sequence `z1`, each with a length of 100 values. Set in both cases first the initial value to 0. Type now these commands in MATLAB in the following order:

```
x = x1*5+2;  z = z1*sqrt(0.004);  xy1 = [x exp(1+x*0.6+z)];

x = x1*4*pi;  z = z1*sqrt(0.05);  xy2 = [x 2*sin(x+1+z)];
```

`x = x1*5;  z = z1;  xy3 =`$[\mathrm{x} - 0.6 * \mathrm{x}.^3 + 0.9 * \mathrm{x}.^2 + 3 * \mathrm{x} + 4.5 + \mathrm{z}];$

```
x = x1*5;  z = z1*sqrt(0.004);  xy4 =[x  exp(0.3+log(x)*0.5+z)];

x = x1*2*pi;  z = z1*0.5+6;  xy =[z.*cos(x)+4 z.*sin(x)+2];

save  dat3_1  xy1 xy2 xy3 xy4

save  dat3_1  xy
```

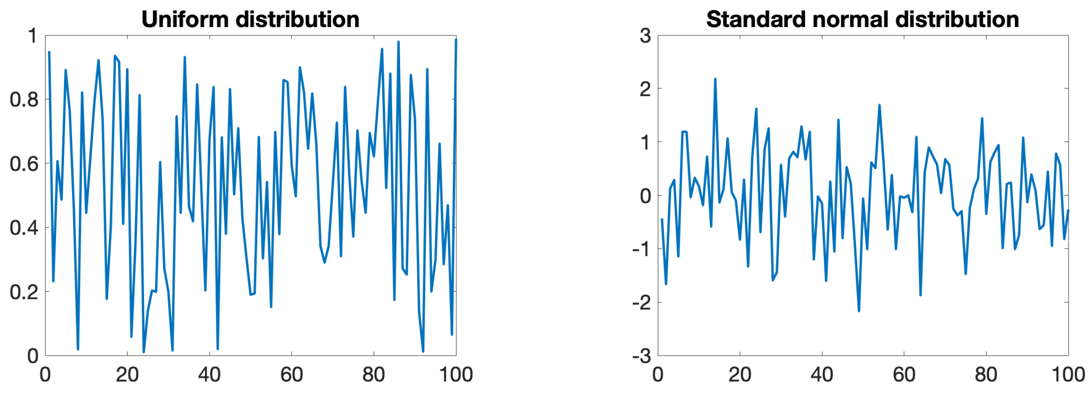
Solution: At first, a uniformly distributed random sequence, `x1` and a standard normal distributed random sequence, `z1` with length as 100 are created. Then, four different signal models are created using the equations mentioned in the task. The four models are: exponential model, sine model, polynomial model and power model. These models are finally saved in `dat3_1`.

MATLAB code:

```
clear all;
close all;
clc;

N = 100;
rand ('state',0);
x1 = rand(N,1);
figure()
plot(x1,'LineWidth',2.5)
title('Uniform distribution')
set(gca,'FontSize',22)

randn ('state',0);
z1 = randn(N,1);
figure()
plot(z1,'LineWidth',2.5)
title('Standard normal distribution')
set(gca,'FontSize',22)
```

```
x = x1*5 + 2;
z = z1*sqrt(0.004);
xy1 = [x exp(1+x*0.6+z)];

x = x1*4*pi;
z = z1*sqrt(0.05);
xy2 = [x 2*sin(x+1)+z];

x = x1*5;
z = z1;
xy3 = [x -0.6*x.^3+0.9*x.^2+3*x+4.5+z];

x = x1*5;
z = z1*sqrt(0.004);
xy4 = [x exp(0.3+log(x)*0.5+z)];

x = x1*2*pi;
z = z1*0.5+6;
xy = [z.*cos(x)+4 z.*sin(x)+2];

save dat3_1 xy1 xy2 xy3 xy4
save dat3_2 xy
```

Output: The plot of uniform distribution and standard normal distribution is shown in Figure 2.1.



(a) Uniform distribution (b) Standard normal distribution

Figure 2.1: Distributions

Inference: Four different models which represent exponential, sine, polynomial and power model are created. Also, the uniform distribution with mean as 0 and variance as 1 is generated.

2.2 Model assignment

Task: Load `dat3_1` that includes the four 100 x 2 matrices `xy1, xy2, xy3` and `xy4.`The two column vectors of each matrix correspond to the observations x_i and y_i of a particular signal model. Show each dataset x_i, y_i in a diagram and assign to each matrix a model. (Note: For the model assignment you can exploit the fact that $\log g(x)$ behaves in case of the power model like a logarithmic function while $\log g(x)$ depends in case of the exponential model only linearly on x.)

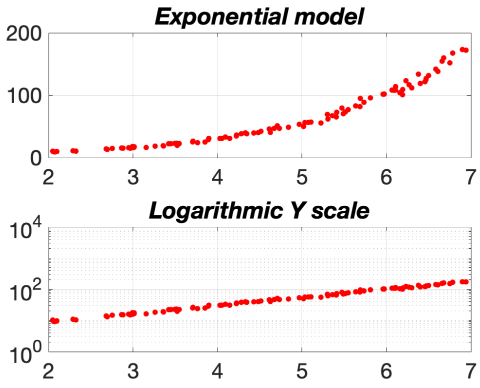
Solution: The four models are first loaded and then are plotted. From the first set of data, i.e., `xy1` exponential graph is plotted. And similarly rest of the models are graphed from rest of the data sets. The exponential and power model is also represented using logarithmic Y scale.

MATLAB code:

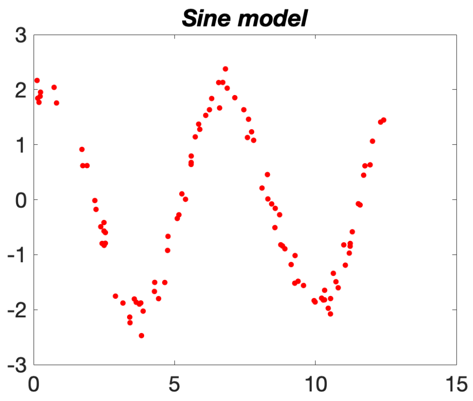
```
clear all;
close all;
clc

load dat3_1;
figure()
subplot(2,1,1)
plot(xy1(:,1),xy1(:,2),'.r','MarkerSize',15)
grid on
set(gca,'FontSize',22)
title('Exponential model','FontAngle', 'italic', 'FontWeight', 'bold')
subplot(2,1,2)
semilogy(xy1(:,1),xy1(:,2),'.r','MarkerSize',15)
grid on
set(gca,'FontSize',22)
title('Logarithmic Y scale','FontAngle', 'italic', 'FontWeight', 'bold')

figure()
plot(xy2(:,1),xy2(:,2),'.r','MarkerSize',15)
set(gca,'FontSize',22)
title('Sine model','FontAngle', 'italic', 'FontWeight', 'bold')

figure()
plot(xy3(:,1),xy3(:,2),'.r','MarkerSize',15)
grid on
set(gca,'FontSize',22)
title('Polynomial model','FontAngle', 'italic', 'FontWeight', 'bold')

figure()
subplot(2,1,1)
plot(xy4(:,1),xy4(:,2),'.r','MarkerSize',15)
grid on
set(gca,'FontSize',22)
```

```
title('Power Model','FontAngle', 'italic', 'FontWeight', 'bold')
subplot(2,1,2)
semilogy(xy4(:,1),xy4(:,2),'.r','MarkerSize',15)
grid on
ytick = 10.^(0:3);
yticklab = cellstr(num2str(round(-log10(ytick(:))), '10^-^%d'));
set(gca,'YTick',ytick,'YTickLabel',yticklab,'TickLabelInterpreter', ...
    'tex','FontSize',22)
title('Logarithmic Y scale','FontAngle', 'italic', 'FontWeight', 'bold')
```

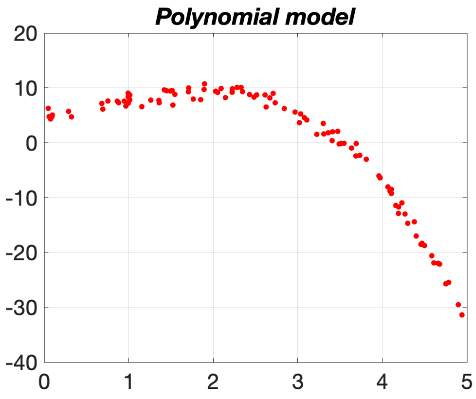
Output: The output after execution of the code is shown in Figure 2.2



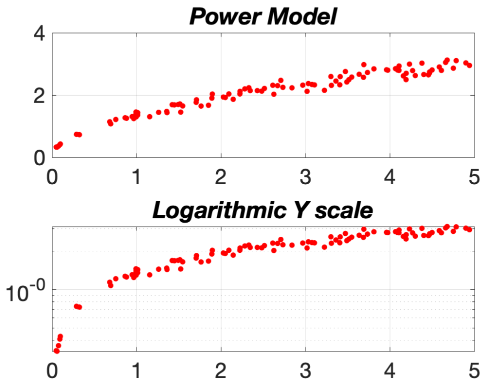
(a) Exponential model



(b) Sine model



(c) Polynomial model



(d) Power model

Figure 2.2: Four different models

Inference: We plotted the generated data sets and assigned them a proper model. The logarithm graph of the exponential model is observed to be a straight line because for the exponential model $g(x)$, $\log(g(x))$ will depend linearly on x. And similarly, for $g(x)$ as a power model, $\log(g(x))$ depends logarithmically on x.

2.3 LS-Estimation of different models

Task: Estimate each of the parameter a, b and the variance of the measurement errors ${\sigma_z}^2$ of the linearized models, i.e. of the exponential model, the power model and the sine model. Write therefore a function `LSE` that is able to deal with these three models and with a polynomial model of arbitrary order.

Solution: We estimate the parameters and the variance of the measurement errors ${\sigma_z}^2$ of the exponential model, the power model, the sine model and the polynomial model. We wrote a function `LSE` to obtain the above mentioned parameters. The inputs of this function are the data sets from `dat3_1` i.e. (x_i, y_i), the model whose parameters we want and the integer p. In the case of polynomial model, p represents the order of the polynomial, while for the rest of the model $p = 1$. We ask the user to input the value of p for the polynomial model.

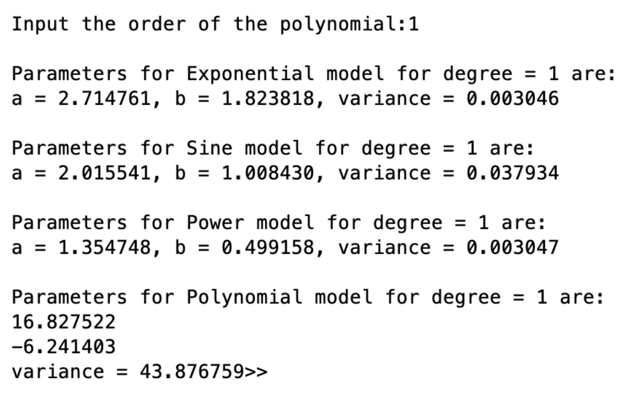
MATLAB function `LSE`:

```
function [coefficient , sigma] = LSE(x, y, model, p)
if strcmp(model,'exponential') == 1
    H = [ones(length(x),1) x];
   [av, sigma] = getVariance(length(x), H, log(y), p);
    coefficient = exp(av);
    fprintf('\nParameters for Exponential model for degree = %d are:',p)
    fprintf('\na = %f, b = %f, variance = %f', ...
        coefficient(1,:),coefficient(2,:),sigma)
elseif strcmp(model,'power') == 1
    H = [ones(length(x),1) log(x)];
    [av, sigma] = getVariance(length(x), H, log(y), p);
    coefficient(1,:) = exp(av(1));
    coefficient(2,:) = av(2);
    fprintf('\n\nParameters for Power model for degree = %d are:',p)
     fprintf('\na = %f, b = %f, variance = %f', ...
        coefficient(1,:),coefficient(2,:),sigma)
elseif strcmp(model,'sine') == 1
    H=[sin(x) cos(x)];
   [av, sigma] = getVariance(length(x), H, y, p);
    coefficient(2,:) = atan(av(2)/av(1));
    coefficient(1,:) = av(1)/cos(coefficient(2,:));
    fprintf('\n\nParameters for Sine model for degree = %d are:',p)
    fprintf('\na = %f, b = %f, variance = %f', ...
        coefficient(1,:),coefficient(2,:),sigma)
elseif strcmp(model,'polynomial') == 1
    for i = 1:length(x)
        for j = p+1:-1:1
            H(i,j) = (x(i).^(j-1));
        end
    end
    [coefficient, sigma] = getVariance(length(x), H, y, p);
    fprintf('\n\nParameters for Polynomial model for degree = %d are:',p)
    for i = 1:length(coefficient)
         fprintf('\n%f',coefficient(i,:))
    end
    fprintf('\nvariance = %f',sigma)
end


    function [av, sigma] = getVariance(ln, H, y1, order)
       P = H*(inv(H'*H))* H';
       PO = eye(ln) - P;
       av = (inv(H'*H))*H'*y1;
       sigma = y1'*PO'*y1/(ln-(order+1));
    end

end
```

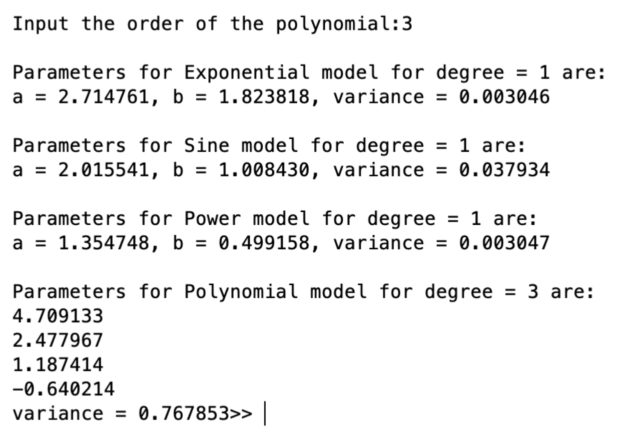
MATLAB code:

```
clear all
close all
clc

load dat3_1;
p = input('Input the order of the polynomial:');
[coefficient_exp , sigma_exp] = LSE(xy1(:,1),xy1(:,2),'exponential',1);
[coefficient_sin , sigma_sin] = LSE(xy2(:,1),xy2(:,2),'sine',1) ;
[coefficient_pow , sigma_pow] = LSE(xy4(:,1),xy4(:,2),'power',1);
[coefficient_ply , sigma_ply] = LSE(xy3(:,1),xy3(:,2),'polynomial',p);
```

Output: The output after execution of the code is shown below. We gave input as $p = 1$ and $p = 3$. The code prints the parameters for each of the model.



```
Input the order of the polynomial:1

Parameters for Exponential model for degree = 1 are:
a = 2.714761, b = 1.823818, variance = 0.003046

Parameters for Sine model for degree = 1 are:
a = 2.015541, b = 1.008430, variance = 0.037934

Parameters for Power model for degree = 1 are:
a = 1.354748, b = 0.499158, variance = 0.003047

Parameters for Polynomial model for degree = 1 are:
16.827522
-6.241403
variance = 43.876759>>
```



```
Input the order of the polynomial:3

Parameters for Exponential model for degree = 1 are:
a = 2.714761, b = 1.823818, variance = 0.003046

Parameters for Sine model for degree = 1 are:
a = 2.015541, b = 1.008430, variance = 0.037934

Parameters for Power model for degree = 1 are:
a = 1.354748, b = 0.499158, variance = 0.003047

Parameters for Polynomial model for degree = 3 are:
4.709133
2.477967
1.187414
-0.640214
variance = 0.767853>>
```

Inference: Estimation of the wanted parameters for our generated data is done successfully.

2.4 Estimating the order of a polynomial

Task: Estimate the order p of the polynomial model. Therefore you have to depict the estimated variance versus the model order $p = 1, 2, ..., 10$. Consider the order that provides the smallest variance as the correct order. Estimate for that order the parameters a_i for $i = 1, 2, ..., p$.

Solution: The same function `LSE` is used to obtain the variance of the polynomial model. The polynomial order p, is taken from 1 to 10 and the parameters at each order are shown in the output.

MATLAB code:

```
clear all
close all
clc

load dat3_1;
for p = 10:-1:1
[~, sigma_ply(p)] = LSE(xy3(:,1),xy3(:,2),'polynomial',p);
end
plotGraph(sigma_ply)

function plotGraph(sigma)
p = 1:10;
plot(p,sigma,'-ro',...
    'LineWidth',3,...
    'MarkerEdgeColor','k',...
    'MarkerFaceColor','r',...
    'MarkerSize',10)
grid on
set(gca,'FontSize',28)
str = 'Variance as a function of polynomial order';
set(gca,'Title',text('String',str,'FontAngle', 'italic', ...
    'FontWeight', 'bold'), ...
        'xlabel',text('String', 'polynomial order (p)',...
        'FontAngle','italic'),...
        'ylabel',text('String', 'variance','FontAngle','italic'), ...
        'FontSize',28)

end
```

Output: The written code plots variance versus polynomial order. It can be seen that the variance decreases drastically and after a point it remains constant. The plot can be seen in Figure 2.3.

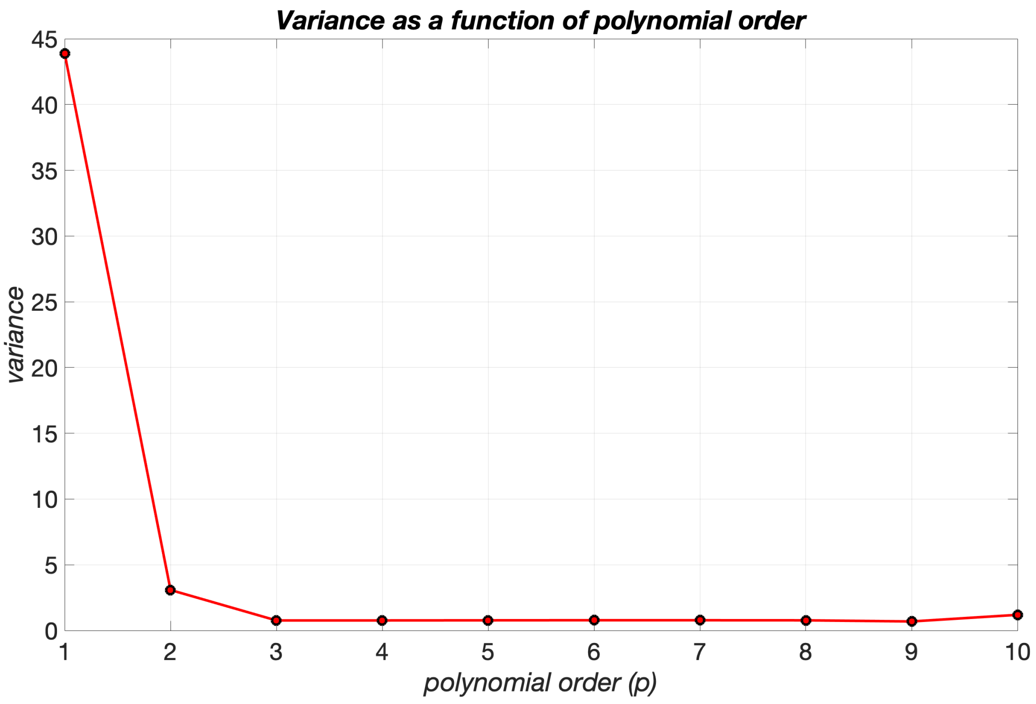
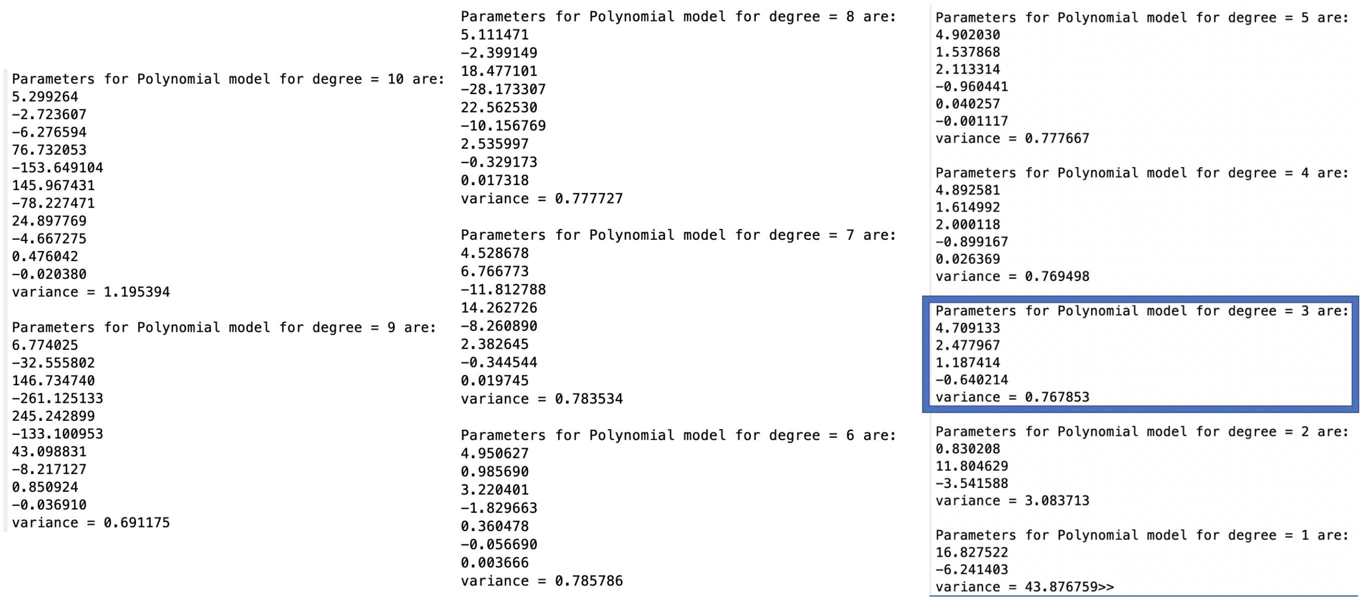


Figure 2.3: Variance as a function of the polynomial order

The output of the execution is shown below. The parameters for polynomial order 1 to 10 are printed and shown.



```
Parameters for Polynomial model for degree = 10 are:
5.299264
-2.723607
-6.276594
76.732053
-153.649104
145.967431
-78.227471
24.897769
-4.667275
0.476042
-0.020380
variance = 1.195394

Parameters for Polynomial model for degree = 9 are:
6.774025
-32.555802
146.734740
-261.125133
245.242899
-133.100953
43.098831
-8.217127
0.850924
-0.036910
variance = 0.691175
```

```
Parameters for Polynomial model for degree = 8 are:
5.111471
-2.399149
18.477101
-28.173307
22.562530
-10.156769
2.535997
-0.329173
0.017318
variance = 0.777727

Parameters for Polynomial model for degree = 7 are:
4.528678
6.766773
-11.812788
14.262726
-8.260890
2.382645
-0.344544
0.019745
variance = 0.783534

Parameters for Polynomial model for degree = 6 are:
4.950627
0.985690
3.220401
-1.829663
0.360478
-0.056690
0.003666
variance = 0.785786
```

```
Parameters for Polynomial model for degree = 5 are:
4.902030
1.537868
2.113314
-0.960441
0.040257
-0.001117
variance = 0.777667

Parameters for Polynomial model for degree = 4 are:
4.892581
1.614992
2.000118
-0.899167
0.026369
variance = 0.769498

Parameters for Polynomial model for degree = 3 are:
4.709133
2.477967
1.187414
-0.640214
variance = 0.767853

Parameters for Polynomial model for degree = 2 are:
0.830208
11.804629
-3.541588
variance = 3.083713

Parameters for Polynomial model for degree = 1 are:
16.827522
-6.241403
variance = 43.876759>>
```

Inference: The variance reduces from 43.87 at $p = 1$ to 0.767 at $p = 3$. Hence the polynomial of order 3 is considered to be the correct order. This is because, as the order increases further than 3, it remains almost constant.

2.5 Comparison of observations and estimated model

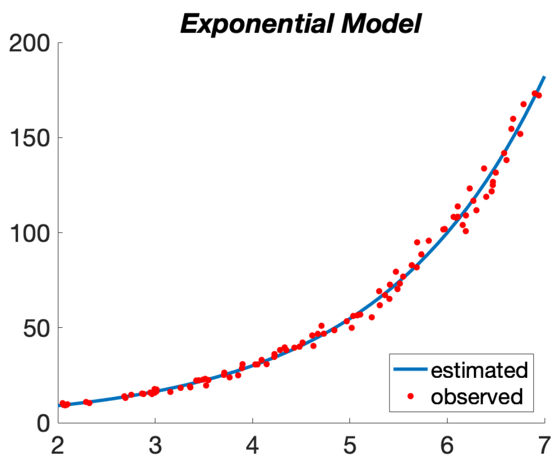
Task: For each of the models estimated above show the observations of the model $(x_i, y_i) : i = 1, 2, ..., N$ and the corresponding reconstructed model curve $(x_i, g(x_i)) : i = 1, 2, ..., N$ in one figure.

Solution: We created a function `compare` inside the function `plotGraph` which used the values of the parameters from the previous exercise i.e., when $p = 3$ for polynomial model and $p = 1$ for the rest of the models to compare with the observed values from the data set which was previously created.

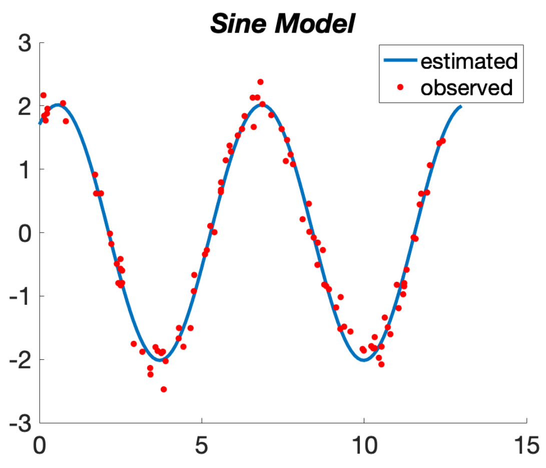
MATLAB code:

```
clear all
close all
clc

load dat3_1;
plotGraph('Exponential Model', xy1(:,1),xy1(:,2));
plotGraph('Sine Model', xy2(:,1),xy2(:,2));
plotGraph('Polynomial Model', xy3(:,1),xy3(:,2));
plotGraph('Power Model', xy4(:,1),xy4(:,2));

function plotGraph(model,observed_x,observed_y)
if strcmp(model,'Exponential Model') == 1
    expected_value = 2 : 0.1 : 7;
    expected_t = 2.714761*(1.823818.^expected_value);
    compare(expected_value, expected_t, observed_x, observed_y, model);
elseif strcmp(model,'Sine Model') == 1
    expected_value = 0 : 0.1 : 13;
    expected_t = 2.015541*sin(expected_value + 1.008430);
    compare(expected_value, expected_t, observed_x, observed_y, model);
elseif strcmp(model,'Polynomial Model') == 1
    expected_value = 0 : 0.1 : 5;
    expected_t = 4.709133 + 2.477967*expected_value  ...
        + 1.18741*expected_value.^2 - 0.640213766*expected_value.^3;
    compare(expected_value, expected_t, observed_x, observed_y, model);
elseif strcmp(model,'Power Model') == 1
    expected_value = 0 : 0.1 : 5;
    expected_t = 1.354748*(expected_value .^ 0.499158);
    compare(expected_value, expected_t, observed_x, observed_y, model);
end
    function compare(expected_x,expected_y,observed_x,observed_y,model)
        figure()
        hold on
        plot(expected_x, expected_y,'LineWidth',3)
        plot(observed_x,observed_y,'.r','MarkerSize',15)
        hold off
        set(gca,'FontSize',22)
        title(model,'FontAngle', 'italic', 'FontWeight', 'bold')
        legend('estimated','observed')
    end

end
```

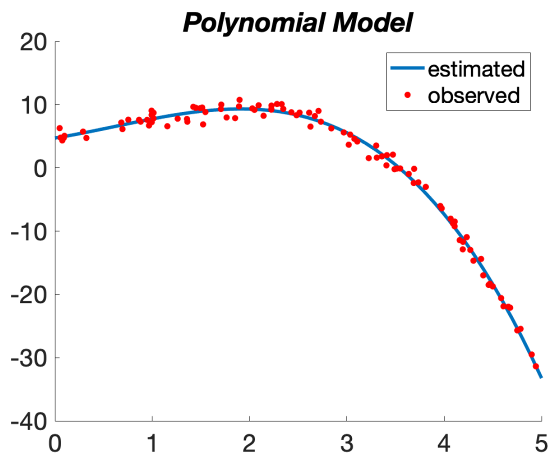
Output: The output of the code is shown below. The estimated value is shown by a line while the observed value is represented by dots.



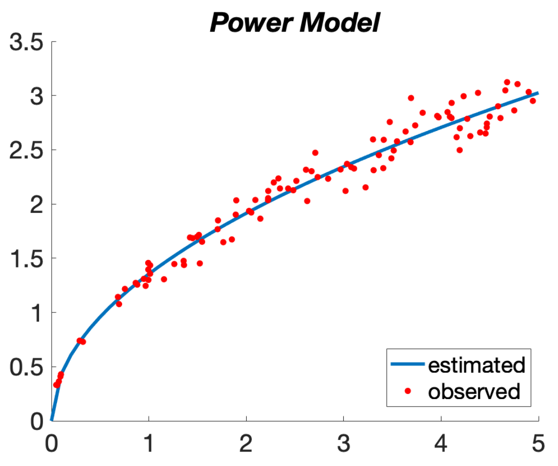
(a) Exponential model



(b) Sine model



(c) Polynomial model



(d) Power model

Figure 2.4: Comparison of observations and estimated models

Inference: We can infer that the observed value almost coincides with the estimated value.

2.6 LS-adjustment to a circle: estimating the centre

Task: Load `dat3_2` containing a 100×2 matrix that represent the coordinates of 100 measurement points in the xy-plane. Take a look at the measurement points and give from this the coordinates of the centre location.

Solution: `dat3_2` was loaded as mentioned in the task. It contains the matrix of data set xy. We estimate and find the centre of the circle to be located at (4,2).

MATLAB code:

```
clear all
close all
clc

load dat3_2;
figure()
hold on
plot(xy(:,1),xy(:,2),'.b','MarkerSize',28)
plot(4,2,'xr','MarkerSize',18)
hold off
set(gca,'FontSize',26)
title('LS - Adjustment to a circle: estimating the center', ...
    'FontAngle', 'italic', 'FontWeight', 'bold')
axis equal
```

Output: The output of the code is shown below.

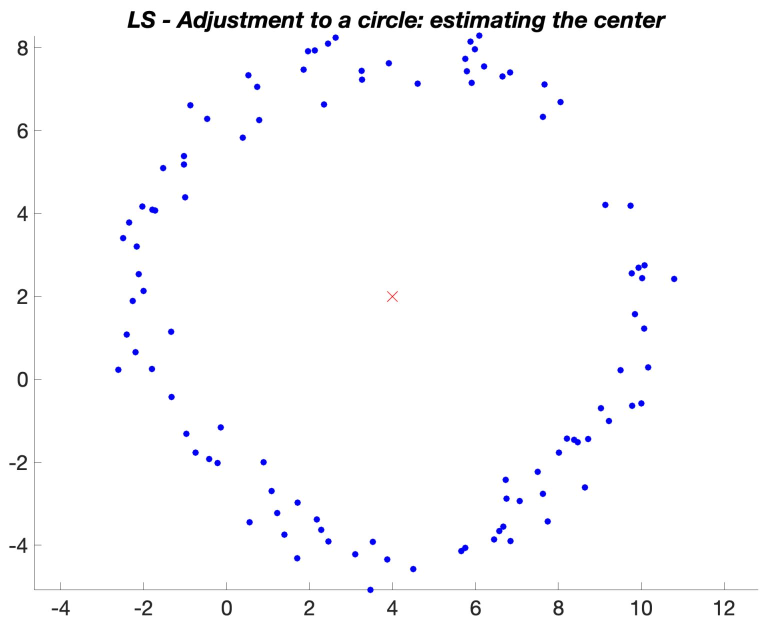


Figure 2.5: LS-adjustment to a circle: estimating the centre

Inference: The measurement points nearly takes the shape of a circle with centre at (4,2)

2.7 LS-Fit to a circle: iteration

Task: Carry out a LS-Estimation as shown in section 3.1.5. Use the estimated centre as an improved initial value and repeat the LS-Estimation as long as $||(\tilde{X}_0(k), \tilde{Y}_0(k)) - (\tilde{X}_0(k-1), \tilde{Y}_0(k-1))|| < 10^{-10}$ is fulfilled, where $(\tilde{X}_0(k), \tilde{Y}_0(k))$ denotes the k-th LS-Estimate of the centre. Write a function `LSE_circle` that carries out the iteration. Depict the reconstructed circle and the measurement points in a diagram.

Solution: We carry a linear LS - Approach to a circle which is mentioned in topic 1.5. We wrote a function `LSE_circle` which outputs the LS-estimated circle fitting when given the measured points as the input. Maximum iteration time is added to control the termination incase the code does not converge.

MATLAB function LSE:

```
function [est_x0, est_y0, r] = LSE_circle(x, y, mean_x, mean_y)
err = 1;
count_max = 100;
count = 0;
while err > 10^-10  && (count < count_max)
    count = count + 1;
    for i = length(x) : -1 : 1
        c(i) = sqrt((x(i)-mean_x)^2 + (y(i)-mean_y)^2);
        a(i) = (x(i)-mean_x)/c(i);
        b(i) = (y(i)-mean_y)/c(i);
    end
    H = [ones(length(x),1) a' b'];
    V = (inv(H'*H))*H'*c';
    r = V(1);
    u0 = V(2);
    v0 = V(3);
    est_x0 = u0 + mean_x;
    est_y0 = v0 + mean_y;
    err = max(u0,v0);
    mean_x = est_x0;
    mean_y = est_y0;
end
end
```

MATLAB code:

```
clear all
close all
clc

load dat3_2;
x = mean(xy(:,1));
```

```
y = mean(xy(:,2));
[x0,y0,r] = LSE_circle(xy(:,1),xy(:,2),x,y);
figure()
hold on
plot(xy(:,1),xy(:,2),'.b','MarkerSize',25)
plot(x0,y0,'xr','MarkerSize',15)
hold off
axis equal
hold on
rectangle('Position',[x0-r,y0-r,2*r,2*r],'Curvature',[1,1],'LineWidth',3);
hold off
set(gca,'Title',text('String','LS-Fit to a circle', ...
    'FontAngle', 'italic','FontWeight', 'bold'), ...
        'xlabel',text('String', 'x', 'FontAngle','italic'),...
        'ylabel',text('String', 'y','FontAngle','italic'), ...
        'FontSize',28)
fprintf('\nThe fitted circle has:')
fprintf('\ncentre at (%f,%f)',x0,y0)
fprintf('\nradius = %f',r)
```

Output: The output of the code is shown below.

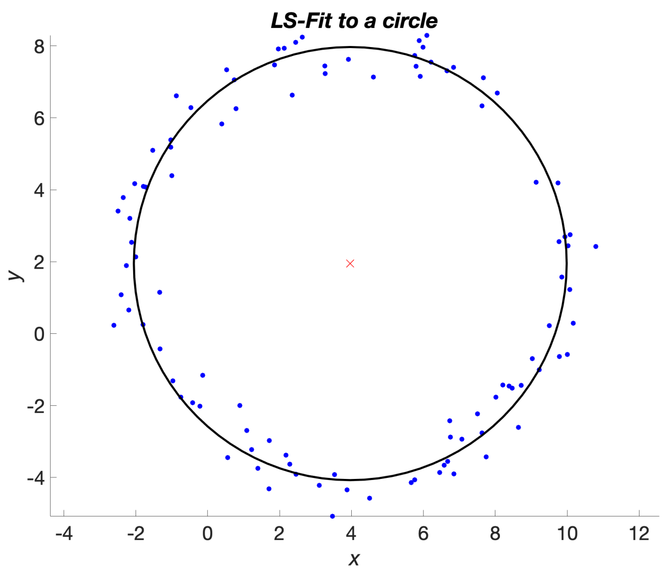


Figure 2.6: LS-Fit to a circle: iteration

The execution of the code gives us the following result.

```
The fitted circle has:
centre at (3.965483,1.947539)
radius = 6.020386>>
```

Inference: The measurement points nearly takes the shape of a circle with centre at (3.965,1.947) and radius of 6.020.

2.8 LLS-Fit to a circle: convergence

Task: Does this method converge, if the initial guess of the circle center is very poor? Try it by giving your function `LSE_circle` intentionally a very poor initial estimate of the centre location. Explain your observation.

Solution: The function `LSE_circle` mentioned earlier is used again in this program. In this method, we require an initial guess of the centre of the circle. If this guess is poor, the method does not converge and no result can be obtained. **MATLAB code:**

```
clear all
close all
clc

load dat3_2;
x = xy(:,1);
y = xy(:,2);
[estx0, esty0, r] = LSE_circle(x, y, mean(x), mean(y));
mx = -4 : 1 : 12;
my = -6 : 1 : 10;
a = 1;
```

```
b = 1;
X_NCONV = [];
Y_NCONV = [];
X_CONV = [];
Y_CONV = [];
for i = 16 : -1 : 1
    for j = 1 : 17
        MX(i) = mx(i);
        MY(j) = my(j);
        [x0,y0,~] = LSE_circle(x,y,MX(i),MY(j));
        u0 = abs(estx0-x0);
        v0 = abs(esty0-y0);
        r1 = sqrt(u0.^2 + v0.^2);
        if r1 > r
            X_NCONV(a) = MX(i);
            Y_NCONV(a) = MY(j);
            a = a + 1;
        else
            X_CONV(b) = MX(i);
            Y_CONV(b) = MY(j);
            b = b+1;
        end
    end
end
figure()
plot(X_NCONV,Y_NCONV,'.r','MarkerSize',15)
hold on;
plot(X_CONV,Y_CONV,'.b','MarkerSize',15)
set(gca,'Title',text('String','LS-Fit to a circle: convergence', ...
    'FontAngle', 'italic','FontWeight', 'bold'), ...
        'xlabel',text('String', 'x', 'FontAngle','italic'),...
        'ylabel',text('String', 'y','FontAngle','italic'), ...
        'FontSize',28)
legend('not converging','converging');
rectangle('Position',[estx0-r,esty0-r,2*r,2*r],'Curvature',[1,1], ...
    'LineWidth',3);
```

```
axis equal;
```

Output: The output of the code is shown below. The blue dots represent convergence while the red represent non convergence. Thus, taking the blue points as initial guess will lead to convergence and vice versa.

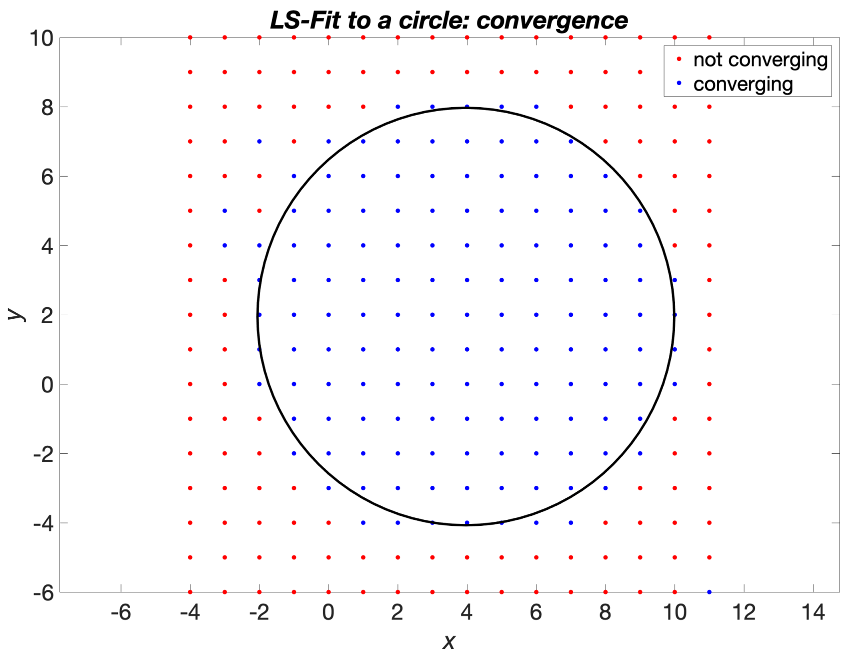


Figure 2.7: LS - adjustment to a circle: estimating the centre

Inference: Majority of the blue dots lie inside the circle leading to convergence. Few blue dots lie outside the circle which can also allow convergence. If the initial guess is placed where there is a red dot, the system will not converge and hence no result will be obtained.

Bibliography

[1] Prof. Dr.-Ing. Dieter Kraus, *"Stochastic Signals and Systems- Probability Theory* lecture notes.

[2] https://de.mathworks.com/help/matlab/

[3] Intuitive Probability and Random Processes using MATLAB, Steven M. Kay, Kluwer Academic Publishers, 2006