Written by Hannes Reindl, 11.09.2023

1 Introduction

The goal is to fit an exponential function in the form of

$$f(a,b,c,\tau) = a - b \cdot e^{-\frac{(t-c)}{\tau}} \tag{1}$$

with the parameters a, b, c, τ to some given measurements $\tilde{\boldsymbol{d}}$.

$$\tilde{\boldsymbol{d}} = \begin{bmatrix} \tilde{d}_0 \\ \tilde{d}_1 \\ \vdots \end{bmatrix} \tag{2}$$

The figure below depicts an exemplary fitted exponential to some data $\tilde{\boldsymbol{d}}$, which is the goal. To

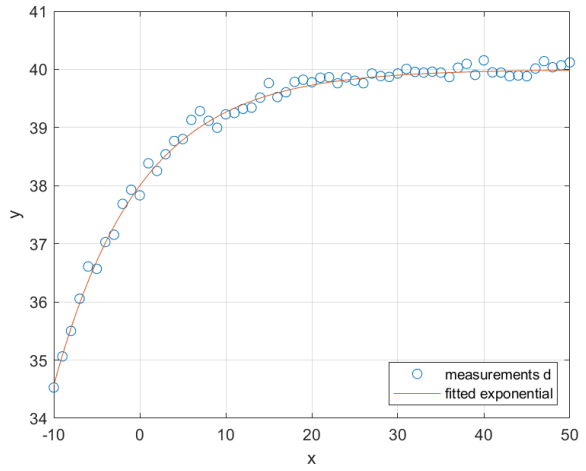

Figure 1: Exponential in the form of $f = a - b \cdot e^{-\frac{(t-c)}{\tau}}$ fitted to some noisy measurements $\tilde{d}$.

get a feeling for these parameters use GeoGebra: `https://www.geogebra.org/classic?lang=de-AT` and enter the function f there.

2 First steps

Getting some fundamental insights is easier by looking at a simplified problem. Therefore the function to fit will be boiled down to only one changing variable. To put this concretely, the new function is

$$f(a) = a - b \cdot e^{-\frac{(t-c)}{\tau}} \tag{3}$$

where b and τ are known values (e.g. $b = 2, c = 0, \tau = 10$).

2.1 Defining a cost function to minimize

The problem at hand can be rewritten to minimize the cost function J,

$$J = \sum_{i=0}^{N-1} \left(\tilde{d}_i - (a - b \cdot e^{-\frac{t_i - c}{\tau}}) \right)^2 \tag{4}$$

$$= \sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}} \right)^2 \tag{5}$$

where $\tilde{d}_i$ is the i-th measurement taken at the time instance t_i. The cost function J reaches its minimum if the function $(a - b \cdot e^{-\frac{t_i - c}{\tau}})$ is the exact same as $\tilde{d}_i$ for all time instances t_i, thus making it a valid cost function. To simplify things, we assume that the variables b, c and τ are known, reducing the problem to one dimension. In this case, the values for b, c and τ were set to $b = 2$, $c = 0$ and $\tau = 10$. Using the measurements $\tilde{d}$ from Fig. 1 and varying the parameter a results in the following figure for the cost function J. The optimal parameter

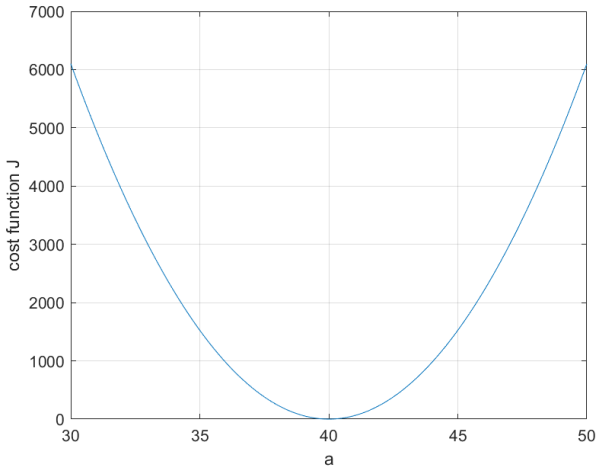

Figure 2: Cost function J over the variable a, assuming that the variables b, c and τ are known.

for a using the data from Fig 1 is $a = 40$. This could also be deducted by simply looking at the data since an exponential in the form $1 - e^{-x}$ never surpasses 1 and the data in Fig. 1 also stops at 40. It does surpass 40 but only because of the additive noise inside the measurements $\tilde{d}$. This can be solved analytically by deriving the cost function J w.r.t. a.

$$\frac{dJ}{da} = \frac{d}{da} \left(\sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}} \right)^2 \right) \tag{6}$$

The differentiation quotient can be pulled inside the sum using the sum rule.

$$\frac{dJ}{da} = \sum_{i=0}^{N-1} \frac{d}{da} \left(\left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}} \right)^2 \right) \tag{7}$$

Applying the chain rule:

$$\frac{dJ}{da} = \sum_{i=0}^{N-1} 2 \cdot \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}}\right) \cdot \frac{d}{da}(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}}) \tag{8}$$

$$= \sum_{i=0}^{N-1} 2 \cdot \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}}\right) \cdot (-1) \tag{9}$$

The first derivative w.r.t. a is:

$$\frac{dJ}{da} = -2 \cdot \sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}}\right) \tag{10}$$

Plotting the derivation alongside the cost function:

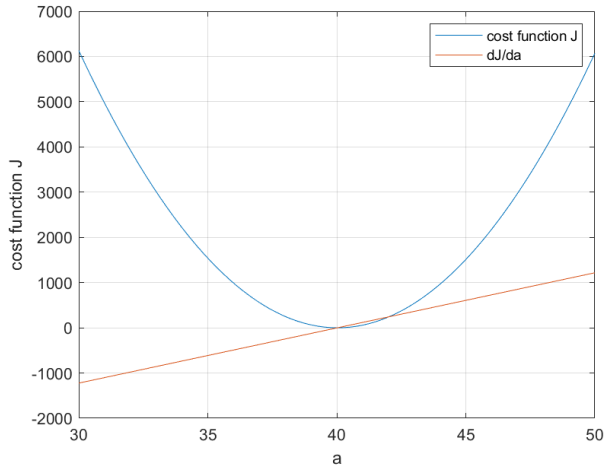

Figure 3: Cost function J and its derivation w.r.t a over the variable a

Using the age-old handbook trick of setting the first derivative to zero, results in an optima for the given cost function J.

$$0 \stackrel{!}{=} -2 \cdot \sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}}\right) \tag{11}$$

Getting rid of the 2 and splitting up the sum.

$$0 \stackrel{!}{=} \sum_{i=0}^{N-1} \left(\tilde{d}_i\right) + \sum_{i=0}^{N-1} (-a) + \sum_{i=0}^{N-1} \left(b \cdot e^{-\frac{t_i - c}{\tau}}\right) \tag{12}$$

The sum over $-a$ can be simplified since a does not depend on i. Also b can be pulled out of the sum.

$$0 \stackrel{!}{=} \sum_{i=0}^{N-1} \left(\tilde{d}_i\right) - N \cdot a + b \cdot \sum_{i=0}^{N-1} \left(e^{-\frac{t_i - c}{\tau}}\right) \tag{13}$$

Rearranging things results in an analytic formula to find the optimal value for a.

$$a = \frac{1}{N}\left(\sum_{i=0}^{N-1} \left(\tilde{d}_i\right) + b \cdot \sum_{i=0}^{N-1} \left(e^{-\frac{t_i - c}{\tau}}\right)\right) \tag{14}$$

As a last step, we would have to prove, that this is indeed a minimum and not a maximum by either using the second derivative or some other means. We will skip this here, since from Fig. 3 it is clear, that this is in fact a minimum.

Unfortunately, this nice analytic derivation is not possible if there are several parameters to be determined, which is why Gradient Descent will be introduced now.

2.2 Introducing Gradient Descent

Gradient Descent is an iterative approach to finding a minimum. It uses the gradient w.r.t. u ∇_u, where u is a vector containing all parameters. In this case, the entries of u are all parameters we would like to know of equation 1.

$$\boldsymbol{u} = \begin{bmatrix} a \\ b \\ c \\ \tau \end{bmatrix} \tag{15}$$

and ∇_u the derivations for all parameters.

$$\boldsymbol{\nabla_u} = \begin{bmatrix} \frac{d}{da} \\ \frac{d}{db} \\ \frac{d}{dc} \\ \frac{d}{d\tau} \end{bmatrix} \tag{16}$$

The gradient w.r.t. u applied to cost function J results in the vector:

$$\boldsymbol{\nabla_u} J = \begin{bmatrix} \frac{dJ}{da} \\ \frac{dJ}{db} \\ \frac{dJ}{dc} \\ \frac{dJ}{d\tau} \end{bmatrix} \tag{17}$$

The gradient of any function points towards the steepest uphill direction, which means the negative gradient points towards the steepest downhill direction. This property is ideal for going downwards the valley i.e. to the minimum of the cost function. Using this information, the iterative equation of gradient descent can be written as,

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n - \gamma \cdot \boldsymbol{\nabla_u} J \tag{18}$$

where γ is the step size parameter and $\gamma > 0$. If $\gamma < 0$, the algorithm moves in the uphill direction i.e. it could not converge towards a minimum. The main idea of the Gradient Descent is, to pick a starting location $\boldsymbol{u}_0$, evaluate the gradient $\boldsymbol{\nabla_u} J$ at the point $\boldsymbol{u}_0$ and make a step towards the steepest descent scaled by the step size γ, ending up in $\boldsymbol{u}_1$. In the next iteration step, the gradient in the point $\boldsymbol{u}_1$ is evaluated and the cycle continues until some convergence criterion is met. Exemplary convergence criteria are:

- the maximum amount of iterations is met
- ΔJ over five iterations is less than a certain threshold

Simplifying things and assuming that all parameters but a are known, meaning that parameter vector $\boldsymbol{u}$ only depends on a

$$\boldsymbol{u} = [a] \tag{19}$$

results in the gradient

$$\boldsymbol{\nabla}_{\boldsymbol{u}} = \left[\frac{dJ}{da}\right] \tag{20}$$

, which was already calculated in the previous section. Therefore the Gradient Descent update equation

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n - \gamma \cdot \boldsymbol{\nabla}_{\boldsymbol{u}} \tag{21}$$

can be written as a simple scalar function since $\boldsymbol{u}$ only has the parameters a as an entry.

$$a_{n+1} = a_n - \gamma \cdot \frac{dJ}{da} \tag{22}$$

Figure 4 shows the points along which the gradient descent algorithm traveled for the simplified problem. Note

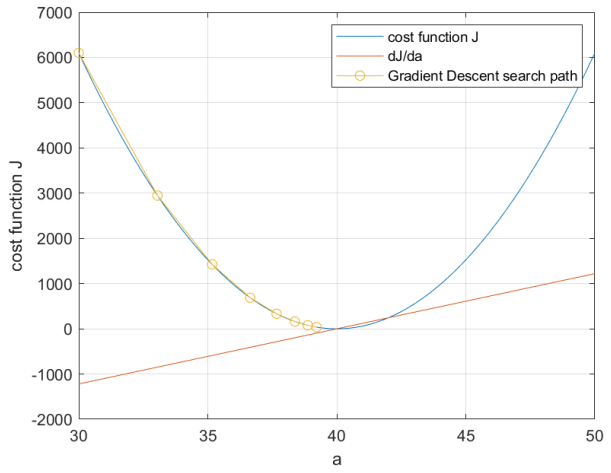

Figure 4: Path taken from the Gradient Descent algorithm for 7 iteration steps.

that the steps towards the minima are initially bigger because the first derivative of $\frac{dJ}{da}$is bigger, further away from the minima. Looking at equation 18 clarifies this. The magnitude of the step taken ($\gamma \cdot \boldsymbol{\nabla}_{\boldsymbol{u}} J$) depends on the length of the first vector $\nabla_{\boldsymbol{u}} J$. In some cases, this can be unwanted due to extremely high values of the first derivative. Figure 4 used for $\gamma = 0.002$. Increasing it to only $\gamma = 0.014$ results in Fig. 5, depicted below. The solution does still converge towards the minima, but it oscillates around it. But an increase to $\gamma = 0.018$ already causes it to diverge (see Fig. 6).
This behavior is often unwanted and the step size γ has rather arbitrary values. To solve this, a new Gradient Descent equation will be introduced, which normalizes the gradient by its length:

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n - \gamma \cdot \frac{\boldsymbol{\nabla}_{\boldsymbol{u}} J}{||\boldsymbol{\nabla}_{\boldsymbol{u}} J||} \tag{23}$$

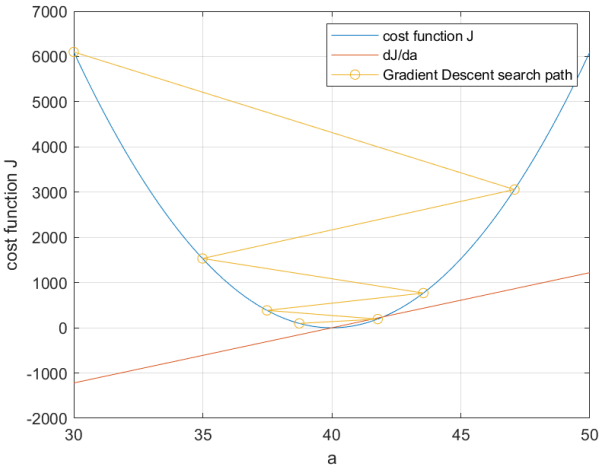

Figure 5: Path taken from the Gradient Descent using a bigger step size γ.

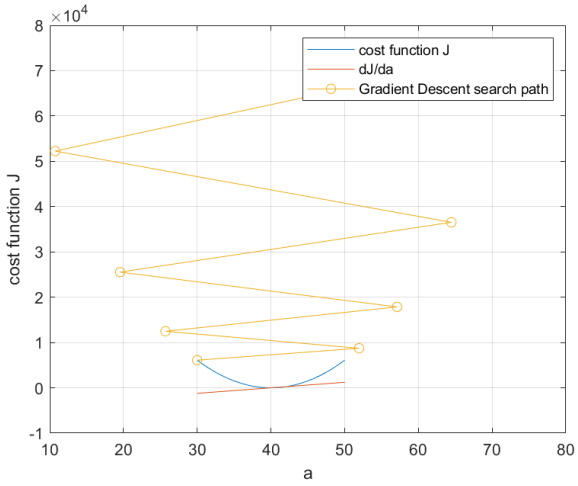

Figure 6: Gradient descent converges if the step size γ is too big.

The norm operator $||\,.\,||$ ensures, that the gradient is normalized to 1, meaning that the magnitude of the step is solely controlled by the step size parameter γ. Figure 7 displays the behavior of the Gradient Descent equation with normalization and γ set to $\gamma = 2$. All steps have the same magnitude of 2 in the x-direction, which is expected given the new update equation 23.

2.3 What are the benefits of Gradient Descent

Until now it was possible to find the minima by simply looking at the graph of the cost function, but if the function depends on several parameters, the resulting plot would be higher dimensional which can not be grasped by us beings stuck in three dimensions. But Gradient Descent has no problem with more dimensions

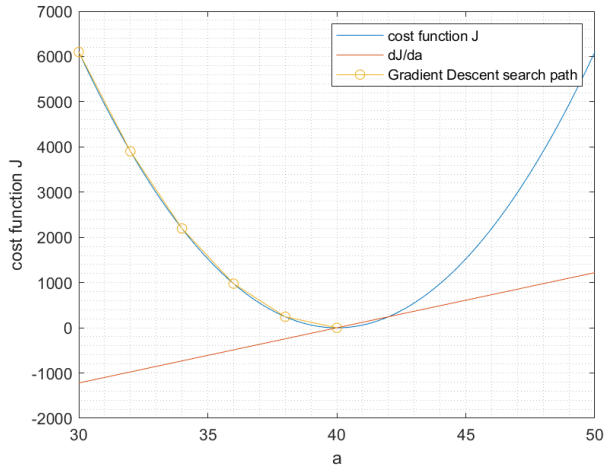

Figure 7: Gradient descent converges if the step size γ is too big.

since it only sees an array of numbers and combines them according to the algorithm.

2.4 Applying Gradient Descent to the initial problem

The initial problem was, finding the parameters a, b, c, τ of the function

$$f(a,b,c,\tau) = a - b \cdot e^{-\frac{(t-c)}{\tau}} \tag{24}$$

which fits the measurements $\tilde{\boldsymbol{d}}$ best (see Fig. 1).
Using the normalized update equation for Gradient Descent

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n - \gamma \cdot \frac{\boldsymbol{\nabla}_{\boldsymbol{u}} J}{||\boldsymbol{\nabla}_{\boldsymbol{u}} J||} \tag{25}$$

, with the variables

$$\boldsymbol{u} = \begin{bmatrix} a_n \\ b_n \\ c_n \\ \tau_n \end{bmatrix}; \qquad \boldsymbol{\nabla}_{\boldsymbol{u}} J = \begin{bmatrix} \frac{dJ}{da} \\ \frac{dJ}{db} \\ \frac{dJ}{dc} \\ \frac{dJ}{d\tau} \end{bmatrix}. \tag{26}$$

The derivations w.r.t. to the different parameters of the cost function J

$$J = \sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}} \right)^2 \tag{27}$$

were calculated using the online tool `ableitungsrechner.net` found here. The sum can not be expressed using this tool, but it is no problem since the derivative can be pulled inside the sum. The derivation variable can be changed in the "Options" tab. The derivations required for the gradient are:

$$\frac{dJ}{da} = -2\left(-a + b\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} + d_\mathrm{i}\right) \tag{28}$$

$$\frac{dJ}{db} = 2\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} \cdot \left(\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} b + d_\mathrm{i} - a\right) \tag{29}$$

$$\frac{dJ}{dc} = \frac{2b\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} \cdot \left(b\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} + d_\mathrm{i} - a\right)}{\tau} \tag{30}$$

$$\frac{dJ}{d\tau} = \frac{2b \cdot (t_\mathrm{i} - c)\,\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} \cdot \left(b\mathrm{e}^{-\frac{t_\mathrm{i}-c}{\tau}} + d_\mathrm{i} - a\right)}{\tau^2} \tag{31}$$

With these derivatives, all information required to fit the exponential is given. To test this algorithm some artificial data using equation 24 was created. Furthermore, some Gaussian and rounding noise was added on top. Figure 8 and 9 show the measurements as well as the fitted exponential. The true coefficients for the exponential

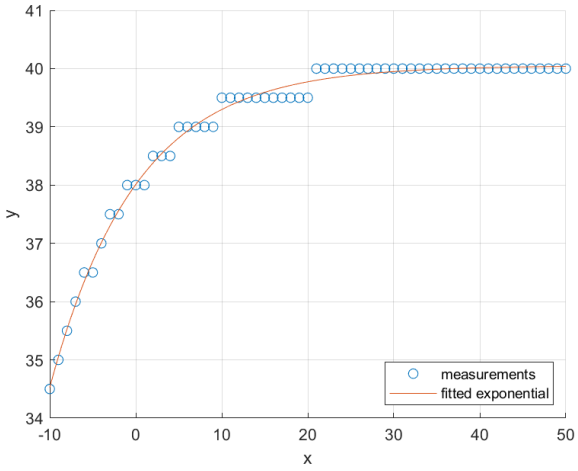

Figure 8: Exponential function fitted to some data with additive Gaussian noise.

function from which the measurements were generated are:

$$\boldsymbol{u}_{true} = \begin{bmatrix} 40 \\ 2 \\ 0 \\ 10 \end{bmatrix} \tag{32}$$

with the starting point

$$\boldsymbol{u}_{true} = \begin{bmatrix} 30 \\ 1 \\ 2 \\ 5 \end{bmatrix} \tag{33}$$

, which are quite close to the correct values but I argue, that with some trial and error, a good starting point is always possible to find.

The Gradient Descent algorithm calculated these values for the Gaussian additive noise example.

$$\boldsymbol{u}_{gaussian} = \begin{bmatrix} 40.0517 \\ 2.0409 \\ 0.1023 \\ 10.0412 \end{bmatrix} \tag{34}$$

And for the rounding noise example:

$$\boldsymbol{u}_{rounding} = \begin{bmatrix} 40.1106 \\ 1.9094 \\ -0.1521 \\ 9.1683 \end{bmatrix} \tag{35}$$

Both solutions are extremely close to the correct exponential, showing that Gradient Descent is a promising tool in this scenario.

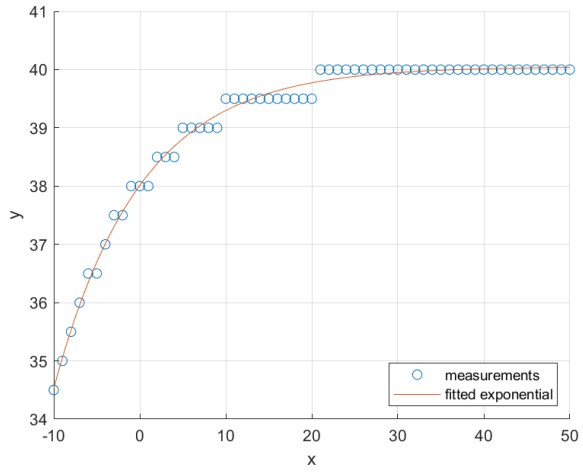

Figure 9: Exponential function fitted to some data with rounding noise.

2.5 Using an adaptive step size

The step size γ was held static until now, but is sensible to adaptively change it over the iterations. For example, the step size can be multiplied by a factor c in each iteration

$$\gamma_{n+1} = c \cdot \gamma \tag{36}$$

If c is smaller than one, (e.g. 0.99), the step size gets smaller and smaller, resulting in a better minima. Note that there are way better methods and this only provides a small reminder that this adaptive change exists and is useful.

2.6 Appendix: Alternative method besides using the normalized update equation of Gradient Descent

This is purely for the sake of documenting some cool information/approach for optimization algorithms. While the initial update equation had problems with convergence due to the total step taken in each iteration depending directly on the magnitude of the gradient, another solution could be to change the cost function. Have a look at the following cost function.

$$J_a = 10 + \sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}} \right)^2 \tag{37}$$

This is the exact same cost function as displayed in Fig. 2 but shifted 10 in the y-axis. Minimizing this function would still result in the same minimum because the entire plot is simply shifted up. The location of the minimum w.r.t. to the x-axis i.e. the wanted parameter did **not** change (see. Fig 10).

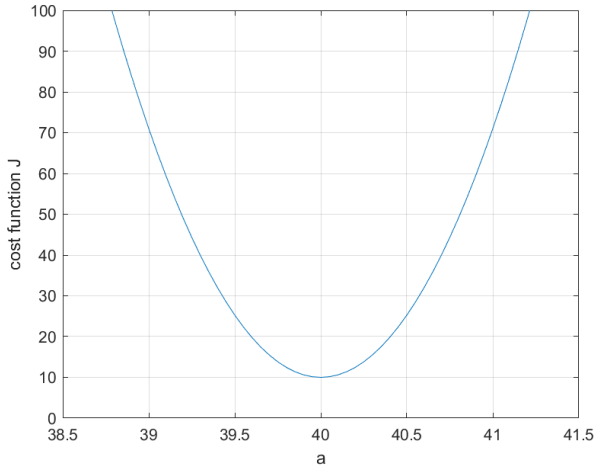

Figure 10: The location of the minima for the altered cost function J did not change.

Now have a look at this cost function L, which simply takes the natural logarithm of the unaltered cost function J.

$$L = \ln(J) = \ln\left(\sum_{i=0}^{N-1} \left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}} \right)^2 \right) \tag{38}$$

The plot of the resulting cost function w.r.t. can be seen in Fig. 11 The logarithm is a strict monotonic increasing function i.e. if $x < y$, then $f(x) < f(y)$. This means that the smallest value of the entire function space before sending it through the logarithm is still the smallest value after the logarithm. This means the logarithm does preserve the location of the minima (i.e. position on the x-axis). This property also holds true for n-dimensional problems. Looking at the gradient of the new cost function L. the values are rather small compared to the gradient of J (see Fig. 3). Only near the optimum, the gradient descends toward negative infinity, because near the optima the cost function J evaluates to $J = 0$ and $L = \ln(J) = \ln(0) = -\infty$. To overcome this problem, we

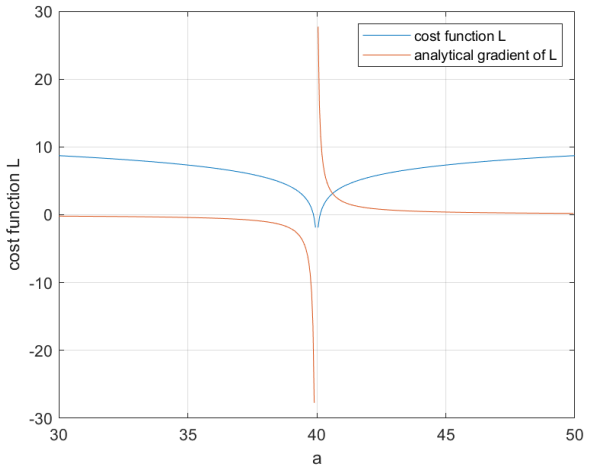

Figure 11: The location of the minima is even preserved through the natural logarithm.

must ensure that the expression inside the ln() function can never be 0, which was done by adding 10. Introducing the final cost function K:

$$K = \ln(J_a) \tag{39}$$
$$= \ln(10 + J) \tag{40}$$
$$= \ln\left(10 + \sum_{i=0}^{N-1}\left(\tilde{d}_i - a + b \cdot e^{-\frac{t_i - c}{\tau}}\right)^2\right) \tag{41}$$

which has the following graph (Fig. 12) The gradient of

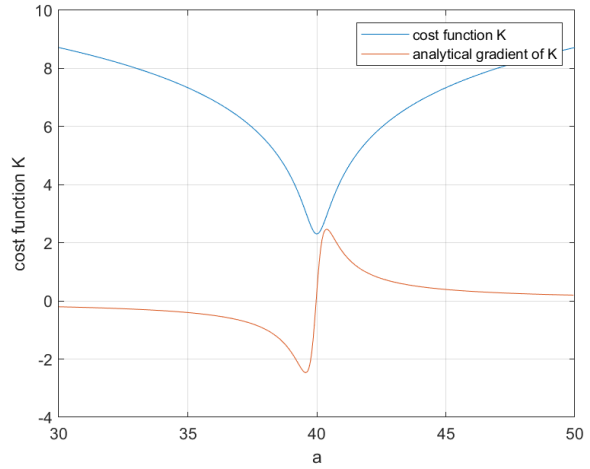

Figure 12: Cost function K and its gradient over the support.

K is now bounded between 2 and -2, which results in a much more stable gradient descent algorithm. It should be noted that this approach does work, but it is not optimal. If the starting point is far from optima, the small gradient causes really small steps toward the minima using the normal Gradient Descent update equation.

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n - \gamma \cdot \boldsymbol{\nabla}_{\boldsymbol{u}} J \tag{42}$$

I do not recommend using this approach, the sole purpose of this appendix is, to show some nifty tricks. With this said, this documentation comes to an end and I can finally start to work on something else.