Project 1: Estimating functions from noisy data

SSY 230, Learning dynamical systems
using system identification

Jonas Sjöberg
Electrical Engineering, Chalmers

March 18, 2024

In this project you will develop a number of functions for estimating regression models. You will use the functions to explore properties of the algorithms.

You will also use of the functions in coming projects in the course where they will be building blocks for functions for estimating models of dynamic systems. For this to be possible, it is important that you make the function calls as indicated.

Matlab is the intended tool to use in the project.

Each function need to be validated so that you can trust it when you use it later on. It is often good to design a simple example where you by hand can calculate the output and compare with your function.

You work in groups of two students, and a report of the project should be handed in. In the report, you answer the questions which are indicated in this document and you explain your solutions. Try to write in a way so that it is possible to give feedback on your report. For example, if you validate a function and you write a motivation why you think it is correct, then we can give you feedback on if you have the logic right. This is in contrast to if you do not give any motivation, and we can not help you much.

You should also be prepared to demonstrate the functions.

In Canvas, you can find Matlab Grader problems which help you to develop and validate parts of tasks.

1 Linear regression functions

You should write estimation functions which returns a) structure with one field for the parameters, called `theta` and one field for the parameter uncertainties, called `variance`. That is, if the structure is `m` we could have the following object, if there are three parameters

```
>> m.theta=[-3, 4 2]
>> m.variance=matrix.
>> m.model='LR'
```

where the variance matrix is a symmetric matrix of dimension 3. The third field "model" indicates that it is a linear regression model.

(a) Write a function which computes the least squares estimate given two matrices `x` and `y`. `x` is a regressor matrix with one row for each sample and `y` contains the output values one row for each sample (there can, hence, be more than one output in which case `y` has several columns and `theta` is a matrix). Call the function `LinRegress`. The module **LinRegress** in Canvas gives you good help to develop the function. Start using it and when the function is ready, you copy it from the module to a directory on your computer.

Test the function on an example, you design, where you generate `y` with `theta` of your choice, and you check that your function returns the same `theta`. The variance is harder to validate, in Appendix A you get some hints and examples giving you a start so that you can validate your programming with some theoretical calculations.

Use the command "\ "to solve the least squares problem.

An expression for the variance can be found in the literature, eg, in S & S (4.12, 4.13) or R.J. (5.24). That expression is for the case you have one output signal, and it is enough that you have uncertainty for that case, ie, if there are several outputs, you can set the variance to `None`. Strictly, S & S (4.12, 4.13S & S (4.12, 4.13) hold when the number of data goes to infinity, see S& S Example 7.6, p 207.

Include your validation test that the estimator works as intended, in your report.

(b) Write a function `evalModel` which takes a model and a matrix of regressors (one on each row) as arguments and returns the output of the model, ie, a prediction of the output for these input regressors. It is a trivial function, but it will later be extended so it is important to implement it.

(c) To avoid overfitting, we will sometimes use *regularization*. Write a function `linRegressRegul` which takes inputs `x`, `y`, and λ and returns the same variable as `LinRegress`. The estimate should now minimize

$$\sum_{k=1}^{N}(y(k) - \theta \cdot x^T(k))^2 + \lambda\theta \cdot \theta^T.$$

This is easily obtained by appending `x` and `y` in the following way and then calling `LinRegress` with the appended matrices.

```
>> x2=[x;sqrt(lambda)*eye(size(x))];
>> y2=[y; zeros(size(y))];
```

Verify that this is correct (in equations, not in Matlab) and include this verification in your report. You do this by starting with the expression `LinRegress` minimizes, ie, $\sum(y_i - x_i\theta^T)^2$ and insert the extended matrices.

You can verify `linRegressRegul` by choosing λ zero and re-obtaining the true values, and choosing a very large number and obtaining a parameter vector close to zero.

(d) Polynomial fitting. For example, if you have two regressors, x_1 and x_2, for a degree 3 model the regressor should be $[1\ x_1\ x_2\ x_1^2\ x_1x_2\ x^2\ x_1^3\ x_1x_2^2\ x_1x_2^2\ x_2^3]$. Write a function `polyfit` which takes inputs `x`, `y`, λ and a positive integer n and returns the same variable as `LinRegress`. n indicates the degree of the polynomial. You need create a new regressor matrix `x2` depending on n. When that is done, you need just to call `linRegressRegul`. You should, however, include one more field in the returned structure, the degree of the polynomial, `m.n`. In Canvas you find a module giving some support for the function calculating `x2`, it is called `poly_x2`.

`x2` should contain one column of 1 giving a constant term in the model, then the original `x` should be included and all the powers up to `n`, also the mixed terms. The tricky part is to construct all the combinations (but each one only once) when $n > 1$. One way to do this, is to first generate the exponents for each monomial, and then use them to generate the monomials. For the example above, with x_1 and x_2, for a degree 3 there will be 10 monomials and the exponents for the two first ones are

$$\begin{matrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ \vdots & & \end{matrix}$$

For example giving

```
x=[2 3;4 5]; n=3;
```

Should return `x2 =`

```
1 2 3 4 6 9 8 12 18 27
1 4 5 16 20 25 64 80 100 125
```

To validate your function, generate some data with a quadratic and cubic function, and check that you re-obtain the correct parameters. **Include your validation in the report.**

(e) You should now write a function `plotModel` which plots your model output together with the estimation data for the cases that the function has one or two inputs. In the one dimensionalcase, plots your data together with the plot also the estimated uncertainty, eg the 95% percentil.

The uncertainty can be illustrated as a shaded region. How to calculate this shaded area as a function of the uncertainty of the parameters is not obvious. There are several ways it can be done in approximate ways. One way to do this is described in Appendix B.

Make sure that you use the function `evalModel` when you calculate the model output. In that way, your plot function will automatically work for other model types.

Write also a version of the function where several models can be submitted and plotted together. In this case, do not include the uncertainty since that would make the plot messy.

The file `Test_plotModel` contains a script you can use to validate your results. Appendix C describes how it should be used.

2 KNN-regression functions

A K-nearest neighbor estimator is trivial. Nevertheless, we will write a couple of functions to handle this estimator so that it becomes more convenient to work with it.

(a) In fact, there are no computations to be done in a KNN-estimator until when you want to compute the estimate of the output for a new input regressor. Therefor, the KNN "estimator" should return a structure which contains all necessary information. Write a function `knnRegress` which takes `x` and `y` and a positive integer k as inputs. The function should return a structure with the following fields: `x`, `y`, `k` and `model='knn'`, where the three first just store the input argument and the last one indicate that it is a KNN model.

(b) Now, modify your function `evalModel` so that it works for both linear regression and KNN models. The part for linear regression you already have. The part for KNN should work the following way:

- Given an input regressor $\tilde{x}$ find the n closest regressor in the KNN model.
- Form the mean of the outputs associated with the n closest regressor in the KNN model and deliver this mean value as the model estimate.

Make sure that the function works also for the case that you submit a matrix of input regressors.

(c) If necessary, modify your function `plotModel` so that it works also for the KNN-regressor models. Note that there is no uncertainty to display for these models.

The module **knnRegress** in Canvas gives help to test your nearest neighbor model. When the function works in Matlab grader, copy the code and test it on your computer, in that way you also get your `evalModel` tested.

Test the function on the same example and include the result in the report. Make an example where you plot a linear regression model and a KNN-model in the same plot.

3 Estimating one dimensional functions

You will now use the functions you developed in previous section. On a couple of examples you will investigate properties of least squares estimation. Aspects you will investigate are, the number of available data, the parameter uncertainty, model quality, the influence of noise on the estimate, and how the flexibility of the model influence the result.

To estimate the quality of the models, you should generate a second, noise-freed data set. On that data set you can compute the mean square error (MSE) as a measure of the model quality.

You should also use the plot function you created to illustrate your results.

1 Use the Matlab function `linearData(N, var,0)` to generate data.

(a) Set the noise variance to 1 (constant) and generate data sets of size 10, 100, 1000 and 1000. Estimate linear regression models (constant + linear term) and evaluate the result. Does the estimate converge to the true function when the number of data goes to infinity? **In the report, indicate a reference to the question (3.1.a etc) where you write the answere and suport the answere with figures illustrating what you write. Do this for all questions coming.**

(b) Repeat the experiment above but with a 5th order polynomial. Compare the result with the one above. What is the difference? You can illustrate the result in a plot with the model quality versus number of data.

(c) Validate the expression for the parameter variance with a Monte Carlo simulation. You can do this, eg, by estimating linear models on different data sets with, eg, 10 data. Then you make a histogram by the parameter estimates. The file `MonteCarloExample.m` illustrate how this can be done. Note the difference of this validation from the one you did in Problem 1.a. There you validated that you program gave the same variance as the theoretical expression. Here you validate that the estimate you obtain has the variance you validated earlier.

(d) Try KNN models on some of the settings above. How many neighbors is good depending on the noise level and the number of data? Don't spend a lot on time on this, a very approximate answer is enough indicating how thebest number of neighbors change when number of data change.

2 Use the Matlab function `polyData(N, var)` to generate data.

(a) Repeat the the experiment in a) in the previous exercise. Does the parameter uncertainty go to zero for the linear model? Does the model converge to the correct function? Explain how the parameter uncertainty can go to wards zero although the model cannot explain the data.

(b) Try polynomial models of various degree on the data. Can you obtain a good result? Try a high degree polynomial, eg, 10th order, on a small data set, eg, 15 data. How does the result look? include regularization, try 0.01, 0.1, ... 100 and compare the result. Note, it is not uncomon that the solver get numerical problems for degree above, about 12. Matlab give you a warning and the result can be very strange. We are not interested of thet issue here.

(c) Generate an unsymmetrical data set by including a third argument set to '1' `polyData(N, var,1)` and estimate a linear model to this data. Why is it different from the linear model you had in a)? Could this be a problem? If yes, what could you do to solve it? There are many ways so try to suggest one or to possible ways.

(d) Try KNN models on some of the settings above. How many neighbors is good depending on the noise level and the number of data? Again, don't spend too much time, appriximate result is enough indicating the general dependence.

3 Use the Matlab function `chirpData(N, var)` to generate data.

(a) Does the "best" model depend on the number of data? With a given noise level, eg 0.2, generate 50 data and search for a polynomial model (try different degrees) which

gives the best model. Now, generate 1000 data, can you obtain a better model given this larger data set? Explain the result. Test also data with less noise, can you obtain a better high order model if the data has less noise?

(b) Is it better to have a high-degree polynomial + some regularization than having a lower degree polynomial without regularization?

(c) Try KNN models on this data set. Are they better then the polynomial models? Give the MSE on validation data for the different models.

4 Estimating two and high dimensional functions

One-dimensional function estimation problems have the advantage that you can easily visually inspect the result, and often get a feeling for if your estimate is under, or over-fitting the data. That is, if you model is too rigid or too flexible. Also, general function estimation is harder when the number of dimensions is higher, that is called the curse of dimensionality.

Here yo will test to estimate functions in two dimensions, where you still have possibilities to plot the result, and then in 10 dimensions.

For the two-dimensional problems you can visualize the true function by generating, eg, 1000 noise-free data and then with the following commands.

```
>> [xq,yq] = meshgrid(0:.2:10, 0:.2:10);
>> vq = griddata(x(:,1),x(:,2),y,xq,yq);
>> mesh(xq,yq,vq);
```

If you want to see the data in the same plot you give the following additional commands.

```
>> hold on
>> plot3(x(:,1),x(:,2),y,'o');
```

Explain how you measure the quality of the models you estimate in the following problems.

Note that the outcome is stochastic due to the data which is stochatically generated. If you write your commands in an m-file you can easily repeat them to see to what extent the results vary and, hence, depends on the particular realization of the data.

1 In this exercise you can choose one of the following two matlab functions to generate data, `twoDimData1(N, var)` or `twoDimData2(N, var)`. Before you start, find another group who choose the other data set, and compare your results with each other.

(a) Generate first noise-free data and look at the function as described above.

(b) Generate 100 noisy data with variance 1. Generate also a validation (also called test) data set with, for example, 1000 samples.

(c) Test a linear regression model.

(d) Test polynomial models of degree 2 to 5.

(e) Test KNN models, which number of neighbors gives best result? Is the KNN better than the polynomial model? Depending if it is or not, lower or increase the number of estimation data to see (approximately) the condition when this change.

3 (Optional) Use the Matlab function `tenDimData(N, var)` to generate data.

(a) How many regressors will you have for a linear regression model using this data set? How many regressors will yo have if you make a linear regression model containing polynomial regressors up to degree 3?

(b) Generate 1000 noisy data with variance 1. Generate also a validation data set with, for example, 10000 samples.

(c) Test a linear regression model.

(d) Test polynomial model of degree 3. Verify that the number of regressors equals what you calculated above.

(e) If you use regularization, can you improve the quality of the polynomial model?

(f) Test KNN models, which number of neighbors gives best result? Is the KNN better than the polynomial model? Depending if it is or not, lower or increase the number of estimation data to see (approximately) the condition when this change.

Appendix A: Simple test of uncertainty of estimated parameters.

Assume the data is generated by

$$y_i = x_i \cdot \theta^T + e_i$$

x is the regressor vector, y is the function output and θ is an unknown parameter vector which you estimate by Least Squares algorithm. This gives you an estimate $\hat{\theta}_N$ based on a set of N estimation data $\{y_i, x_i\}_{i=1}^N$. In the exercise you should implement an estimate of the variance of the estimate $\hat{\theta}_N$. Here is a very simple example how you can validate your estimate. It is not a total validation test but if your function does not pass this test, then there is something with it.

To make things simple, let dim $x = 1$, $x_i = 1$, $\theta = 0$ and $e_i \in N(0, 1)$. Then the estimate of θ becomes

$$\hat{\theta}_N = \frac{1}{N} \sum_{i=1}^{N} y_i$$

You can now formulate the expresion for the variance of $\hat{\theta}_N$ as a function of N as

$$\mathrm{E}[\hat{\theta}_N \hat{\theta}_N^T] = \mathrm{E}[\frac{1}{N} \sum_{i=1}^{N} y_i \frac{1}{N} \sum_{j=1}^{N} y_j]$$

Since $\theta = 0$, y_i can be replaced by e_i, and since you know the variance of e_i you can calculate the variance of $\hat{\theta}_N$. **Do that and include your calculations in the report.**

Compare your calculation with the estimate from your Matlab function on a test example for some values of N to verify that you get the same value.

This was a very trivail validation for the special case that there is only one parameter. It is not so difficult to extend it to the case with two parameters. However, x cannot be chosen constant, so it becomes a bit more elaborated, but not much. You can, eg, choose $x_i = [1,\ 1 + (-1)^i]$, that is, the first component is constant and the second varies. θ can be chosen to [0 0], although non-zero parameters does not make the equations much more complicated. Now, the estimate of θ becomes

$$\hat{\theta}_N = (\frac{1}{N}\sum_{i=1}^{N} x_i^T x_i)^{-1} \frac{1}{N}\sum_{i=1}^{N} x_i y_i$$

Since $\theta = [0\ 0]$, y_i can be replaced by e_i, and since you know the variance of e_i you can calculate the variance of $\hat{\theta}_N$, as above but you will now have an expresion involving matrices. Do that and include your calculations in the report.

Appendix B: Function depending on uncertain parameters

This appendix describes how you can use the estimated parameter uncertainty to describe the uncertainty of functions depending on the parameters. This is useful in project 1 and 2.

We want to describe the variance of $f(x, \theta)$ as a function of the variance of θ. If f is a linear function, this can be done analytically. But if f is nonlinear, it can only be done approximately. Here we show how it can be done using a numerical sampling method, we generate samples of θ according to its distribution, and obtain a set of functions f, one for each θ. Then calculate the variance of $f(x, \cdot)$ using the set of θ values.

Given N data, the uncertainty of the obtained estimate $\hat{\theta}_N$ is described by the estimate of its variance P_θ. How P_θ is estimated is given by, eg, in S & S (4.12, 4,13) or R.J. (5.24).

The parameter estimate can the be considered as a realization from the following multi-dimensional normal distribution

$$\frac{1}{\sqrt{(2\pi)^2 \det P_\theta}} \exp(-\frac{1}{2}(\hat{\theta}_N - \theta_0) P_\theta^{-1} (\hat{\theta}_N - \theta_0)^T) \tag{1}$$

where θ_0 is the true, unknown parameter value. From this follows that

$$(\hat{\theta}_N - \theta_0)^T P_\theta^{-1} (\hat{\theta}_N - \theta_0)) \in \chi^2(d) \tag{2}$$

where

$$d = \dim \theta$$

Strictly, the distributions given above are only correct in the case that N goes to infinity, but the result hold approximately for finite, large, N. Therefor we say that we have an *asymptotic* normal distribution, and an *asymptotic* χ^2 distribution.

The total volume of the distribution function (1) is 1, ie,

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \frac{1}{\sqrt{(2\pi)^2 \det P_\theta}} \exp(-\frac{1}{2}(\hat{\theta}_N - \theta_0)^T P_\theta^{-1}(\hat{\theta}_N - \theta_0)) d\theta_1 \dots d\theta_d = 1 \tag{3}$$

This integral is over the entire θ space. A confidence region, eg, 95% confidence region, is an ellipsoid so that the integral over the ellipsoid becomes 0.95. We are interested to generate number of examples, say n examples, on the boarder of this ellipsoid as examples of the uncertainty of the parameter estimate. It is, in fact, very easy to generate such examples, and that can be done by following this algorithm.

1. For each $k = 1$ to n.
2. Generate a random vector $\Delta\theta_k$ of dimension d and scale it so that its length becomes $\| \Delta\theta_k \|^2 = \chi^2_\alpha(d)$, where α is the confidence level you want to illustrate. You can obtain $\chi^2_\alpha(d)$ with the command `chi2inv`.
3. Generate the kth example vector as

   $$\theta_k = \hat{\theta} + P^{1/2}\Delta\theta_k$$

   $P^{1/2}$ can be obtained with `sqrtm`.

Now, assume you have a function $f(x, \theta)$ and you have estimate $\hat{\theta}_N$ and $\{\theta_k\}_{k=1}^n$ obtained as described above. The function $f(x, \theta)$ can be an estimate of a static function, as in project 1, or a model of a dynamic system which depends on the parameters in a complicated way, as in a prediction in project 2. In any case, a function describing the upper limit of the uncertainty can be formulated as

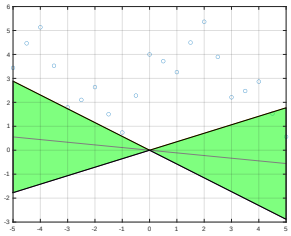
$$f_{\max}(x, \hat{\theta}) = \max_k f(x, \theta_k) \tag{4}$$

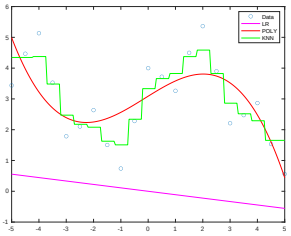
A function for the lower limit, $f_{\min}$, can be defined similarly.

How good are these estimated of upper and lower limits? Well, they depend on the quality of the estimate of P_θ, N, and also how the data were generated, ie, how close to the assumptions they were generated. The quality also depends on how many random examples, n, you generated. You can try, eg, 5, 10 and 20 to see how much that change the result.

Appendix C: Test plotModel

The file `Test_plotModel` is intended to be a help for validating your some of your functions. If everything is correct, when you run the command in Matlab you should obtain three figures identical to the following three.




Data
LR
POLY
KNN