Mathematical Modelling and Simulation of Technological Systems I

T. Arildsen, T. L. Jensen, T. Larsen

January 10th, 2014

Abstract

Modelling is one of the major disciplines in all of engineering and science. A mathematical model is some kind of lower-complexity description than the corresponding physical data describing the behaviour.

Contents

Chapter 1

Introductory example – the greenhouse effect

This chapter presents a small simple example of a mathematical model and gives an example of how its parameters may be determined and the model applied to some problem.

1.1 Problem

As likely known to everyone alive there is, and has been, a heated debate as to the rise of the global temperature and the importance of e.g. burning of fossil fuel on this. Table 1.1 shows measured results for the increase in temperature measured in degrees Celsius of a given year compared to the temperature in 1860 [1]. Thus, Table 1.1 shows:

$$t(y) \;=\; T(y) \;-\; T(1860), \quad y \in \mathbb{Y} \tag{1.1}$$

where $T(y)$ is an absolute measured temperature (although in °C) of year $y \in \mathbb{Y} = \{1880, 1896, 1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980\}$ (allowed y appear from Table 1.1) and $T(1860)$ is the absolute measured temperature in the year 1860. According to reference [1] most of the UK will be below sea level in case $t(y) = 7\,°\mathrm{C}$. The problem is then whether we can predict, based on the provided data in Table 1.1, if and when a flooding of the UK can happen – provided the conditions remain unchanged.

1.2 Possible solution

First of all it is important to emphasise that matters as complicated as global climate can hardly be boiled down to 11 measurements as in Table 1.1. However, as long as we handle this as an example and do not use it for more than it is, we may gain some knowledge from the data.

y	$t(y)$	$\log_{10}(t(y))$
$[-]$	[°C]	$[\log_{10}(°\mathrm{C})]$
1880	0.01	-2.00000
1896	0.02	-1.69897
1900	0.03	-1.52288
1910	0.04	-1.39794
1920	0.06	-1.22185
1930	0.08	-1.09691
1940	0.10	-1.00000
1950	0.13	-0.88606
1960	0.18	-0.74473
1970	0.24	-0.61979
1980	0.32	-0.49485

Table 1.1: Measured rise in temperature compared to 1860, $t(y)$, versus year, y. Although obviously not an ideal source of information, Wikipedia has some references for Earth temperature that clearly demonstrate that this area is far from trivial – see `http://en.wikipedia.org/wiki/Instrumental_temperature_record`. Data from [1].

It is notoriously difficult for us humans to gain insight from tabular data. Therefore, we start by plotting the data such that we have a linear plot of $t(y)$ versus y. This is shown in Figure 1.1. As seen from Figure 1.1, the temperature rise $t(y)$ is far from linear versus year y.

Often when observing data that seems to have some polynomial or exponential behaviour, we make a (semi-)logarithmic plot of the ordinate axis values (typically referred to as the y-axis). This is shown in Figure 1.2 where we clearly see something behaving closer to linear than in Figure 1.1. When observing the data in Figure 1.2 it seems like the behaviour may be something like:

$$\log_{10} t(y) \;=\; k\,y \;+\; b, \quad y > 1860,\; k, b \in \mathbb{R} \tag{1.2}$$

where k and b are constants, and the model is restricted to be used only for years after measurements were available – i.e. $y > 1860$. Note here that since (1.2) is a model, we are not (necessarily) limited to $y \in \mathbb{Y}$ as for the measured data. The simplest way to determine the model parameters k and b is to form two equations with two unknowns as:

$$\log_{10} t(y_0) \;=\; k\,y_0 \;+\; b \tag{1.3}$$

$$\log_{10} t(y_1) \;=\; k\,y_1 \;+\; b \tag{1.4}$$

Here we use the measured data from Table 1.1 such that we have a pair of year and measurement as $(y, t(y))$ for a number of years for which we have measured data available. Thus by simple manipulations of (1.3) and (1.4) we can compute k and b as:

$$k = \frac{1}{y_1 - y_0} \log_{10} \frac{t(y_1)}{t(y_0)} \tag{1.5}$$

$$b = \frac{y_1 \log_{10} t(y_0) \; - \; y_0 \log_{10} t(y_1)}{y_1 \; - \; y_0} \tag{1.6}$$

If we wish to know in what year y_p a certain temperature t_p is met, we simply rearrange (1.2) to yield:

$$y_\mathrm{p} \; = \; \frac{1}{k} \{\log_{10} t_\mathrm{p} \; - \; b\} \tag{1.7}$$

From the above we can make a formalised algorithm on how to compute the model parameters. This is shown in Algorithm 1.1. We define two arrays $\mathbf{y}$ and $\mathbf{t}$ to hold the 11 measurement points of year with respective temperature compared to 1860 level. To extract the model parameters according to (1.5)–(1.6), we choose two year pointers $p_0, p_1 \in \{0, 1, \ldots, 10\}$ where $p_0 \neq p_1$ such that $y_0 \; = \; \mathbf{y}[p_0]$ and $y_1 \; = \; \mathbf{y}[p_1]$. Returning to the formulated problem,

Algorithm 1.1 Algorithm for extraction of model parameters k and b.

```
1: INPUT: $\mathbf{t}$, $\mathbf{y}$, $p_0$, $p_1$
2: OUTPUT: $k$, $b$

3: % Initialize
4: Choose indices $p_0, p_1 \in \{0, 1, \ldots, 10\}$ where $p_0 \neq p_1$

5: % Data
6: $\mathbf{t} \leftarrow [0.01, 0.02, 0.03, 0.04, 0.06, 0.08, 0.10, 0.13, 0.18, 0.24, 0.32]^\mathrm{T}$
7: $\mathbf{y} \leftarrow [1880, 1896, 1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980]^\mathrm{T}$

8: % Compute model parameters
9: $y_0 \leftarrow \mathbf{y}[p_0]$ ;    $t_0 \leftarrow \log_{10} \mathbf{t}[p_0]$
10: $y_1 \leftarrow \mathbf{y}[p_1]$ ;    $t_1 \leftarrow \log_{10} \mathbf{t}[p_1]$
11: $k \leftarrow \frac{t_1 - t_0}{y_1 - y_0}$
12: $b \leftarrow \frac{y_1 t_0 - y_0 t_1}{y_1 - y_0}$
```

we first note from Figure 1.3 that the model seems to fit the data quite well. When using a model for predictions way outside the excitation range where we have supporting data material, it is always a shaky business. It is impossible to know if conditions met in a certain range of years in this case hold a hundred or more years later. Utmost care should be taken not to rely too much on such simple models. In the specific case it seems unrealistic that 11 measurements should describe such complicated effects as global warming. As for the potential reason for what is going on, a model such as the above is obviously way too simple.

If we consider the question asked as to what year y_{UK} the United Kingdom is flooded, except for a few mountain tops, the answer is that in $y_{\mathrm{UK}} = 2091$ the temperature exceeds 7 °C.

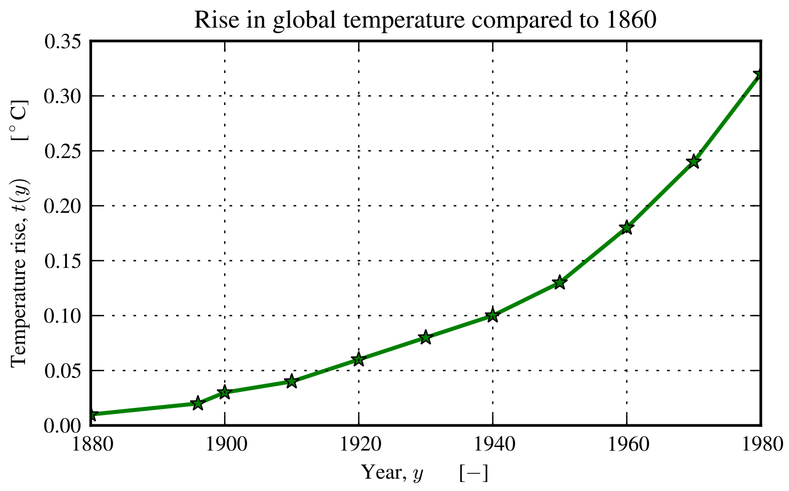


Figure 1.1: Measured rise in global temperature compared to 1860 level, $t(y)$, plotted in linear scale versus year, y. The measurements are indicated by $\star$. Data from [1].

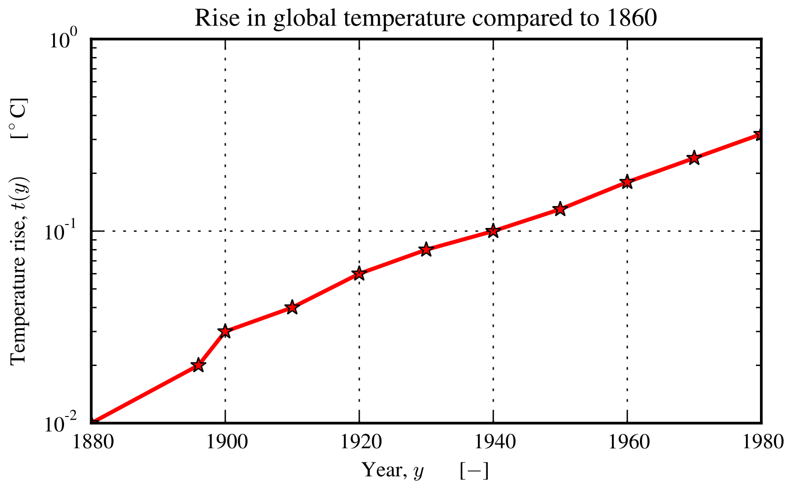


Figure 1.2: Measured rise in global temperature compared to 1860 level, $t(y)$, plotted log-wise versus year, y. The measurements are indicated by $\star$. Data from [1].

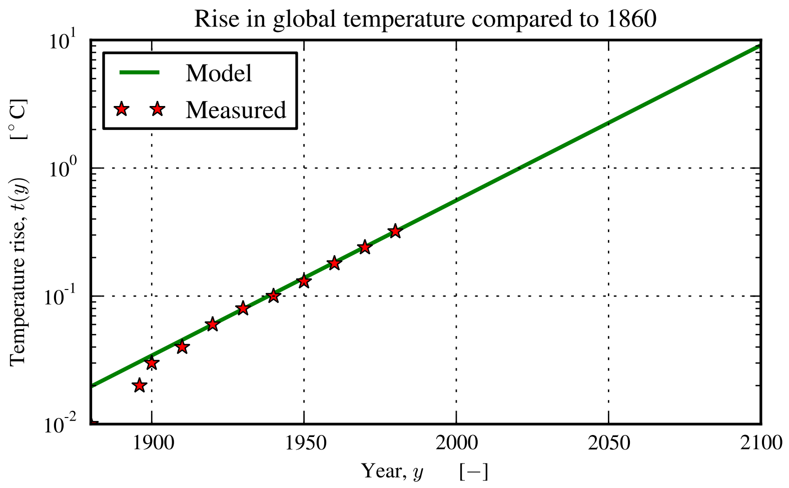


Figure 1.3: Measured rise in global temperature compared to 1860 level, $t(y)$, plotted log-wise versus year, y. The measurements are indicated by a red $\star$ and the model prediction is shown as a green line. Data from [1]. The model parameter extraction is based on 1920 and 1980 data leading to $k = 1.212 \cdot 10^{-2}$ and $b = -24.49$.

Chapter 2

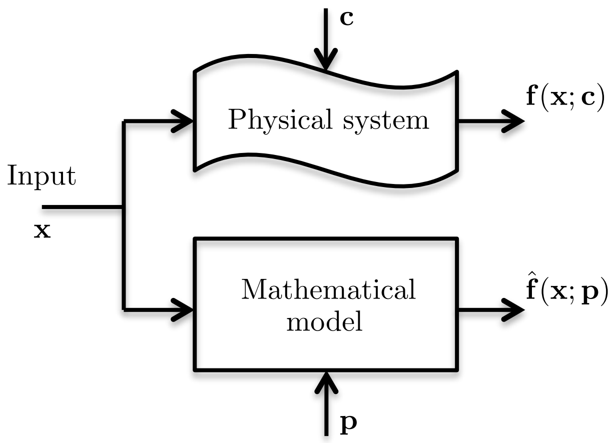
Models and Modelling

Two concepts are required when discussing mathematical modelling. One is the mathematical model, which is a mathematical representation or conceptual simplification of some physical system. This model describes the dependence between some output and an input given some constraints of the system (could for example be temperature, pressure and/or maximum voltage). The second concept is the modelling – this is the procedure or process required to develop the model including a clear recipe for how to identify or extract model parameters. This chapter provides a brief introduction to this subject. Since mathematical modelling is a huge and complex field, the chapter only provides a primer to the subject to avoid over-complicating things.

2.1 Mathematical model

A mathematical model is a formalised mathematical description of a (physical) system describing some relations between input(s) and output(s) with some system-describing parameters. Mathematical modelling is the process of establishing the model and the parameters describing the model. Often the purpose of having a model is that it allows one to describe some system by relatively few parameters rather than having a huge set of (often measured) data to represent the system behaviour. Also, we can test the mathematical model with signals we may not even be able to construct in reality. As long as the mathematical model and the modelling procedure are well designed, it is often possible to perform model-based computations to determine system behaviour in many different scenarios. The concepts can be explained from Figure 2.1. Typically, we have some physical system for which we have some knowledge of what it is and how it operates. Often we also have a number of measurements describing the system under some different conditions. Say that we describe the conditions by a vector:

$$\mathbf{c} \;=\; [c_0, \ldots, c_{L-1}]^{\mathrm{T}} \;\in\; \mathbb{R}^{L\times 1} \tag{2.1}$$



$\mathbf{c}$
Physical system
$\mathbf{f}(\mathbf{x};\mathbf{c})$
Input
$\mathbf{x}$
Mathematical model
$\hat{\mathbf{f}}(\mathbf{x};\mathbf{p})$
$\mathbf{p}$

Figure 2.1: Illustration of the modelling system. We have a physical system operating under some conditions described by $\mathbf{c}$ which is excited by some input $\mathbf{x}$ leading to the output $\mathbf{f}(\mathbf{x};\mathbf{c})$. A mathematical model is derived which is described by some model parameters $\mathbf{p}$ and excited with the same input $\mathbf{x}$ as the physical system leads to the response $\hat{\mathbf{f}}(\mathbf{x};\mathbf{p})$.

This $\mathbf{c}$-vector may for example include minimum and maximum temperatures used when the measurements were taken. The input to the system may be described by:

$$\mathbf{x} \;=\; [x_0, \ldots, x_{N-1}]^{\mathrm{T}} \;\in\; \mathbb{R}^{N\times 1} \tag{2.2}$$

Then we have the output (or response) from the physical system as:

$$\begin{aligned} \mathbf{y} &= \mathbf{f}(\mathbf{x};\mathbf{c}) && (2.3)\\ &= [y_0, \ldots, y_{N-1}]^{\mathrm{T}} \;\in\; \mathbb{R}^{N\times 1} && (2.4) \end{aligned}$$

From the mathematical model we similarly have the same type of input $\mathbf{x}$ as given by (2.2). The mathematical model is further described by a set of parameters described by:

$$\mathbf{p} \;=\; [p_0, \ldots, p_{I-1}]^{\mathrm{T}} \;\in\; \mathbb{R}^{I\times 1} \tag{2.5}$$

As an example, the parameter vector $\mathbf{p}$ could contain the parameters k and b used in Chapter 1. The output (or response) from the mathematical model can be described as:

$$\begin{aligned} \hat{\mathbf{y}} &= \hat{\mathbf{f}}(\mathbf{x};\mathbf{p}) && (2.6)\\ &= [\hat{y}_0, \ldots, \hat{y}_{N-1}]^{\mathrm{T}} \;\in\; \mathbb{R}^{N\times 1} && (2.7) \end{aligned}$$

We define a function to describe the agreement between the response from the physical system and the mathematical model via:

$$\mathcal{P}(\mathbf{y};\hat{\mathbf{y}}) \;=\; \mathcal{P}\left(\hat{\mathbf{f}}(\mathbf{x};\mathbf{p}), \mathbf{f}(\mathbf{x};\mathbf{c})\right) \tag{2.8}$$

This agreement can be described in many different ways such as:

$$\mathcal{P}(\mathbf{y};\hat{\mathbf{y}}) = \sum_{m=0}^{M-1} |y_m - \hat{y}_m| \tag{2.9}$$

or

$$\mathcal{P}(\mathbf{y};\hat{\mathbf{y}}) = \sum_{m=0}^{M-1} (y_m - \hat{y}_m)^2 \tag{2.10}$$

but other metrics can of course also be defined such as a relative difference measure of $\mathbf{y} - \hat{\mathbf{y}}$.

The task of determining the model parameters $\mathbf{p}$ is then basically an optimisation problem where we aim to solve the following:

$$\begin{aligned} \underset{\mathbf{p}}{\text{minimise}} \quad & \mathcal{P}(\mathbf{y};\hat{\mathbf{y}}) \\ \text{subject to} \quad & \mathcal{C}(\mathbf{p}) \end{aligned} \tag{2.11}$$

where $\mathcal{C}(\mathbf{p})$ expresses some possible constraints on the model parameters – for example that a model parameter p_i must be positive. Seen from a model point of view, the condition vector $\mathbf{c}$ should not be part of (2.11). The conditions are related to the physical system and therefore only indirectly represents information about which conditions the measurements were achieved under. However, indirectly it is of course important as $\mathbf{c}$ may for example include information on minimum and maximum temperature used when measuring the system. This means that the model only with care should be used outside this temperature range if the model somehow includes temperature information.

When we develop a model there are often many different techniques that can be applied and often the applied model is developed for a specific application. This means that there is not one single 'mother' model that embraces all models. There are some classes, though, that can typically be formulated like:

- Black-box models.
  These are models where we have no information of the physical system we need to model. We only have access to a more or less elaborate set of measurement points described as:

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{f}_0(\mathbf{x}_0;\mathbf{c}_0) \\ \mathbf{y}_1 &= \mathbf{f}_1(\mathbf{x}_1;\mathbf{c}_1) \\ &\vdots \\ \mathbf{y}_{Q-1} &= \mathbf{f}_{Q-1}(\mathbf{x}_{Q-1};\mathbf{c}_{Q-1}) \end{aligned}$$

This means that we in this case need to develop a model based entirely on knowledge of measured data. This is an extremely difficult task and the task of the model of course is to provide answers to what response we achieve for an input and/or conditions we did not have available for the model development.

- White-box models.
  This kind of models is the opposite of black-box models. Here we have all information of the physical system and we also have empirical data for the system's behaviour for different excitation signals and conditions. While modelling is rarely easy, this kind of modelling is in many cases a reasonable task.

- Grey-box models.
  This type of models is somewhere between the black- and white-box models. For grey-box models we usually have some knowledge of the physical system and we may also have some measured data. Often we do not have the full information of the system available – or it may not be fully understood. We may have some measurements available but this is not always the case. Sometimes we need to rely on knowledge of the physical system alone. This could for example be a situation where we need to rely on thermodynamics, Maxwell's equations, Newton's laws or similar.

Normally, we are not in a position to freely choose the type of model we prefer. In some cases, a vendor of some product we may wish to model often only provides information of what type of device or physical system we have and some measurements in a data sheet. In such cases we, can do little but to choose a black-box approach.

In other cases, we may have quite detailed information of the underlying physical system. For a mechanical system; perhaps some measurements combined with drawings, materials etc. that would allow us to make a model based on some physical laws (e.g. Newton's laws, Maxwell's equations, and the thermodynamic laws). This would point more in the direction of a white-box model.

If we in the white-box example have access to the detailed drawings and some specifications for the used materials etc., we could derive a model. Assuming that we do not have any measurements available, we are facing a grey-box model indicating that we have some but far from all information on the physical system.

This boils down to a practical side where the specific model to use is often not a choice as such. Often the decision is dictated by the amount of information we have available when the model must be derived. This is also one of the reasons that make modelling a challenging task as each problem often ends up with a specific solution. There are many things in common between different approaches, but often some fine tuning is needed in the specific situation which means that we must use some creativity and some artistic freedom. What this means is then that there does normally not exist one and only one model for a specific modelling problem – normally it is possible to come up with several models and then it is important to have some objective criteria to use when choosing the specific model to use. This could for example be issues such as model complexity, ease of parameter extraction, model fit to measurements, and extendability just to mention a few.

2.2 Modelling procedure

The modelling procedure is a multi-step process which can be described as:

1. **Specify the problem**.
   The first step is to understand the problem to be analysed. This includes an analysis of all available information. At this stage it is usually best to use common terms (and not math) to describe different knowledge.

2. **Model purpose**.
   Once the problem is understood and formulated, the next step is to clarify the purpose of the model. The main reason for this is that the more we wish the model to describe, the higher complexity we have in terms of e.g. variables, required math, simulation time and perhaps even more importantly the chance of making errors in the process. Therefore, the model should describe what is needed and nothing more. One very important aspect of this is to clarify the assumptions that we may be able to apply. We obviously need to check later that the assumptions are fair and do not lead to incorrect or unacceptably imprecise results.

3. **Model proposal**.
   Next we make a proposal for the model given the knowledge we have. Normally, one should always start from a simple model – also sometimes even if it is known to be too simple. No matter what model we propose it must be done such that we can fairly easily expand the model to include more detail. Normally, any model includes some changing input which together with some model parameters maps to an output which is typically what we want to further study and analyse.

4. **Develop the mathematics and algorithm**.
   Given a proposal for a model we need to develop the necessary mathematics. This typically covers two parts:

   (a) A procedure to determine model parameters – these are parameters which are not part of any input we wish to analyse. Often we here need to have measured data or knowledge of likely initial behaviour of the system we analyse.
   (b) A relation for the desired output we wish to analyse given model parameters and input to the system. Sometimes we can derive closed-form expressions for this but more often the output determination demands computer simulations or computations to be determined.

   Once the mathematical relations are in place we normally need to reformulate this to an algorithm[1], which is a description suitable for computer implementation.

5. **Implement and test algorithm**.
   Next we usually need to implement a computer program to perform the simulations or computations needed for us to analyse the model. Typically, we need to investigate how the model behaves with various settings of the model parameters and with different types of model input. Also we should check different boundary cases to check that the model provides useful results within the parameter boundaries we have decided on. Test scripts should also be made to ensure that we do not make mistakes in the computer programming itself. This is a typical place for errors and it is a must to have procedures to validate the behaviour. The testing part also includes a thorough inspection of the assumptions to ensure that the model is only used within its limits of applicability.

6. **Perform simulations**.
   Once the computer implementation has been verified it is time to describe all the simulation setups that we wish to analyse. This is often done via configuration files describing specific values for model input and parameters that must be analysed. By using such an approach it is way easier to just start simulations and we ensure that for any

[1]An algorithm is not a formally and well defined term but normally an algorithm is seen as a step-by-step procedure for how to compute the states and response from some system. The algorithm is normally the link between the mathematics describing an analysis of a given problem and the code needed to perform the computations to provide the desired results. This means that the algorithm must be quite detailed and clearly specify all inputs to the computations, in what order the different computations must be performed and all outputs. For an example of an algorithm see Algorithm 4.1 on page 26. An algorithm is sometimes referred to as pseudo-code which means that the algorithm resembles code but it is not linked to any specific programming language. However, with the algorithm it must be possible for one with sufficient knowledge in e.g. *Python* to map the algorithm to a *Python* program.

result, we know the precise setting of parameters that led to this result. This is crucial to ensure that the work is reproducible by both ourselves and others.

7. **Interpret results**.
   Then we need to interpret the results. Often we have lots of data material and efficient data analytic methods should be found. Often we can plot the data in 2 or 3 dimensions to get a fast overview of the results. Other times we need to post-process the data to compute e.g. averages[2] or standard deviations[3].

8. **Reiterate**.
   Sometimes the initially proposed model is not sufficiently accurate – or it may be over-complicated (hopefully not, though). If necessary, we need to dig into the results and propose modifications to the model and enter step 3 once more.

When applying this procedure, it is important to remember that modelling is not an exact science in the sense that several models can usually be derived, each of which leads to a reasonable agreement with the physical system behaviour. It is therefore very important to have considered the objectives that are considered important when a specific model to use is selected.

[2]The average value (or arithmetic mean) of a sequence of data points $x_0, x_1, \ldots, x_{N-1}$ is:

$$\mu_\mathrm{x} \ = \ \sum_{n=0}^{N-1} x_n \tag{2.12}$$

μ_x expresses the central value of the data points obtained by adding all data values and divide by the number of data points. Note that the unit of the average is the same as the unit of the data whatever that might be.

[3]The standard deviation of data points $x_0, x_1, \ldots, x_{N-1}$ is:

$$\sigma_\mathrm{x} \ = \ \sqrt{\sum_{n=0}^{N-1} [x_n - \mu_\mathrm{x}]^2} \ \geq \ 0 \tag{2.13}$$

The standard deviation σ_x is an expression of the variation of the data points from the average. If the standard deviation is low, we know that the data is fairly close to the average and if the standard deviation is high, at least some data points are quite far from the average. Note that the standard deviation has the same unit as the data itself no matter what that unit may be. This allows for a fairly easy and direct interpretation of the standard deviation.

Chapter 3

Example: sinusoid

This chapter describes an example where we know that the behaviour should be sinusoidal, but where the available measurements could be affected by noise of some kind. Thus the measurements are not perfect. We need to find a simple method to extract three model parameters to fit the measurements. The model parameters are an amplitude, a frequency and an initial phase. A simple but robust extraction is very difficult to achieve and we propose one solution – more simple than robust though.

3.1 Model

Suppose we have a set of data describing some simple physical system. We are informed that the system is observed at N time instants as $\frac{T}{N} \cdot (0, 1, \ldots, N-1)$ where $(\cdots)$ denotes a sequence of data points. This means we cover the time interval $t \in [0; T)$ where the time instants can conveniently be collected in a time vector as:

$$\begin{aligned}
\mathbf{t} &= [t_0,\, t_1,\, \ldots,\, t_{N-1}]^{\mathrm{T}} \in \mathbb{R}^{N\times 1} && (3.1)\\
&= [\mathbf{t}[0],\, \mathbf{t}[1],\, \ldots,\, \mathbf{t}[N-1]]^{\mathrm{T}} && (3.2)\\
&= \frac{T}{N} \cdot [0,\, 1,\, \ldots,\, N-1]^{\mathrm{T}} && (3.3)
\end{aligned}$$

At each of the N time instants we have the corresponding observed data values described by a vector as:

$$\begin{aligned}
\mathbf{d} &= [d_0,\, d_1,\, \ldots,\, d_{N-1}]^{\mathrm{T}} \in \mathbb{R}^{N\times 1} && (3.4)\\
&= [\mathbf{d}[0],\, \mathbf{d}[1],\, \ldots,\, \mathbf{d}[N-1]]^{\mathrm{T}} && (3.5)\\
&= [d(t_0),\, d(t_1),\, \ldots,\, d(t_{N-1})]^{\mathrm{T}} && (3.6)
\end{aligned}$$

In a specific case, say we have $N = 20$ data points observed over a time window of duration $T = 1$. The time and data vectors are:

$$\begin{aligned}
\mathbf{t} &= \big[0.00,\ 0.05,\ \ldots,\ 0.95\big]^{\mathrm{T}} \in \mathbb{R}^{20\times 1} && (3.7)\\
\mathbf{d} &= \big[0.00000000,\ 0.309016994,\ 0.587785252, \\
&\quad 0.809016994,\ 0.951056516,\ 1.00000000, \\
&\quad 0.951056516,\ 0.809016994,\ 0.587785252, \\
&\quad 0.309016994,\ 0.00000000,\ -0.309016994, \\
&\quad -0.587785252,\ -0.809016994,\ -0.951056516, \\
&\quad -1.00000000,\ -0.951056516,\ -0.809016994, \\
&\quad -0.587785252,\ -0.309016994\big]^{\mathrm{T}} \in \mathbb{R}^{20\times 1} && (3.8)
\end{aligned}$$

Seen from a general point of view, we need to know the N time instants meaning that we at first glance may get the impression that we need to handle N real numbers. In this specific and very simple example, we could do with just information on N and T since the time between data points is the same for all time instants. In the simpler case we need to handle 2 real numbers (in principle N is normally an integer but for simplicity and without loosing generality we just say that it may be a real).

Thus in the more general case we need to store or handle $2N$ real numbers. Obviously, we could just store all these values and use that to represent the behaviour of the system. If N is large it is clear that we need to store a lot of data for something that may be fairly simple to represent in a different way. This is where a model comes into play. A model is here a conceptually simpler way to describe the behaviour of the system. Or at least simpler than having all the data points available. From Figure 3.1 one might get the idea that the data points are close to describing a sinusoidal behaviour versus time. In such a case we could propose a continuous-time model like:

$$d_{\mathrm{m}}(t) \;=\; A_{\mathrm{m}} \cdot \sin(2\pi f_{\mathrm{m}} t + \phi_{\mathrm{m}}), \quad t \in [0;\, T) \tag{3.9}$$

where index 'm' indicates it is a model response (or A_{m}, f_{m}, and ϕ_{m} are model parameters). When observing the measurements in Figure 3.1 we only have a limited number of those – $N = 20$ in this particular case. However, for the model we are not limited to this. We may choose a different number of 'model time points' such as M which may be, and often is, larger than the number of actual measurement points, N. The description of $d_{\mathrm{m}}(t)$ in (3.9) uses a continuous time t. Making this fit with M evenly distributed time samples in the same interval $t \in [0;\, T)$ as earlier we have:

$$t \;\leftarrow\; m \cdot \frac{T}{M}, \quad m = 0, 1, \ldots, M-1 \tag{3.10}$$

Inserting this discretisation of time into (3.9), we have the following time-sampled model:

$$d_\mathrm{m}(mt_\delta) \; = \; A_\mathrm{m} \cdot \sin(2\pi f_\mathrm{m} m t_\delta + \phi_\mathrm{m}), \quad m = 0, 1, \ldots, M-1 \qquad (3.11)$$

with $t_\delta = \frac{T}{M}$ from (3.10). Using the vector notation from earlier and a single intermediate variable we have:

$$\mathbf{d}_\mathrm{m}[m] \; = \; A_\mathrm{m} \cdot \sin(2\pi f_\mathrm{m} \mathbf{t}[m] + \phi_\mathrm{m}), \quad m = 0, 1, \ldots, M-1 \qquad (3.12)$$

From Table 3.1 we see that the physical model has a complexity given as the number of reals needed to describe the system as $2 + N$ which comes from knowledge of T and M combined with N measurements. The model has a significantly lower complexity of only $2 + 3 = 5$ real numbers to describe T, M, A_m, f_m, and ϕ_m. For a large number of measurements N, the model is therefore of substantially lower complexity, which is an important reason for using models in the first place. We of course lose something when using the simpler model. In this case the measurements are affected by noise and the model we have used here does not include any information on that at all. It is possible to make other models such that noise is also included in the model, for example such that it ensures agreement with the physical measurements regarding noise average and noise variance.

	$\mathcal{S}(\mathbf{t})$	$\mathcal{S}(\mathbf{d})$
	Number of reals	Number of reals
Physical	2	N
Model	2	3

Table 3.1: (S)torage complexity $\mathcal{S}(\cdot)$ given as the number of reals needed to store the necessary data for the **t**ime- and **d**ata-describing vectors for the physical data and for the model proposed in (3.9).

3.2 Parameter extraction

The parameter extraction process is a formal and rigorous description of how to determine the parameters A_m, f_m and ϕ_m from the model described by (3.12). Obviously, we can rarely expect that the model fits perfectly to the measured data – typically some noise is included when we take the measurements such that our physically measured data at a given time t can be described by:

$$d(t) \; = \; d_\mathrm{m}(t) \; + \; n(t) \qquad (3.13)$$

where $n(t)$ represents a noise contribution at the given time. Note that far from all systems have noise obeying this additive representation. We assume in the following that the noise data points can be ignored when determining

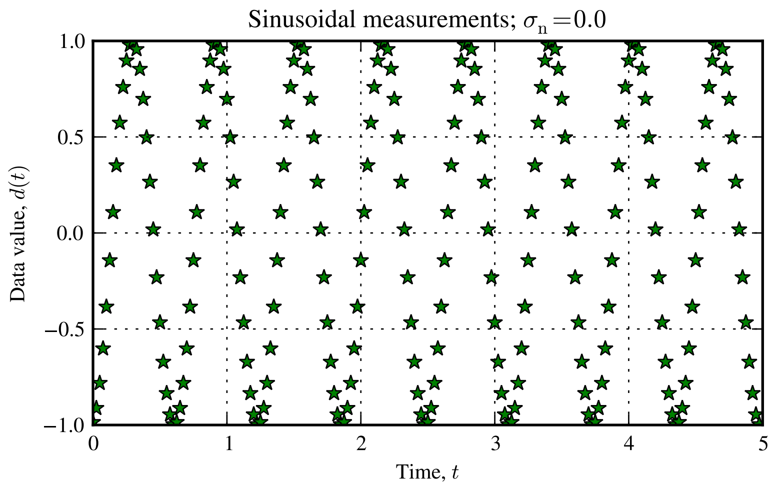


Figure 3.1: Example of observed noise free data versus time for a physical system when observing in a time interval $T = 1$. We have time $t = n \cdot \frac{T}{N}$ with $n = 0, 1, \ldots, 19$ and data values corresponding to these time instants. $\sigma_n = 0.0$ indicates a noise free set of data points.

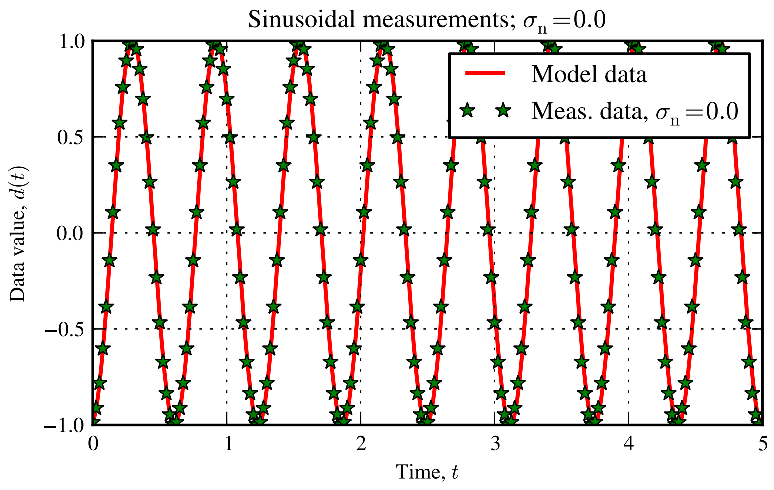


Figure 3.2: Example of observed noise free data from a physical system and model generated data versus time. The physical system has 20 measurement points and the model generates 400 data points. $\sigma_n = 0.0$ indicates a noise free set of data points.

model parameters. This simplification is often unrealistic but it is used here to illustrate the concepts. Of course there may be specific time instants where the noise is dominating the signal but when computing the variance (square of the standard deviation – representative for the signal energy) this effect can be safely ignored in many cases.

Knowing the model and that the noise part is not of concern, it means that the measured data we have fits the model fairly well. With this knowledge we can first use that the magnitude A_m can be determined from two data samples as:

$$A_\mathrm{m} \;=\; \frac{1}{2}\left\{\max(\mathbf{d}) \;-\; \min(\mathbf{d})\right\} \;>\; 0 \tag{3.14}$$

If we then consider the phase ϕ (or the time offset of the sinusoidal) we can use the measured data at time $t = 0$ where:

$$\phi_\mathrm{m} \;=\; \arcsin\left[\frac{\mathbf{d}[0]}{A_\mathrm{m}}\right], \quad |\phi_\mathrm{m}| \le \frac{\pi}{2} \tag{3.15}$$

Finally, the frequency f_m can be estimated via knowledge of the zero-crossings of the $\mathbf{d}$ signal. As $\mathbf{d}$ is affected by noise, we use a procedure to estimate the time index where zero-crossings occur as:

$$\begin{aligned}\mathbb{I} = \Big\{ & i : \operatorname{sgn}\{\mathbf{d}[i-1]\} = \operatorname{sgn}\{\mathbf{d}[i]\} = -\operatorname{sgn}\{\mathbf{d}[i+1]\} = -\operatorname{sgn}\{\mathbf{d}[i+2]\}\,, \\ & i = 1, 2, \ldots, N-3 \Big\} \qquad (3.16) \\ = \{ & i_0, i_1, \ldots, i_{P-1} \} \qquad (3.17)\end{aligned}$$

where $\mathbb{I}$ is an ordered set with $i_0 < i_1 < \cdots < i_{P-1}$, and:

$$\operatorname{sgn}\{x\} \;=\; \begin{cases} +1 & \text{for } x \ge 0 \\ -1 & \text{otherwise} \end{cases} \tag{3.18}$$

To get a fairly noise-robust estimate of f_m we can use all the zero crossings as:

$$f_\mathrm{m} \;=\; \frac{1}{2\,(P-1)} \sum_{p=0}^{P-2} \frac{1}{\mathbf{t}[i_{p+1}] \;-\; \mathbf{t}[i_p]} \tag{3.19}$$

The factor '2' in (3.19) is due to the fact that $\mathbf{t}[i_{p+1}] - \mathbf{t}[i_p]$ corresponds to only half a sinusoidal period of the signal.

The above procedure is somewhat sensitive to noise – it does however help a bit that two consecutive samples above/below the zero crossing must exist. But still, quite few measurements are actually used to estimate the frequency, which is almost always an indication of a somewhat sensitive estimate. However, the procedure works reasonably well in (low) noise conditions and simplicity wins over accuracy and robustness here.

The extraction procedure is described in Algorithm 3.1 which includes determination of all model parameters A_m, f_m and ϕ_m.

Algorithm 3.1 Algorithm for extraction of parameters A_m, f_m and ϕ_m.

```
1:  INPUT: $\mathbf{t}$, $\mathbf{d}$
2:  OUTPUT: $A_\mathrm{m}$, $f_\mathrm{m}$, $\phi_\mathrm{m}$

3:  % Compute model parameters
4:  $A_\mathrm{m} \leftarrow \frac{1}{2}(\max(\mathbf{d}) - \min(\mathbf{d}))$
5:  $\phi_\mathrm{m} \leftarrow \arcsin\left[\frac{\mathbf{d}[0]}{A_\mathrm{m}}\right]$

6:  $\mathbb{I} \leftarrow \emptyset$ ; $P \leftarrow 0$
7:  for $i$ in $(2, 3, \ldots, N-3)$:
8:      if $(\mathrm{sgn}\{\mathbf{d}[i-1]\} = \mathrm{sgn}\{\mathbf{d}[i]\} = -\mathrm{sgn}\{\mathbf{d}[i+1]\} = -\mathrm{sgn}\{\mathbf{d}[i+2]\})$
9:          $\mathbb{I} \leftarrow \mathbb{I} \cup \{i\}$ ; $P \leftarrow P+1$
10:     endif
11: endfor
12: $f_\mathrm{m} \leftarrow \frac{1}{2(P-2)} \sum_{p=0}^{P-2} \frac{1}{\mathbf{t}[i_{p+1}] - \mathbf{t}[i_p]}$
```

3.3 Results

The results obtained from applying the model extraction procedure outlined earlier in the noise-free case designed with parameters $N = 200$, $M = 4000$, $\sigma_\mathrm{n} = 0.0$, $A_\mathrm{m} = 1.0$, $f_\mathrm{m} = 1.6$, and $\phi_\mathrm{m} = -1.4$ led to the parameters:

```
                   True   ;    Estimated
Amplitude:         1.00   ;       1.00
Frequency:         1.60   ;       1.60
Phase:            -1.40   ;      -1.41
```

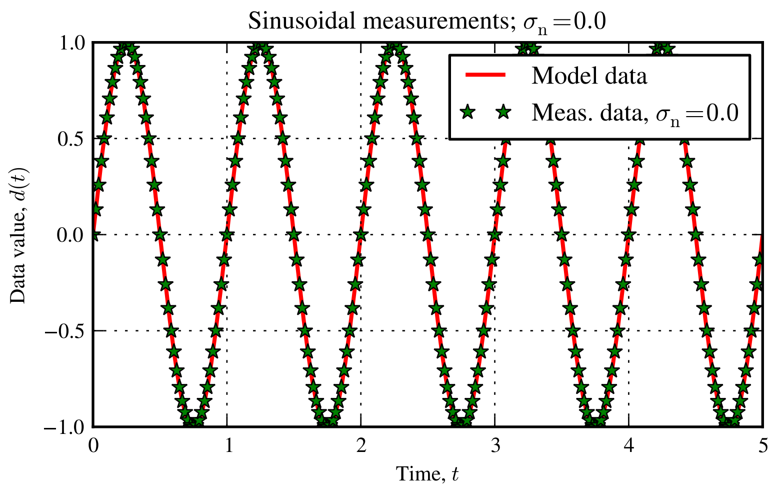


Figure 3.3: Example of observed noise-free data from a physical system and model-generated data versus time. The physical system has 200 measurement points and the model generates 4000 data points.

You can find the data points and modelled response in Figure 3.3.

A more challenging noisy example with parameters $N = 200$, $M = 2000$, $\sigma_\mathrm{n} = 0.02$, $A_\mathrm{m} = 1.0$, $f_\mathrm{m} = 1.0$, and $\phi_\mathrm{m} = 0.0$ led to the parameters:

```
                 True   ;    Estimated
Amplitude:       1.00   ;       1.03
Frequency:       1.00   ;       1.00
Phase:           0.00   ;       0.02
```

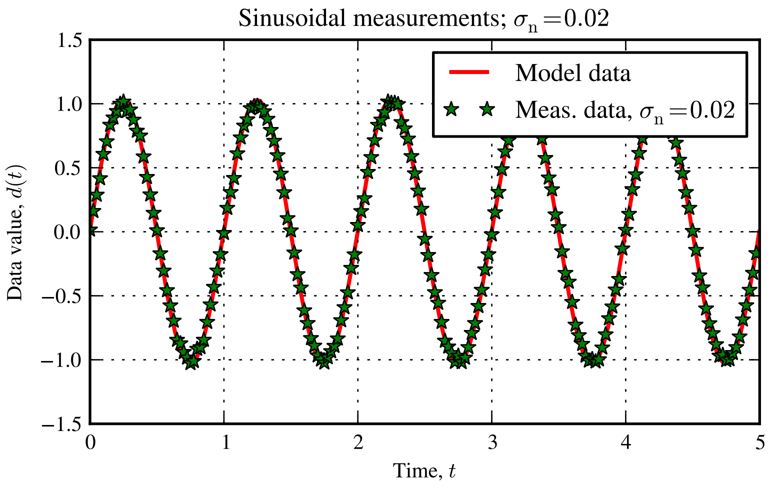


Figure 3.4: Example of observed noisy data from a physical system and model-generated data versus time. The physical system has 200 measurement points and the model generates 2000 data points.

The results in terms of data points and modelled behaviour are shown in Figure 3.4.

Another noisy case is with parameters $N = 200$, $M = 2000$, $\sigma_\mathrm{n} = 0.04$, $A_\mathrm{m} = 0.8$, $f_\mathrm{m} = 1.0$, and $\phi_\mathrm{m} = 0.2$ which led to the result:

```
                 True   ;    Estimated
Amplitude:       0.80   ;       0.90
Frequency:       1.00   ;       0.99
Phase:           0.20   ;       0.18
```

The results in terms of data points and modelled behaviour are shown in Figure 3.5.

Finally, another noisy example with parameters $N = 200$, $M = 2000$, $\sigma_\mathrm{n} = 0.07$, $A_\mathrm{m} = 0.9$, $f_\mathrm{m} = 1.2$, and $\phi_\mathrm{m} = -0.2$ led to the result:

```
                 True   ;    Estimated
Amplitude:       0.90   ;       1.02
Frequency:       1.20   ;       1.20
Phase:          -0.20   ;      -0.20
```

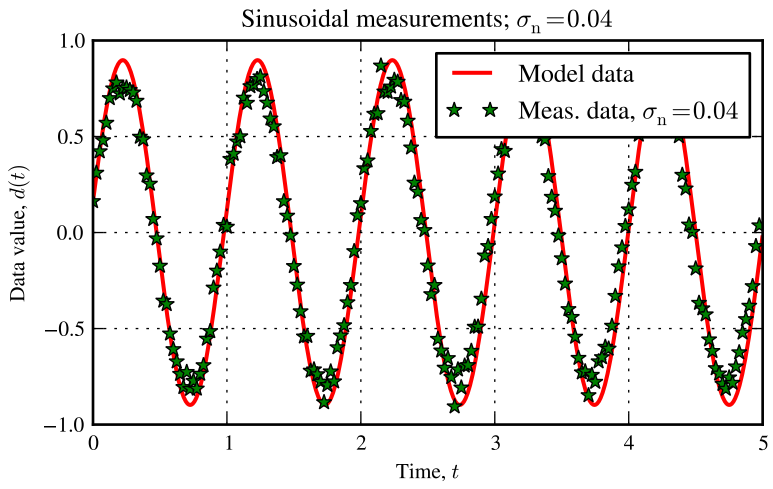


Figure 3.5: Example of observed noisy data from a physical system and model generated data versus time. The physical system has 200 measurement points and the model generates 2000 data points.

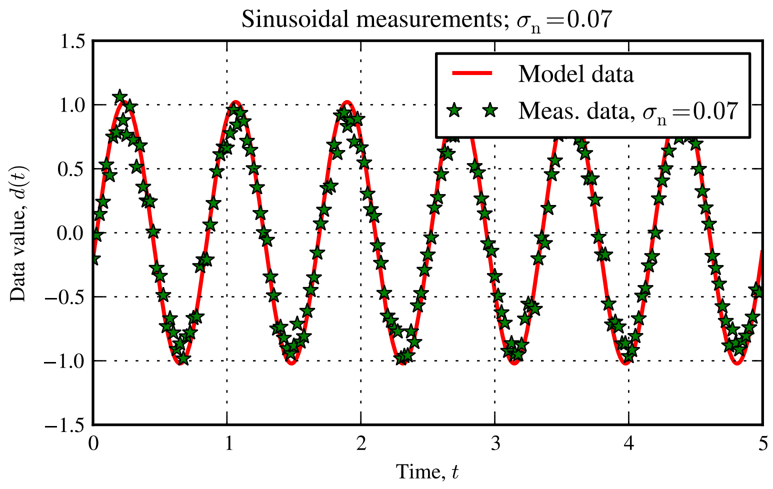


Figure 3.6: Example of observed noisy data from a physical system and model generated data versus time. The physical system has 200 measurement points and the model generates 2000 data points.

The results in terms of data points and modelled behaviour are shown in Figure 3.6.

Although the parameter extraction procedure is very simple, it seems to work reasonably well for low noise conditions. It is relatively easy to see nonsense results, because the procedure is not very robust to noise as the extracted parameters are only based on very few measurements.

Chapter 4

Example: driven damped oscillatory system

In this example we look at a driven damped oscillatory system. Such systems are seen in several areas – for example in electrical engineering as resonance circuits and in mechanical engineering with e.g. a mass connected to a spring with an external force applied. In this example we assume that we do not have access to any measurements but have derived a key describing equation from some fundamental laws of physics such as Newton's laws, and Kirchhoff's laws for voltage/current (special case of Maxwell's equations) as well as a system description to clarify the problem. The focus in on the mechanical example but the behaviour is precisely the same for a driven *RLC* circuit in electrical engineering.

4.1 Mathematical model

First we present the basic key equation describing the mechanical system in continuous time. We do not derive the model as such but leave that for the interested reader. The starting point is a differential equation derived from Newton's laws. Initially we formulate the problem in continuous time, which we cannot analyse on a digital computer. Therefore, the second part is to perform a simple mapping from continuous to discrete time.

4.1.1 Continuous time

We omit the physical kinematic considerations here and just stipulate the key mathematical equation describing the position $y(t)$ in meters of a fixed point on the system versus time as [2]:

$$m\,\frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2} + b\,\frac{\mathrm{d}y(t)}{\mathrm{d}t} + c\,y(t) = F(t) = F_0\,\sin(\omega t) \tag{4.1}$$

where m is a suspended mass, b is a damping coefficient, and c is a system constant. In (4.1), $F(t)$ is the driving force which in this case is a sine with force magnitude F_0 and oscillating at angular frequency ω. We observe the system in a time window with $t \in [0; T)$. A standard form equation of (4.1) is often formulated as:

$$\frac{\mathrm{d}^2 y(t)}{\mathrm{d}t^2} + 2\,\zeta\,\omega_0\,\frac{\mathrm{d}y(t)}{\mathrm{d}t} + \omega_0^2\,y(t) = \frac{F(t)}{m} \tag{4.2}$$

where the damping factor ζ and natural oscillation angular frequency ω_0 can be derived from (4.1) and (4.2) as:

$$\zeta = \frac{b}{2\sqrt{m\,c}} \tag{4.3}$$

$$\omega_0 = 2\pi f_0 = \sqrt{\frac{c}{m}} \tag{4.4}$$

This means that we can form a parameter vector $\mathbf{p}$ as:

$$\mathbf{p} = [m,\, b,\, c,\, F_0,\, \omega]^{\mathrm{T}} \tag{4.5}$$

We need some initial conditions to describe the system:

$$y(0) = Y_0 \quad \wedge \quad \left.\frac{\mathrm{d}y(t)}{\mathrm{d}t}\right|_{t=0} = 0 \tag{4.6}$$

where the first part is the initial displacement of the fixed position, and the second part describes that the system starts from rest with velocity zero of the fixed point.

4.1.2 Discrete time

Since (4.1) describes a second-order continuous differential equation, we cannot directly solve this by a digital computer. The key to move forward is to map from continuous to discrete time. Without going too much into details on this we make the mapping:

$$t \to n\,t_\delta, \quad n = 0, 1, \ldots, N-1 \tag{4.7}$$

such that $N t_\delta = T$. For correct simulations of the system it is recommended to choose the sampling time t_δ such that $t_\delta \ll \frac{\omega}{\pi}$.[1] We then need to ensure that the time-discrete version of (4.1) is fulfilled at all time steps $n = 0, 1, \ldots, N-1$.

[1]Strictly speaking it is sufficient to choose the sampling time according to $t_\delta < \frac{\omega}{\pi}$ according to the Shannon-Nyquist sampling criterion. To ensure nice and smooth figures for the results, a stricter condition is recommended as indicated above.

We need to map the first and second order derivatives with numerical approximations. This can be done in several ways but here we just apply a simple backward Newton approach to the data points:

$$y_n \;=\; y(t_n) \;=\; y(t)|_{t=n\,t_\delta}\,, \quad n = 2, 3, \ldots, N-1 \tag{4.8}$$

such that:

$$\begin{aligned} y(t)|_{t=n\,t_\delta} &= y(n\,t_\delta) && (4.9)\\ &= y_n && (4.10)\\ \left.\frac{\mathrm{d}y(t)}{\mathrm{d}t}\right|_{t=n\,t_\delta} &\simeq \frac{1}{t_\delta}\,[y(n\,t_\delta) - y(n\,t_\delta - t_\delta)] && (4.11)\\ &\simeq \frac{1}{t_\delta}\,[y_n - y_{n-1}] && (4.12)\\ \left.\frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2}\right|_{t=n\,t_\delta} &\simeq \frac{1}{t_\delta^2}\,[y(n\,t_\delta) - 2\,y(n\,t_\delta - t_\delta) + y(n\,t_\delta - 2\,t_\delta)] && (4.13)\\ &\simeq \frac{1}{t_\delta^2}\,[y_n - 2\,y_{n-1} + y_{n-2}] && (4.14) \end{aligned}$$

Mapping (4.1) into time-discrete form then gives us:

$$\frac{m}{t_\delta^2}\,[y_n - 2\,y_{n-1} + y_{n-2}] \;+\; \frac{b}{t_\delta}\,[y_n - y_{n-1}] \;+\; c\,y_n \;=\; F_0\,\sin(\Omega n) \tag{4.15}$$

where:

$$\omega n t_\delta \;=\; 2\pi f\,n\,\frac{1}{f_\delta} \;=\; \Omega\,n, \quad \Omega = 2\pi\,\frac{f}{f_\delta} \tag{4.16}$$

Due to the initial conditions in (4.6), we have in discrete time that:

$$y_0 \;=\; Y_0 \quad \wedge \quad y_1 \;=\; y_0 \;=\; Y_0 \tag{4.17}$$

The latter part in (4.17) ensures that the initial velocity is zero. We then need to map (4.15) to describe y_n with $n = 2, 3, \ldots, N-1$ to determine $y_2, y_3, \ldots, y_{N-1}$ – where we already have y_0 and y_1 from the initial conditions in (4.17). Rewriting (4.15) we get:

$$y_n \;=\; \frac{t_\delta^2}{\alpha}\,F_0\,\sin(\Omega n) \;+\; \frac{\beta}{\alpha}\,y_{n-1} \;-\; \frac{m}{\alpha}\,y_{n-2}, \quad n = 2, 3, \ldots, N-1 \tag{4.18}$$

where:

$$\begin{aligned} \alpha &= m \;+\; b\,t_\delta \;+\; c\,t_\delta^2 && (4.19)\\ \beta &= 2\,m \;+\; b\,t_\delta && (4.20) \end{aligned}$$

So, in conclusion the key equation governing the behaviour when simulating the driven mechanical oscillatory system is given by (4.18).

4.2 Algorithm

The simulation procedure is shown in Algorithm 4.1. The computational procedure is fairly simple and consists of i) initial conditions; ii) definition of constants; and iii) a loop to advance through time (via time samples). The data of course also needs to be visualised, but this is omitted in the computational algorithm. As normally used for time series data like the data we have here for $y_0, y_1, \dots, y_{N-1}$, we collect this in a vector $\mathbf{y} = [y_0, y_1, \dots, y_{N-1}]$. This data organisation is also applied in Algorithm 4.1.

Algorithm 4.1 Algorithm for the mechanical oscillator.

```
1:  INPUT: $m, b, c, F_0, \omega, Y_0, t_\delta$
2:  OUTPUT: Simulated oscillator output $\mathbf{y}$
3:  % Initialize
4:  $\mathbf{y} \leftarrow [0, 0, \dots 0] \in \mathbb{R}^N$
5:  $\mathbf{y}[0] \leftarrow Y_0$ ;   $\mathbf{y}[1] \leftarrow Y_0$
6:  $\alpha \leftarrow m + b t_\delta + c t_\delta^2$
7:  $\beta \leftarrow 2m + b t_\delta$
8:  % Loop through time
9:  for $n = 2, 3, \dots, N-1$:
10:     $\mathbf{y}[n] \leftarrow \frac{t_\delta^2}{\alpha} F_0 \sin(\Omega n) + \frac{\beta}{\alpha} \mathbf{y}[n-1] - \frac{m}{\alpha} \mathbf{y}[n-2]$
11: endfor
```

4.3 Validation

As part of validating the obtained result, it turns out that a closed-form expression exists for the differential equation in (4.1). The solution can be found as the superposition of a transient part and a steady-state part. Without going into details, we can say that the transient response is a contribution for 'low t' including initial conditions, and the steady state part is a contribution which exists for all t. Due to the nature of the transient response this often dominates for 'low t' whereas the steady state part is practically solely contributing at 'high t'.

In the following, we focus entirely on the steady state solution, which is the one we have after 'some time' (when the transients have died away). The type of differential equation in (4.1) has a steady state solution as:

$$y(t) = A \sin(\omega t + \psi) \tag{4.21}$$

A quick look at (4.1) indicates that $y(t)$ in (4.21) is indeed a solution to the key equation in (4.1). By inserting (4.21) into (4.1) it is possible to derive expressions for A and ψ to obtain the theoretical steady-state solution. The derivation is made by use of trigonometric relations combined with identifying terms involving $\cos(\omega t)$ and $\sin(\omega t)$. This procedure is trivial although a bit time-consuming and leads to:

$$\psi = \arctan\left[\frac{b\,\omega}{\omega^2 m\ -\ c}\right] \tag{4.22}$$

$$A = -\frac{F_0}{b\,\omega}\,\sin(\psi) \tag{4.23}$$

When comparing the theoretical result in (4.21) and (4.22)–(4.23) we should notice that we in the validation only see the steady state solution – i.e. the result after the transient behaviour seen particularly for lower t has decayed. In addition, we should also notice that we use a rather crude and potentially unstable direct backward numerical differentiation method where a more sophisticated method such as Runge-Kutta may be a better choice in practical settings [3, Ch. 17]. In this example, simplicity is a major concern, which is the reason for a suboptimal choice of numerical differentiation method.

4.4 Simulation results

Simulations as well as the steady-state solutions (large t) have been performed for different conditions. One example is shown in Figures 4.1–4.2. The specific data used is the same for the two figures where Figure 4.1 shows 700 data samples starting from $t = 0$ (equivalent to $n = 0$). Figure 4.2 shows 700 data samples, further into the signal at $t = 1215$, where the signal seems to approximately have reached a steady-state behaviour. In terms of validation, Figures 4.4–4.3 show a comparison of the simulated output and the computed steady-state solution from (4.22)–(4.23). As expected, there is a substantial difference between the simulated result and the steady state computed output for low t, which is confirmed by Figure 4.3. When observing the similar signals for higher t, we see a much better agreement. There seems to be a small time offset and also a small difference in amplitude. However, a more detailed study has not been performed to find the precise reasons for the small differences. For the purpose here it is considered that Figure 4.4 confirms the simulated result.

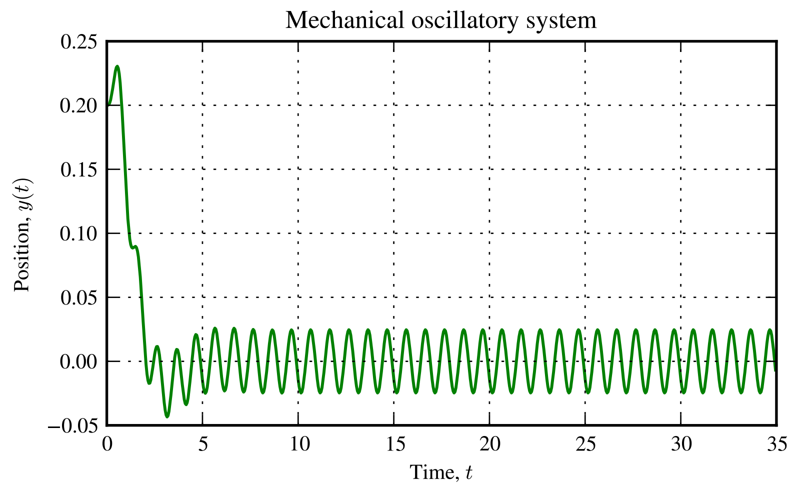


Figure 4.1: Position $y(t)$ versus time t for the mechanical oscillatory system example. The data used is: $m = 1.0$, $b = 1.7$, $c = 2.0$, $F_0 = 1.0$, $f = 1.0$, $Y_0 = 0.2$, $N = 700$.

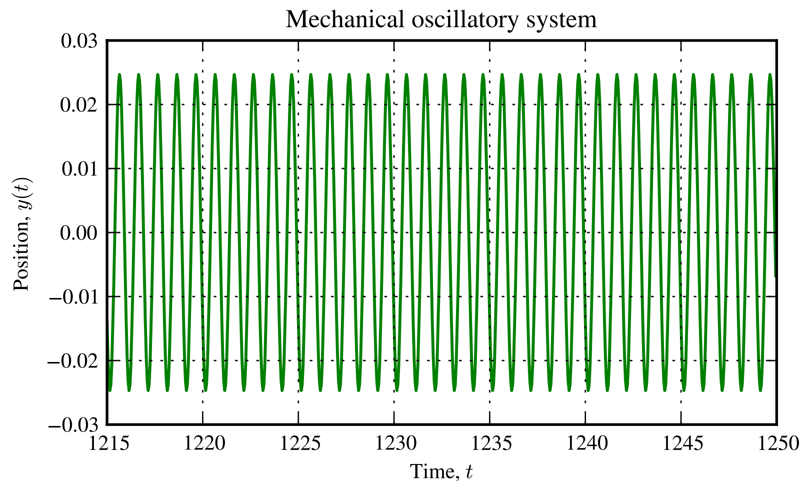


Figure 4.2: Position $y(t)$ versus time t for the mechanical oscillatory system example. The data used is: $m = 1.0$, $b = 1.7$, $c = 2.0$, $F_0 = 1.0$, $f = 1.0$, $Y_0 = 0.2$, $N = 25000$.

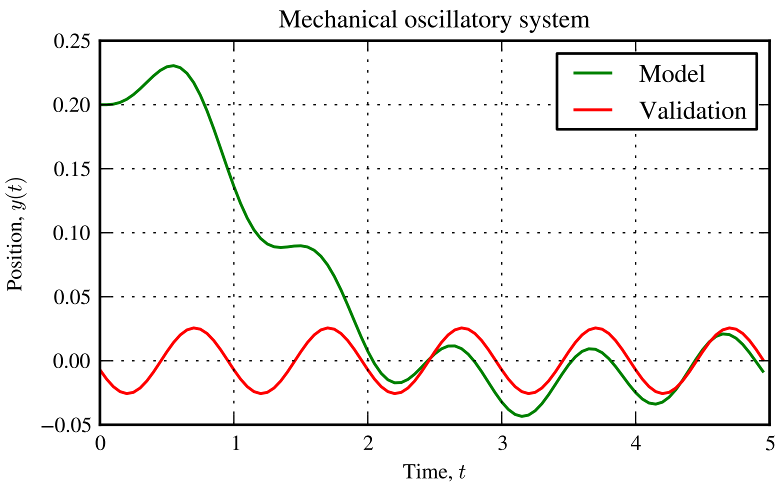


Figure 4.3: Position $y(t)$ versus time t for the mechanical oscillatory system example based on the model (green) and the predicted steady state solution (red). The data used is: $m = 1.0$, $b = 1.7$, $c = 2.0$, $F_0 = 1.0$, $f = 1.0$, $Y_0 = 0.2$, $N = 25000$.

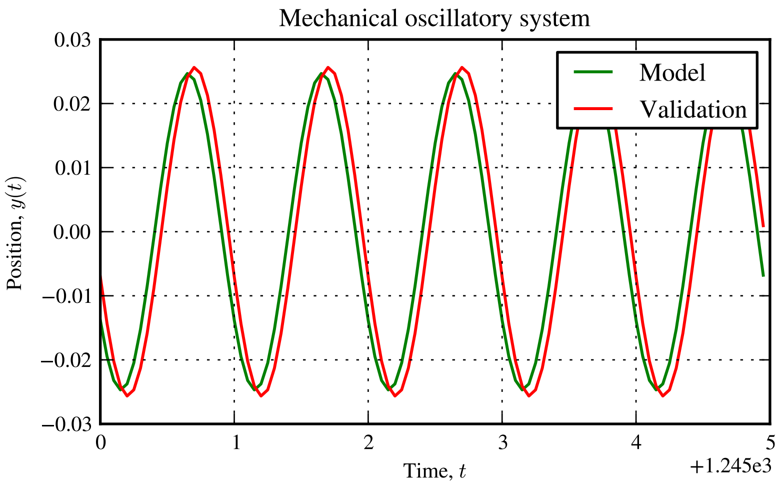


Figure 4.4: Position $y(t)$ versus time t for the mechanical oscillatory system example based on the model (green) and the predicted steady state solution (red). The data used is: $m = 1.0$, $b = 1.7$, $c = 2.0$, $F_0 = 1.0$, $f = 1.0$, $Y_0 = 0.2$, $N = 25000$.

Chapter 5

Exercises

5.1 Exercise 1

Let us try to dig a bit deeper into the example shown in chapter 1 on pages 2–5. Suppose that all the temperatures indicated in Table 1.1 are nominal values that are affected by $\pm 25\%$ uncertainties. In the two point method, check all combinations to determine the minimum and maximum number of years predicted to reach $t(y) = 7\,^{\circ}\mathrm{C}$.

5.2 Exercise 2

The estimation of A_{m} and f_{m} in (3.14) and (3.19), respectively, are reasonably robust as they rely on more than one possibly noise-affected sample. However, the estimation of the phase ϕ_{m} only uses a single sample to estimate the phase to be used in the model, which is obviously very noise sensitive. Can you come up with a better proposal of how to estimate the phase ϕ_{m} in (3.15) to yield a more accurate and robust result?

5.3 Exercise 3

Do one of the following (or both):

- Prove that the solutions for ψ and A in (4.22) and (4.23), respectively, are steady-state solutions to the driven damped oscillatory system described by (4.1).
- Derive the theoretical steady state solutions for A and ψ in (4.23) and (4.22), respectively, on page 27.

Bibliography

[1] J. Berry and K. Houston: *Mathematical Modeling.* Elsevier, 1995.

[2] Hans Petter Langtangen: *Python Scripting for Computational Science.* Springer, 2008.

[3] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery: *Numerical Recipes – The Art of Scientific Computing (Third Edition).* Cambridge University Press, 2007.

Appendix A

Python code for examples

This appendix contains the Python code used to generate the examples shown in Chapters 1 and 3-4.

A.1 Code for "Introductory example – the greenhouse effect"

```python
import numpy as np
import math
import operator
import matplotlib as mpl;   mpl.rcParams['backend'] = 'Agg'
import matplotlib.pyplot as mplpp
import datetime


#-----------------------------------------------------------------------------
try:
    mpl.rc('text', usetex=False)
    mpl.rc('font', family='serif')
    mpl.rc('font', serif='STIXGeneral')
    mpl.rc('font', size=8)
except AttributeError:
    None


#-----------------------------------------------------------------------------
def two_point_solution(y, t, points, t_prediction):
    (p0, p1) = points
    k = math.log10(t[p1]/t[p0]) / (y[p1] - y[p0])
    b = (y[p1]*math.log10(t[p0]) - y[p0]*math.log10(t[p1])) / (y[p1] - y[p0])

    # The first year where model temperature exceeds t_prediction
    y_prediction = np.int( (math.log10(t_prediction) - b) / k + 1.0)

    return k, b, y_prediction


#-----------------------------------------------------------------------------
def plot_data_lin(y, t):
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.plot(y, t, 'g-*', linewidth=1.5)
    mplpp.ylabel(r'Temperature rise, $t(y)\quad$ [${}^\circ$C]')
    mplpp.xlabel(r'Year, $y\quad$ [$-$]')
    mplpp.title(r'Rise in global temperature compared to 1860')
    mplpp.grid(True)
    mplpp.savefig('temp_year_lin.pdf', dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()


#-----------------------------------------------------------------------------
def plot_data_log(y, t):
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.semilogy(y, t, 'r-*', linewidth=1.5)
    mplpp.ylabel(r'Temperature rise, $t(y)\quad$ [${}^\circ$C]')
    mplpp.xlabel(r'Year, $y\quad$ [$-$]')
    mplpp.title(r'Rise in global temperature compared to 1860')
    mplpp.grid(True)
    mplpp.savefig('temp_year_log.pdf', dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()


#--------------------------------------------------------------------------------
def plot_data_model_log(y, t, (k, b), y_max):
    # Model
    y_model = y[0] + np.arange(y_max - y[0] + 1)
    t_model = 10**(k * y_model + b)

    # Plot
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.semilogy(y_model, t_model, 'g-',
                   y, t, 'r*',
                   y, linewidth=1.5)
    mplpp.axis([1880, 2100, 10**-2, 10**1])
    mplpp.ylabel(r'Temperature rise, $t(y)\quad$ [${}^\circ$C]')
    mplpp.xlabel(r'Year, $y\quad$ [$-$]')
    mplpp.title(r'Rise in global temperature compared to 1860')
    mplpp.legend([r'Model', r'Measured'], loc='upper left')
    mplpp.grid(True)
    mplpp.savefig('temp_year_model_log.pdf', dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()


#--------------------------------------------------------------------------------
if __name__ == '__main__':
    # DATA MATERIAL (Berry & Houston: mathematical Modelling, Elsevier, 1995)
    # y is the array of years and t is the corresponding array of temperatures
    # (in Celcius) above the 1860 level.
    y = np.array([1880.0, 1896.0, 1900.0, 1910.0, 1920.0, 1930.0,
                  1940.0, 1950.0, 1960.0, 1970.0, 1980.0])
    t = np.array([0.01, 0.02, 0.03, 0.04, 0.06, 0.08,
                  0.10, 0.13, 0.18, 0.24, 0.32])
    t_prediction = 7.0
    p0, p1 = 4, 10

    # Perform plotting and data analysis
    plot_data_lin(y, t)
    plot_data_log(y, t)
    k, b, y_prediction = two_point_solution(y, t, (p0, p1), t_prediction)
    print('Model > p0 = %i ; p1 = %i' % (p0, p1))
    print('Model > k = %6.3E ; b = %6.3E' % (k, b))
    print('In year %i the temperature rise exceeds %3.1f [C]' %
          (y_prediction, t_prediction))
    plot_data_model_log(y, t, (k, b), 2100)

#--- EOF ---------------------------------------------------------------------
```

A.2 Code for “Example: sinusoid”

```python
import numpy as np
import matplotlib as mpl;   mpl.rcParams['backend'] = 'Agg'
import matplotlib.pyplot as mplpp


#-----------------------------------------------------------------------------
try:
    mpl.rc('text', usetex=False)
    mpl.rc('font', family='serif')
    mpl.rc('font', serif='STIXGeneral')
    mpl.rc('font', size=8)
except AttributeError:
    None


#-----------------------------------------------------------------------------
def model(Am, fm, pm, T, N):
    tm = np.arange(N) * T / N
    dm = Am * np.sin(2*np.pi*fm*tm + pm)
    return dm, tm


#-----------------------------------------------------------------------------
def data(Am, fm, pm, T, N, sigma_d):
    d, t = model(Am, fm, pm, T, N)
    d += sigma_d * np.random.normal(size=(N))
    return d, t


#-----------------------------------------------------------------------------
def par_extraction(d, t):
    # Magnitude and phase extraction
    A = (np.max(d) - np.min(d)) / 2.0
    p = np.arcsin(d[0] / A)

    # Frequency extraction
    idx = []
    s = (d >= 0)
    for i in xrange(1, len(d)-2):
        if s[i-1] == s[i] and s[i+1] == s[i+2] and s[i] != s[i+1]:
            idx.append(i)
    f = 1.0 / (2.0*np.mean(np.array(t[idx[1:]]) - np.array(t[idx[:-1]])))

    return A, f, p

#-----------------------------------------------------------------------------
def plot_data(d, t, sigma_n, fname):
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.plot(t, d, 'g*', linewidth=1.5)
    mplpp.xlabel(r'Time, $t$')
    mplpp.ylabel(r'Data value, $d(t)$')
    st1 = r'Sinusoidal measurements; $\sigma_\mathrm{n}=%s$' % str(sigma_n)
    mplpp.title(st1)
    mplpp.grid(True)
    mplpp.savefig(fname, dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()


#-----------------------------------------------------------------------------
def plot_data_model(d, t, md, mt, sigma_n, fname):
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.plot(mt, md, 'r-', t, d, 'g*', linewidth=1.5)
    mplpp.xlabel(r'Time, $t$')
    mplpp.ylabel(r'Data value, $d(t)$')
    st1 = r'Sinusoidal measurements; $\sigma_\mathrm{n}=%s$' % str(sigma_n)
    mplpp.title(st1)
    st1 = r'Meas. data, $\sigma_\mathrm{n}=%s$' % str(sigma_n)
    mplpp.legend([r'Model data', st1], loc='upper right')
    mplpp.grid(True)
    mplpp.savefig(fname, dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()


#-----------------------------------------------------------------------------
if __name__ == '__main__':
    # Time interval
    T  = 5.0

    # Physical data and model data: noise free case 0
    N, M, A, f, p, sigma_n = 200, 4000, 1.0, 1.6, -1.4, 0.0
    d, t = data(A, f, p, T, N, sigma_n)
    plot_data(d, t, 0.0, 'meas_noisefree_0.pdf')
    Am, fm, pm = par_extraction(d, t)
    dm, tm = model(Am, fm, pm, T, M)
    plot_data_model(d, t, dm, tm, sigma_n, 'model_meas_noisefree_0.pdf')
    print(79*'-')
    print('Amplitude:  %8.2f  ;  %8.2f' % (A, Am))
    print('Frequency:  %8.2f  ;  %8.2f' % (f, fm))
    print('Phase:      %8.2f  ;  %8.2f' % (p, pm))
    print(79*'=')

    # Physical data and model data: noise free case 1
    N, M, A, f, p, sigma_n = 240, 4000, 1.0, 1.0, 0.0, 0.0
    d, t = data(A, f, p, T, N, sigma_n)
    plot_data(d, t, sigma_n, 'meas_noisefree_1.pdf')
    Am, fm, pm = par_extraction(d, t)
    dm, tm = model(Am, fm, pm, T, M)
    plot_data_model(d, t, dm, tm, sigma_n, 'model_meas_noisefree_1.pdf')
    print('Amplitude:  %8.2f  ;  %8.2f' % (A, Am))
    print('Frequency:  %8.2f  ;  %8.2f' % (f, fm))
    print('Phase:      %8.2f  ;  %8.2f' % (p, pm))
    print(79*'-')

    # Physical data and model data: noisy case 1
    N, M, A, f, p, sigma_n = 200, 2000, 1.0, 1.0, 0.0, 0.02
    d, t = data(A, f, p, T, N, sigma_n)
    plot_data(d, t, sigma_n, 'meas_noisy_1.pdf')
    Am, fm, pm = par_extraction(d, t)
    dm, tm = model(Am, fm, pm, T, M)
    plot_data_model(d, t, dm, tm, sigma_n, 'model_meas_noisy_1.pdf')
    print('Amplitude:  %8.2f  ;  %8.2f' % (A, Am))
    print('Frequency:  %8.2f  ;  %8.2f' % (f, fm))
    print('Phase:      %8.2f  ;  %8.2f' % (p, pm))
    print(79*'-')

    # Physical data and model data: noisy case 2
    N, M, A, f, p, sigma_n = 200, 2000, 0.8, 1.0, 0.2, 0.04
    d, t = data(A, f, p, T, N, sigma_n)
    plot_data(d, t, sigma_n, 'meas_noisy_2.pdf')
    Am, fm, pm = par_extraction(d, t)
    dm, tm = model(Am, fm, pm, T, M)
    plot_data_model(d, t, dm, tm, sigma_n, 'model_meas_noisy_2.pdf')
    print('Amplitude:  %8.2f  ;  %8.2f' % (A, Am))
    print('Frequency:  %8.2f  ;  %8.2f' % (f, fm))
    print('Phase:      %8.2f  ;  %8.2f' % (p, pm))
    print(79*'-')

    # Physical data and model data: noisy case 3
    N, M, A, f, p, sigma_n = 200, 2000, 0.9, 1.2, -0.2, 0.07
    d, t = data(A, f, p, T, N, sigma_n)
    plot_data(d, t, sigma_n, 'meas_noisy_3.pdf')
    Am, fm, pm = par_extraction(d, t)
    dm, tm = model(Am, fm, pm, T, M)
    plot_data_model(d, t, dm, tm, sigma_n, 'model_meas_noisy_3.pdf')
    print('Amplitude:  %8.2f  ;  %8.2f' % (A, Am))
    print('Frequency:  %8.2f  ;  %8.2f' % (f, fm))
    print('Phase:      %8.2f  ;  %8.2f' % (p, pm))
    print(79*'-')

#--- EOF ---------------------------------------------------------------------
```

A.3 Code for “Example: driven damped oscillatory system”

```python
import numpy as np
import matplotlib as mpl;   mpl.rcParams['backend'] = 'Agg'
import matplotlib.pyplot as mplpp


#-----------------------------------------------------------------------------
try:
    mpl.rc('text', usetex=False)
    mpl.rc('font', family='serif')
    mpl.rc('font', serif='STIXGeneral')
    mpl.rc('font', size=8)
except AttributeError:
    None


#-----------------------------------------------------------------------------
def model(m, b, c, F_0, f, Y_0, N, t_delta):
    """
    Returns y and t representing the position versus time. Both y and t
    are vectors of length N.

    """
    # Constants
    alpha = m + b * t_delta + c * t_delta**2
    beta = 2 * m + b * t_delta
    Omega = 2.0 * np.pi * f * t_delta

    # Damping factor and oscillation frequency
    zeta = b / (2 * np.sqrt(m * c))
    f_0 = np.sqrt(c / m) / (2 * np.pi)
    print('Damping factor, zeta = %.2f ; natural frequency, f_0 = %.2f'
          % (zeta, f_0))

    # Initialization
    y, t = np.empty(N), np.arange(N) * t_delta
    y[0] = Y_0
    y[1] = Y_0

    # Compute position versus time
    for n in xrange(2, N):
        y[n] = ( F_0*t_delta**2/alpha * np.sin(Omega*n)
                + beta/alpha*y[n-1]
                - m/alpha*y[n-2] )
    return y, t


#-----------------------------------------------------------------------------
def validation(m, b, c, F_0, f, Y_0, N, t_delta):
    """
    Returns y and t representing the position versus time when using
    closed form derived expressions. Both y and t are vectors of length N.

    """
    # Initial
    omega = 2*np.pi*f
    zeta = b / (2 * np.sqrt(m * c))
    omega_0 = np.sqrt(c / m)
    t = np.arange(N) * t_delta

    # Steady state part of solution
    psi = np.arctan(b*omega / (omega**2 * m - c))
    A = -F_0 / (b*omega) * np.sin(psi)
    y_ss = A * np.sin(omega * t + psi)

    return y_ss, t


#-----------------------------------------------------------------------------
def plot_data(y, t, points, fname):
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.plot(t[-points:], y[-points:], 'g', linewidth=1.1)
    mplpp.xlabel(r'Time, $t$')
    mplpp.ylabel(r'Position, $y(t)$')
    mplpp.title('Mechanical oscillatory system')
    mplpp.grid(True)
    mplpp.savefig(fname, dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()



#---------------------------------------------------------------------------
def plot_data_2((y, t), (y_v, t_v), points, fname):
    mplpp.figure(1, (4.50, 2.50))
    mplpp.clf()
    mplpp.plot(t[-points:], y[-points:], 'g',
               t_v[-points:], y_v[-points:], 'r',
               linewidth=1.1)
    mplpp.legend([r'Model', r'Validation'], loc='upper right')
    mplpp.xlabel(r'Time, $t$')
    mplpp.ylabel(r'Position, $y(t)$')
    mplpp.title('Mechanical oscillatory system')
    mplpp.grid(True)
    mplpp.savefig(fname, dpi=1200,
                  bbox_inches='tight', pad_inches=0.05)
    mplpp.close()

#---------------------------------------------------------------------------
if __name__ == '__main__':
    # Model parameters
    m = 1.0
    b = 1.7
    c = 2.0
    F_0 = 1.0
    f = 1.0
    Y_0 = 0.2
    N1 = 700
    N2 = 25000
    points1 = N1
    points2 = 100

    # Compute sampling time
    t_delta = 1.0 / (20.0 * f)
    y, t = model(m, b, c, F_0, f, Y_0, N1, t_delta)
    plot_data(y, t, points1, 'y_vs_t_N700.pdf')
    y, t = model(m, b, c, F_0, f, Y_0, N2, t_delta)
    plot_data(y, t, points1, 'y_vs_t_N2000.pdf')

    # Validation
    y_v, t_v = validation(m, b, c, F_0, f, Y_0, N2, t_delta)
    plot_data(y_v, t_v, points1, 'y_vs_t_N700_validation.pdf')
    plot_data_2((y[:points2], t[:points2]),
                (y_v[:points2], t_v[:points2]),
                points2, 'compare_1.pdf')
    plot_data_2((y, t), (y_v, t_v), points2, 'compare_2.pdf')

#--- EOF -------------------------------------------------------------------
```