Project 1: Estimating functions from noisy data

SSY 230, System Identification
Jonas Sjöberg
Electrical Engineering, Chalmers

March 25, 2019

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.

You work in groups of two students, and a report of the project should be handed in. You should also be prepared to demonstrate the functions.

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 `theta` is a matrix). Call the function `LinRegress` and test it on an example 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. If you make a small example, you can validate by performing the calculation by hand.

Used 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`.

Include your test 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. Verify that this is correct (in equations, not in Matlab) and include this verification in your report.

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

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

(d) Polynomial fitting. Write a function `polyfit` which takes inputs `x`, `y`, λ and an 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`.

`x2` should contain one column of 1 giving a constant term in the model, then the original `x` should be included and all th powers up to `n`. The tricky part is to construct all the combinations (but each one only once) when $n > 1$.

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

(d) You should now write a function `plotModel` which, in case you are estimating a one dimensional function, plots your data together with the estimated function and the uncertainty.

The uncertainty can be illustrated as a shaded region. How to calculate this shaded area is not obvious. There are several ways it can be done in approximate ways.One way to do this is described in the document *ModelVariance* which you find on the course web.

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.

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 n as inputs. The function should return a structure with the following fields: `x`, `y`, `n` 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.

Test the function on 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)` 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?

(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.

(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?

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.

(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?

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.