Model-building and parameter estimation

Introduction

The goal of this exercise is to get more familiar with some of the tools for model-building and parameter estimation.

The task is to implement a simulation model that generates data (simulating real-world measurements), and then fit two types of models to these data. One of the models is a simple polynomial model, and the other is the *correct* model, but with an unknown parameter.

The exercise is to be performed in MATLAB as per instructions.

1 Preparation

Read the handouts on model-building and parameter estimation. If necessary, refresh your MATLAB programming skills by solving some of the problems in the text book.

Read this entire document thoroughly. There are some MATLAB hints at the end. Make sure you understand the questions before you start coding. Also, try to plan your program before you start.

2 System description

The goal of this exercise is to implement and evaluate the different steps of the system described in Fig. 1.

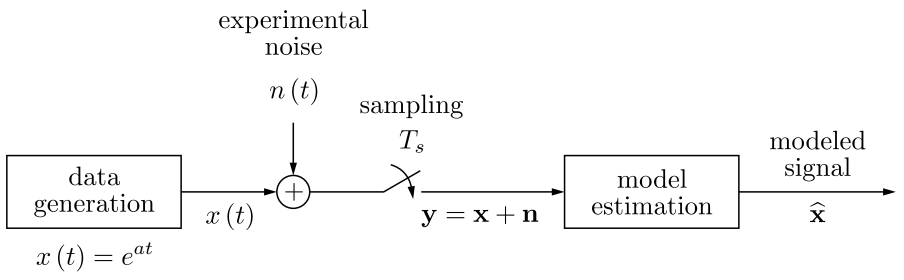

Figure 1: The system to be modeled.

The system consists of an input signal $x(t)$, known to be generated as an exponential function of time, t, but where the parameter a is unknown. This signal is then corrupted with experimental noise and then sampled using a sampling time T_s. The resulting sequence is recorded in the vector $\mathbf{y}$.

The task will be to implement both the data generation part and the model estimation block, and then to compare the performance of two different models.

3 Simulation setup

3.1 Model

We are going to simulate the case where data originate from a system which can be assumed to have the following underlying physical model:

$$x(t) = e^{at}, \tag{1}$$

where t is time (in seconds) and a is an unknown parameter. For this exercise we also assume that $0 < t < 10$.

Any measurement will also contain noise, here assumed to be additive white Gaussian noise with zero mean and variance σ^2. This means that the measured data (or signal) $y(t)$ will be

$$y(t) = x(t) + n(t), \tag{2}$$

where $n(t) \sim N(0, \sigma^2)$ and $x(t)$ is given by Eq. (1).

Throughout this work we also assume that the signals have been sampled with a sampling time T_s, so that

$$t = 0, T_s, 2T_s, \ldots, (N-1)T_s. \tag{3}$$

In MATLAB this means that all signals are represented by vectors, so that

$$\mathbf{x} = \begin{bmatrix} x(0) & x(T_s) & x(2T_s) & \cdots & x((N-1)T_s) \end{bmatrix}^T \tag{4}$$

is a column vectors of size $N \times 1$.

3.2 Problem

- Write a MATLAB function, that given the sampling time T_s and the noise standard deviation σ returns $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{t}$. Set the parameter a in the model to $a = 0.5$.
  Hint: In MATLAB, `n = s*randn(size(x));` generates a noise sequence with standard deviation `s`, with the same number of elements as the vector/matrix `x`. The command `t=0:Ts:10` generates the time vector, given the sampling time `Ts`.
- Modify the function so that it returns K realizations of $\mathbf{y}$, in a matrix $\mathbf{Y}$, where each column corresponds to one realization of $\mathbf{y}$.
  Note: In all realizations, $\mathbf{x}$ is the same, but the noise vector is different.
- Plot the noise-free signal $\mathbf{x}$ and one realization of the noisy signal $\mathbf{y}$ in the same plot, for a noise standard deviation $\sigma = 10$ and a sampling time $T_s = 0.05$ s, for $0 < t < 10$. This should look something like Fig. 2.

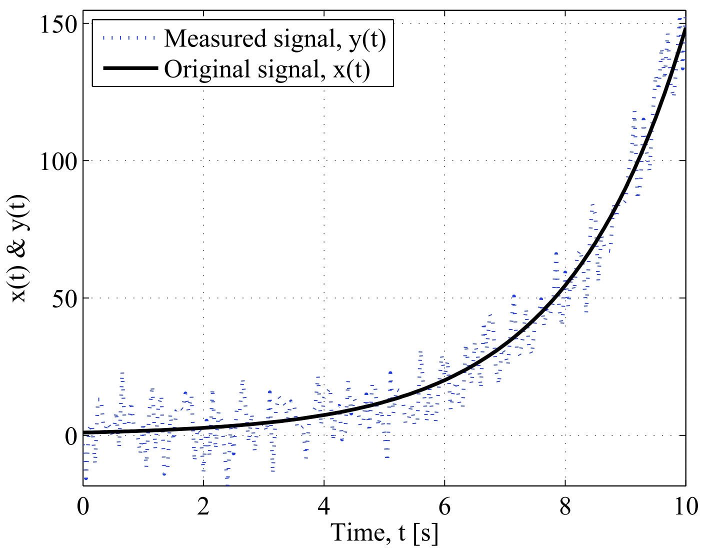

Figure 2: The noise-free signal together with an example of the noisy signal, with $\sigma = 10$.

4 Model-building

We are now going to fit two different models to the data generated in Section 3. The first one is a simple polynomial model, which has the advantage that it is linear in the unknown parameters and hence the estimator has a closed-form solution. The other model is the true exponential model, as given by Eq. (1). This model has only one unknown parameter, but we need to use some iterative optimization technique to find it.

4.1 Polynomial model

4.1.1 Model description

We are now assuming that $x(t)$ can be approximated as

$$\widehat{x}(t) = b_0 + b_1 t + b_2 t^2 + \ldots + b_p t^p, \tag{5}$$

i.e. a polynomial of order p. As explained in [1], this can be expressed in matrix notation as

$$\mathbf{y} = \mathbf{A}\mathbf{b}, \tag{6}$$

where $\mathbf{y}$ is an $N \times 1$ vector containing the measured data, $\mathbf{A}$ is an $N \times (p+1)$ matrix, and $\mathbf{b}$ is the $(p+1) \times 1$ vector with the unknown polynomial coefficients.

The least-squares estimate of $\mathbf{b}$ is now given by

$$\widehat{\mathbf{b}}_{LS} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y}, \tag{7}$$

And the modeled x, let's call it $\widehat{\mathbf{x}}$ is given by:

$$\widehat{\mathbf{x}} = \mathbf{A}\widehat{\mathbf{b}}_{LS}. \tag{8}$$

4.1.2 Problem

- Write a MATLAB function that, given the vector $\mathbf{t}$ and the polynomial model order p, creates the matrix $\mathbf{A}$.
- Write a MATLAB function that, given K realizations of the measured sequence $\mathbf{y}$, returns the K estimates, $\widehat{\mathbf{x}}$, as columns of a matrix $\widehat{\mathbf{Y}}$ (similar to the assignment in Section 3.2). You are NOT allowed to use the MATLAB functions `polyfit` or `polyval` for these assignments, although you could use them to validate your results.
- Plot some examples of the estimated polynomial together with the true signal $x(t)$, for polynomial model orders $p = 3, 4$, and 5. This could look like in Fig. 3.

4.2 Exponential model

4.2.1 Problem description

Now you will implement the Gauss-Newton method for solving the non-linear least-squares problem of finding a in Eq. (1), i.e. an iterative approach so that

$$\widehat{a}_{j+1} = \widehat{a}_j + \left(\mathbf{H}^T(\widehat{a}_j)\mathbf{H}(\widehat{a}_j)\right)^{-1}\mathbf{H}^T(\widehat{a}_j)\left(\mathbf{y} - \mathbf{x}(\widehat{a}_j)\right), \tag{9}$$

where j is the iteration number, $\widehat{a}_j$ is the estimate of the unknown model parameter a for iteration j. $\mathbf{H}(\widehat{a}_k)$ is the model gradient, and $\mathbf{x}(\widehat{a}_j)$ is the model, evaluated for the current value of $\widehat{a}_j$.

The gradient is given by

$$\frac{\partial \mathbf{x}}{\partial a}, \tag{10}$$

i.e. the partial derivative of the model with respect to the model parameter.

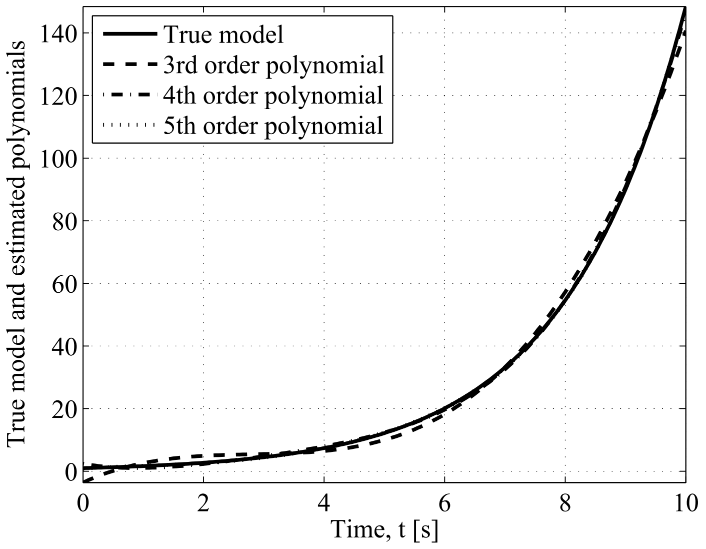

Figure 3: True model and polynomials of orders 3, 4, and 5.

4.2.2 Problem

- Derive an expression for the gradient.
- Write a MATLAB function that, given the time vector **t** and a value of a, returns a vector containing the gradient.
- Implement the Gauss-Newton algorithm as a MATLAB function that returns the estimated value of a and the resulting model $y(t)$.

Here is a pseudo-code example of the algorithm. All you need to do is to fill in the blanks and write it as proper MATLAB code. See [1] for more details.

```
j = 0;            % Iteration number
a = 0.2;          % Starting guess of a
update = inf;     % Initial change of the parameter
thresh = 1e-10;   % Stop criterion

while (j < max no. of iter.) and (update > thresh),
   H = model_gradient(a)
   da = inv(H'*H)*H'*(y-signal_model(t,a)
   update = a;
   a = a + da;
   update = update - a;    % Check how much a has changed
   j = j + 1;
end
```

5 Evaluation

5.1 Problem formulation

Now it is time to summarize and compare the two models. We know that the polynomial model is wrong, although the results seem quite good. The advantage of this model is primarily its simplicity in terms of parameter estimation. It is also fairly easy to evaluate its performance analytically . However, this is beyond the scope of this course, so we will now do some simulation-based evaluation. The other model, which in this case happens to be the correct one, is non-linear with respect to the unknown parameter, a. Because of this we used an iterative method to find the parameter. The questions we should consider now are:

1. Is the polynomial model just as good? If not, how?
2. How does the accuracy (or bias) of the two models compare?
3. How does the uncertainty (standard deviation) of the two models compare?

We will consider these questions in relation to:

1. Noise variance.
2. Sampling time (i.e. the number samples available for fitting the models).
3. Computational complexity.

5.2 Problem

- Write a MATLAB script that, based on $K = 300$ realizations, returns the matrices $\widehat{\mathbf{Y}}_{poly}$ and $\widehat{\mathbf{Y}}_{exp}$, containing the estimates using the polynomial and exponential models, respectively.
- Evaluate the repeatability (uncertainty) of the two models. Do it by generating and discussing the following plots:
  1. The average of the K estimated polynomial models and the $\pm 2\sigma$ interval, for a sampling time $T_s = 0.01$ s.
  2. Same as above, but for the exponential models.
  3. Same as the two above, but now using a sampling time of $T_s = 0.05$ s. Describe what happens?

  Note: The `mean` and `std` functions in MATLAB work on columns, so you need to transpose the matrices before applying these operations.
- Evaluate the model mismatch of the two models, by generating the following plots:
  1. The average error as function of time, i.e.

$$\bar{\mathbf{e}} = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{x} - \widehat{\mathbf{y}}_k) \tag{11}$$

  Note: Plot the average error of both models in the same plot.

2. All model errors as function of time. Make one plot for the polynomial model and one for the exponential model. Does the model mismatch appear to be approximately the same over the whole time interval? If not, discuss what you see and what you think causes this.
3. Repeat the two above, but now look at what happens if the polynomial order $p = 3, 4$, or 5.

6 MATLAB hints

Below is a list of MATLAB functions that might be of use in solving the problems. Use the `help` command for information on how to use them.

- `mean` and `std`, computes mean and standard deviation of the elements in a vector (or matrix).
- `inv`, computes the inverse of a matrix.
- `exp`, the exponential function.
- `ones` and `zeros`, creates matrices consisting of either ones or zeros.
- `repmat`, creates a matrix consisting of replicas of an existing vector/matrix.
- `randn`, returns random numbers from the Normal/Gaussian distribution.
- .* or .^ , i.e. the *dot* notation, which means element-wise multiplication or power (in this example).
- `tic` and `toc`, measures the time it takes to execute code.

7 Reporting

Same principles as for previous exercises. Attach your MATLAB code as appendices of the report.

References

[1] J. E. Carlson, "Introduction to empirical model-building and parameter estimation," Luleå University of Technology, Luleå, Sweden, Tech. Report, Sept. 2009.

[2] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice Hall, 1993, vol. 1.