# Newton's method and variants used in nonlinear least squares regression


by

Kenneth Rory Kolster (SNR: 2070539)


A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor in Econometrics and Operations Research


Tilburg School of Economics and Management
Tilburg University

Supervised by: prof. dr. Etienne de Klerk


Date: April 26, 2024

# Abstract

The Newton method is an optimization technique used to find minima of real-valued functions that are twice continuously differentiable. It is an iterative algorithm that approximates the roots of the derivative if the method successfully converges to a minimizer. The Newton method has favourable quadratic convergence rates under specific conditions, but it also has notable drawbacks. The Gauss-Newton method is a variant of the Newton method where a term in the Hessian of the objective function is neglected. Nonlinear Least Squares (NLS) regression aims to fit a model to a set of data by minimizing their residuals. Hence, NLS regression is a common area for the Newton and Gauss-Newton methods to be used in. To examine performance of these methods, I apply them in NLS regression, fitting a logistic function model to some sets of COVID-19 cumulative deaths data. These data nicely exhibit the sigmoid curvature, allowing for some convergence results and comparison between the two methods.

# Contents

# Chapter 1

# Introduction

## 1.1 Nonlinear Least Squares

Nonlinear Least Squares (NLS) is a type of optimization technique used in regression analysis where models contain nonlinear coefficients. We follow the notation as used in Nocedal and Wright (2006). Consider the following model:

$$y = \phi(x; t) + \varepsilon \tag{1.1}$$

where $\phi(x; t)$ is a nonlinear function with independent variable $t$, coefficients $x$ that need to be estimated, and $\varepsilon$ is the error term. For $f$ to be nonlinear, the coefficients must appear as nonlinear functions. Nonlinear forms include polynomial: $y = x_0 + x_1 t + x_2^2 t^2 + \varepsilon$, multiplicative: $y = x_0 + x_1 \cdot x_2 t + \varepsilon$, or exponential: $y = e^{x_0 + x_1 t} + \varepsilon$, and many more.

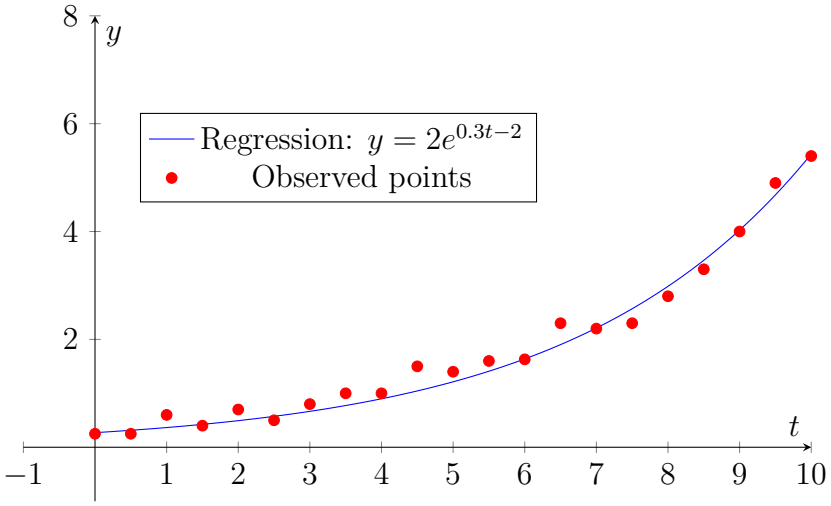


Figure 1.1: Nonlinear regression example

Nonlinear Least Squares continues in a similar fashion to the well-known Ordinary Least Squares (OLS). We define the following terms:

$$\begin{aligned} (t_j, y_j) \text{ for } j = 1, \ldots, m & \qquad \text{(Observations)} \\ r_j(x) = y_j - \phi(x; t_j) & \qquad \text{(Residuals)} \\ f(x) = \frac{1}{2} \sum_{j}^{m} r_j^2(x). & \qquad \text{(Objective function)} \end{aligned}$$

In this setup, $m$ is the number of observations, and there are coefficients $x_i$, with $i = 1, \ldots, n$. The objective function is sometimes called the sum of squares. The $\frac{1}{2}$ does not affect the minimization, but is often included for mathematical convenience. We proceed by minimizing the objective function with respect to the $x$ coefficients: $\min_x f(x)$. In the case of OLS, we would normally take the derivative and set it to zero in order to determine the minimum, where we would obtain the well-known OLS closed-form estimate. However, in our case we would obtain a nonlinear system of equations where we cannot easily find a closed-form solution. Instead, there are various ways to numerically determine the minimum and for this paper we look at Newton's method as follows in the next Section.

## 1.2 Newton's method and variants

Newton's method was initially developed as an iterative method to numerically find solutions to polynomials, but since then has been applied to a wide variety of problems. Here I show briefly the derivation for Newton's method for root finding. The concept starts with an initial guess, $x^{(0)}$, of the root, where the tangent to the curve at $x^{(0)}$ is followed to the intercept at the x-axis, which gives the next guess. Let $x^{(k)}$ be an arbitrary iterate in the method and let $x^{(k+1)}$ be the x-intercept of the tangent at $x = x^{(k)}$. This can be shown in the following figure:

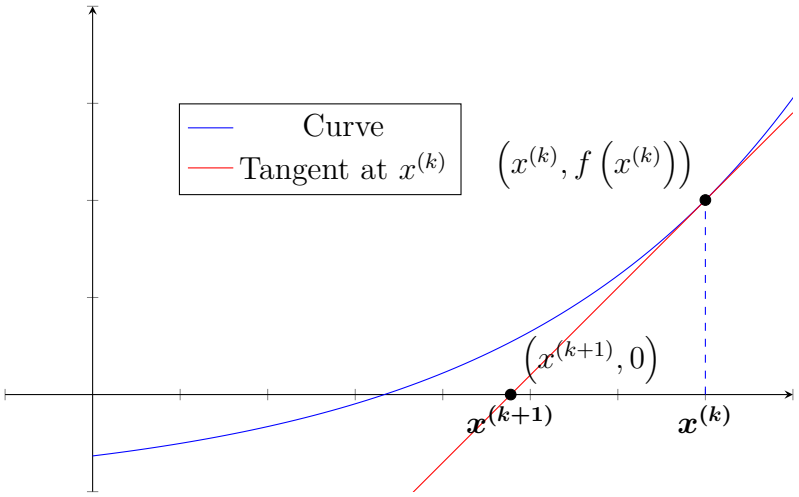


Figure 1.2: Newton's method

Then the slope can be given below and then solved for $x^{(k+1)}$:

$$f'\left(x^{(k)}\right) = \frac{f\left(x^{(k)}\right) - 0}{x^{(k)} - x^{(k+1)}} \tag{1.2}$$

$$\implies x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}. \tag{1.3}$$

The general idea for Newton's method in an optimization (minimization) problem is to use the second-order Taylor expansion of the function $f$, namely the quadratic approximation:

$$f(x^{(k)} + p) \approx f\left(x^{(k)}\right) + f'\left(x^{(k)}\right) p + \frac{1}{2} f''\left(x^{(k)}\right) p^2. \tag{1.4}$$

We can gain the next iterate $x^{(k+1)}$ by minimizing the approximation of $f$ in $t$ around $x^{(k)}$, and setting $x^{(k+1)} = x^{(k)} + p$. Provided we have a convex function where the second derivative is positive, we can set the first derivative of the right-hand side of (1.4) to zero and solve to obtain the $p$ for which the $f$ is minimal:

$$0 = \frac{\partial}{\partial p}\left(f\left(x^{(k)}\right) + f'\left(x^{(k)}\right) p + \frac{1}{2} f''\left(x^{(k)}\right) p^2\right) \tag{1.5}$$

$$\implies 0 = f'\left(x^{(k)}\right) + f''\left(x^{(k)}\right) p \tag{1.6}$$

$$\implies p = -\frac{f'\left(x^{(k)}\right)}{f''\left(x^{(k)}\right)} \tag{1.7}$$

$$\implies x^{(k+1)} = x^{(k)} - \frac{f'\left(x^{(k)}\right)}{f''\left(x^{(k)}\right)}. \tag{1.8}$$

This is the same as Newton's method for root finding, since we are only using one order of derivative higher. The case above exists in the setting for

univariate functions where $f : \mathbb{R} \to \mathbb{R}$. The method can be readily generalized to multivariate cases, like the one described in Section 1.1. Let $f$ be a function such that $f : \mathbb{R}^n \to \mathbb{R}$. We then use the gradient vector $\nabla f(x)$ and the Hessian matrix $\nabla^2 f(x)$ evaluated at $x$, where

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix}, \text{ and } \nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n \partial x_n} \end{pmatrix}. \tag{1.9}$$

Hence, $\nabla f(x) \in \mathbb{R}^n$ and $\nabla^2 f(x) \in \mathbb{R}^{n \times n}$. Analogously to the univariate case, by setting the second-order Taylor approximation of $f$ to zero, we yield the next iterate $x^{(k+1)} \in \mathbb{R}^n$:

$$x^{(k+1)} = x^{(k)} - \left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1} \nabla f\left(x^{(k)}\right). \tag{1.10}$$

In its original state, the Newton method is easy to understand and easy to apply. However, it also has its drawbacks, which we discuss in Chapter 2. Hence, there are variants and extensions of the method, for instance the Gauss-Newton method, the Levenberg-Marquardt algorithm, and Cubic Regularization and so forth.

## 1.3 Scope of dissertation

The scope of this thesis revolves mainly around the Newton's method and its use in practical applications, namely nonlinear least squares regression. I begin with developing the mathematical understanding, and then move on to the application, using it to ultimately replicate a logistic growth model for COVID-19 cumulative deaths.
In this thesis, I will first look at the Newton's method applied in optimization. In Section 2.1, I will briefly focus on one convergence result and the method's drawbacks and caveats, where I consider primarily cases of convex optimization. Then, in Section 2.2, I explore the Gauss-Newton variant used in Nonlinear Least Squares regression.
Thereafter, I will link the mathematics covered to the case of growth models, providing motivation as to why the methods are well equipped to solve these cases of NLS regression. Next I collect data of COVID-19 cumulative deaths and apply NLS regression using both the Newton and Gauss-Newton method to replicate the logistic curves commonly seen in these data sets. These models are highly relevant as they are frequently used for policy planning, both at a national and global level.

Newton's method is a long-standing area of research in the field of optimization but due to its fundamental nature, it is not out-dated. The field receives many relevant developments even in the last few decades. For instance, there are papers like Ahmadi et al., 2023 which have incorporated a

*sum of squares-convex approximation* of the $d^{th}$-order Taylor expansion into the Newton method, while keeping the cost per iteration at polynomial level. Nesterov and Polyak, 2006 introduces a cubic term in the Newton step which helps the algorithm converge faster and more robustly. It does this by regularizing the Newton step and reducing the chance of overshooting or diverging. Mishchenko, 2023 builds on this concept by fusing the *cubic regularization* with an adaptive Levenberg-Marquadt penalty, providing a global convergence result with $\boldsymbol{\mathcal{O}(1/k^2)}$ rate.

Most adaptations of Newton-style methods in the last century focus on concepts like *line search* (damped Newton method), *trust-region* approaches, and *Levenberg-Marquadt regularization*, amongst some others.[1] Due to the current age of vast amounts of data and complexity of models ever-increasing, the advantage of approximate methods like Gauss-Newton is also increasing. Frequent research into these methods can be highlighted by papers like Gratton et al., 2007 where they study *truncated* and *perturbed* Gauss-Newton methods for nonlinear least squares problems.

[1]For papers on the given topics see: Hanzely et al., 2022, Lin et al., 2008, Levenberg, 1944, Marquardt, 1963

# Chapter 2

# Newton's method: Variants and Extensions

## 2.1 Convergence and Drawbacks

Newton's method is relatively simple in concept and in structure. In fact, one can prove certain convergence to a unique minimizer, given some conditions. This is touched upon in Section 2.1.1. Despite its elegant nature, it also has its drawbacks, for instance issues with the Hessian, non-quadratic (damped) convergence, and even no convergence at all.

### 2.1.1 Convergence

Under certain conditions, Newton's method has a quadratic rate of convergence. For illustrative purposes, this means that on every iteration the number of correct digits of the iterate approximately doubles. Below we provide the theorem and proof of the convergence result as given by Nocedal and Wright (2006), page 44/45.
Before beginning the proof we provide the definition of the spectral norm (operator norm) of a matrix. Let $A \in \mathbb{R}^{n\times n}$. For any norm on $\mathbb{R}^n$, we have the induced matrix norm

$$\|A\| \coloneqq \sup_{x\in\mathbb{R}^n\setminus\{0\}} \frac{\|Ax\|}{\|x\|}. \tag{2.1}$$

We define the Euclidean norm of a vector $x$ as follows:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{2.2}$$

This then yields:

$$\|A\|_2 \coloneqq \sup_{x\in\mathbb{R}^n\setminus\{0\}} \frac{\|Ax\|_2}{\|x\|_2}. \tag{2.3}$$

It is equal to the largest singular value of $A$, or equivalently the square root of the largest eigenvalue of the matrix $A^\top A$, where $A^\top$ is the transpose of $A$. From (2.3), it is implied that:

$$\implies \|Ax\|_2 \leq \|A\|_2 \cdot \|x\|_2. \tag{2.4}$$

We also remind the reader of a property of integrals: Let $\boldsymbol{v} = (v_1, \ldots, v_n) = \int_a^b \boldsymbol{g}(x)\,dx$, where $\boldsymbol{g} = (g_1, \ldots, g_n) : \mathbb{R} \longrightarrow \mathbb{R}^n$ is a continuous function. We have by definition that $v_i = \int_a^b g_i(x)\,dx$. If $\boldsymbol{v} = \boldsymbol{0}$, then $\|\boldsymbol{v}\|_2 \leq \int_a^b \|\boldsymbol{g}(x)\|_2\,dx$ holds. Thus we have:

$$\begin{aligned}
\implies \|\boldsymbol{v}\|_2^2 &= \sum_{i=1}^n v_i^2 \\
&= \sum_{i=1}^n v_i \int_a^b g_i(x)\,dx \\
&= \int_a^b \sum_{i=1}^n (v_i \cdot g_i(x))\,dx \\
&= \int_a^b \boldsymbol{v} \cdot \boldsymbol{g}(x)\,dx \\
&\leq \int_a^b \|\boldsymbol{v}\|_2 \|\boldsymbol{g}(x)\|_2\,dx \qquad (2.5) \\
&= \|\boldsymbol{v}\|_2 \int_a^b \|\boldsymbol{g}(x)\|_2\,dx \\
\implies \|\boldsymbol{v}\|_2 &\leq \int_a^b \|\boldsymbol{g}(x)\|_2\,dx, \qquad (2.6)
\end{aligned}$$

where (2.5) is due to the Cauchy-Schwarz inequality and (2.6) is yielded by dividing by $\|\boldsymbol{v}\|_2$. Thus $\|\boldsymbol{v}\|_2 \leq \int_a^b \|\boldsymbol{g}(x)\|_2\,dx$.

**Theorem 2.1.1** (Quadratic Convergence)**.** *Suppose that $f$ is twice differentiable and that the Hessian $\nabla^2 f(x)$ is Lipschitz continuous with Lipschitz constant $L$ (see (Definition A.1.1)), in a neighbourhood of a solution $x^*$ at which the sufficient conditions (see Theorem A.1.1) are satisfied. Consider the iteration $x^{(k+1)} = x^{(k)} - \left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1} \nabla f\left(x^{(k)}\right)$. Then the rate of convergence of $\{x^{(k)}\}$, once inside the aforementioned neighbourhood, is quadratic, i.e. $\left\|x^{(k+1)} - x^*\right\|_2 \leq L \left\|\nabla^2 f(x^*)^{-1}\right\|_2 \left\|x^{(k)} - x^*\right\|_2^2$.*

*Proof.* From the right hand side of the iteration and the optimality condition $\nabla f(x^*) = 0$ we have that

$$\begin{aligned}
x^{(k+1)} - x^* &= x^{(k)} - \left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1} \nabla f\left(x^{(k)}\right) - x^* \\
&= \left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1} \left[\nabla^2 f\left(x^{(k)}\right)\left(x^{(k)} - x^*\right) - \left(\nabla f\left(x^{(k)}\right) - \nabla f(x^*)\right)\right].
\end{aligned} \tag{2.7}$$

Since Taylor's theorem (Theorem A.1.2) tells us that

$$\nabla f(x^*) = \nabla f\left(x^{(k)}\right) + \int_0^1 \nabla^2 f\left(x^{(k)} + t\left(x^* - x^{(k)}\right)\right)\left(x^* - x^{(k)}\right)\,dt$$

$$\implies \nabla f\left(x^{(k)}\right) - \nabla f(x^*) = \int_0^1 \nabla^2 f\left(x^{(k)} + t\left(x^* - x^{(k)}\right)\right)\left(x^{(k)} - x^*\right)\,dt,$$

we have

$$\begin{aligned}
&\left\|\nabla^2 f\left(x^{(k)}\right)\left(x^{(k)}-x^*\right)-\left(\nabla f(x^{(k)}-\nabla f(x^*)\right)\right\|_2 \\
&=\left\|\nabla^2 f\left(x^{(k)}\right)\left(x^{(k)}-x^*\right)-\int_0^1 \nabla^2 f\left(x^{(k)}+t\left(x^*-x^{(k)}\right)\right)\left(x^{(k)}-x^*\right)\,dt\right\|_2 \\
&=\left\|\int_0^1\left[\nabla^2 f\left(x^{(k)}\right)-\nabla^2 f\left(x^{(k)}+t\left(x^*-x^{(k)}\right)\right)\right]\left(x^{(k)}-x^*\right)\,dt\right\|_2 \\
&\leq \int_0^1\left\|\left[\nabla^2 f\left(x^{(k)}\right)-\nabla^2 f\left(x^{(k)}+t\left(x^*-x^{(k)}\right)\right)\right]\left(x^{(k)}-x^*\right)\right\|_2 dt \qquad (2.8)\\
&\leq \int_0^1\left\|\nabla^2 f\left(x^{(k)}\right)-\nabla^2 f\left(x^{(k)}+t\left(x^*-x^{(k)}\right)\right)\right\|_2\left\|x^{(k)}-x^*\right\|_2 dt \qquad (2.9)\\
&\leq \int_0^1 L\left\|x^{(k)}-x^{(k)}-t\left(x^*-x^{(k)}\right)\right\|_2\left\|x^{(k)}-x^*\right\|_2 dt \\
&=\left\|x^{(k)}-x^*\right\|_2^2\int_0^1 Lt\,dt=\frac{1}{2}L\left\|x^{(k)}-x^*\right\|_2^2, \qquad (2.10)
\end{aligned}$$

where $L$ is the Lipschitz constant for $\nabla^2 f(x)$ for $x$ near $x^*$. Line (2.8) follows from the property of integrals mentioned in (2.6) and line (2.9) follows from (2.4) Since $\nabla^2 f(x^*)$ is nonsingular, there is a radius $r>0$ such that $\left\|\nabla^2 f\left(x^{(k)}\right)^{-1}\right\|_2 \leq 2\left\|\nabla^2 f(x^*)^{-1}\right\|_2$ for all $x^{(k)}$ with $\left\|x^{(k)}-x^*\right\|_2 \leq r$. By substituting in (2.7) and (2.10), we obtain

$$\begin{aligned}
\left\|x^{(k+1)}-x^*\right\|_2 &= \left\|x^{(k)}-\left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1}\nabla f\left(x^{(k)}\right)-x^*\right\|_2 \\
&\leq \left\|\nabla^2 f\left(x^{(k)}\right)^{-1}\right\|_2\left\|\nabla^2 f\left(x^{(k)}\right)\left(x^{(k)}-x^*\right)-\left(\nabla f(x^{(k)}-\nabla f(x^*)\right)\right\|_2 \\
&\leq 2\left\|\nabla^2 f(x^*)^{-1}\right\|_2\cdot\frac{1}{2}L\left\|x^{(k)}-x^*\right\|_2^2 \\
&= \tilde{L}\left\|x^{(k)}-x^*\right\|_2^2, \qquad (2.11)
\end{aligned}$$

where $\tilde{L}=L\left\|\nabla^2 f(x^*)^{-1}\right\|_2$. Choosing $x^{(0)}$ so that $\left\|x^{(0)}-x^*\right\|_2 \leq \min(r,\, 1/(2\tilde{L}))$, we can use this inequality to deduce that the sequence converges to $x^*$, and the rate of convergence is quadratic and we yield the following as required:

$$\left\|x^{(k+1)}-x^*\right\|_2 \leq L\left\|\nabla^2 f(x^*)^{-1}\right\|_2\left\|x^{(k)}-x^*\right\|_2^2. \qquad (2.12)$$

□

### 2.1.2 Drawbacks

As mentioned above, a few conditions are required for quadratic convergence. If these assumptions are not met, then the Newton's method in optimization can run into problems. Sometimes, the rate of convergence is not quadratic and thus alternative methods, which are less complex compared to Newton's method, converge just as quick and hence are preferred, such as gradient and steepest descent methods. In cases where the minimum of the objective function has multiplicity higher than 1, convergence rate drops to linear. This is

elaborated on by Gilbert (1998) on for Newton's method for root finding.
Another condition of the noted in Section 2.1.1 was the initial point $x^{(0)}$ being close enough to the minimizer $x^*$. This is an important requirement because if not satisfied, Newton's method may not converge at all, despite other favourable conditions like convexity and Lipschitz continuity still being satisfied.
Consider the following two examples.

**Example 2.1.1** (Freund (2004) page 5)**.** *Let $f(x) = 7x - \ln(x)$. Then we have $\nabla f(x) = 7 - \frac{1}{x}$ and $\nabla^2 f(x) = \frac{1}{x^2}$. Then applying our iterative formula yields:*

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \left[\frac{1}{\left(x^{(k)}\right)^2}\right]^{-1} \cdot \left[7 - \frac{1}{x^{(k)}}\right] \\ \implies x^{(k+1)} &= 2x^{(k)} - 7\left(x^{(k)}\right)^2. \end{aligned}$$

*Since $f$ is convex and its derivative vanishes at this point, we have $x^* = \frac{1}{7}$. Note that $f(x)$ is also strongly convex (on $S = \{x \in \mathbf{dom}\ f \mid f(x) < f(0.2857\ldots) \approx 3.25\ldots\}$) since $\nabla^2 f(x) = \frac{1}{x^2} \succeq mI$, for $m > 0$ (see Appendix A.2). The following table provides some iterations for three different starting points:*

| $k$ | $x^{(k)}$ | $x^{(k)}$ | $x^{(k)}$ |
|---|---|---|---|
| 0 | 0.1 | 0.3 | 1 |
| 1 | 0.13 | $-0.03$ | $-5$ |
| 2 | 0.1417 | $-0.0663$ | $-185$ |
| 3 | 0.14284777 | $-0.1634$ | $-239,945$ |
| 4 | 0.142857142 | $-0.513567569$ | $-4.0302 \times 10^{11}$ |
| 5 | 0.142857143 | $-2.873396678$ | $-1.1370 \times 10^{24}$ |
| 6 | 0.142857143 | $-63.541652634$ | $-9.0486 \times 10^{48}$ |
| 7 | 0.142857143 | $-2.8390 \times 10^{4}$ | $-5.7314 \times 10^{98}$ |
| 8 | 0.142857143 | $-5.6420 \times 10^{09}$ | $-2.2994 \times 10^{198}$ |

Table 2.1: Initial point comparison

**Example 2.1.2** (Fletcher (1988) page 47)**.** *Let $f(x_1, x_2) = x_1^4 + x_1 x_2 + (1 + x_2)^2$. From the figure below we can see that there is a minimum at $(0.69588, -1.34794)$.*

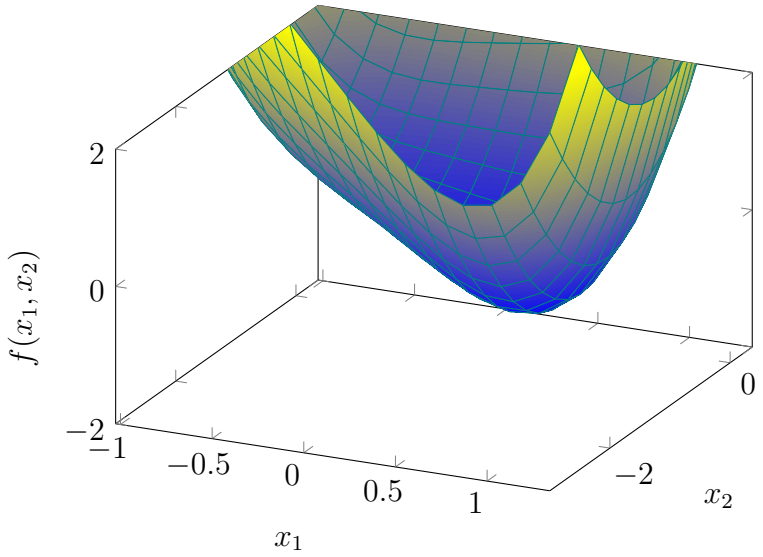


Figure 2.1: $f(x_1, x_2) = x_1^4 + x_1 x_2 + (1 + x_2)^2$

*The gradient and Hessian can be computed to yield:*

$$\nabla f(x_1, x_2) = \begin{bmatrix} 4x_1^3 + x_2 \\ x_1 + 2 + 2x_2 \end{bmatrix} \quad \textit{and} \quad \nabla^2 f(x_1, x_2) = \begin{bmatrix} 12x_1^2 & 1 \\ 1 & 2 \end{bmatrix}.$$

*If we take* $x^{(0)} = (0, 0)^\top$*, then we have*

$$\begin{aligned} x^{(1)} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 2 \end{bmatrix} \\ \implies x^{(1)} &= \begin{bmatrix} -2 \\ 0 \end{bmatrix}. \end{aligned}$$

*Because this Newton step direction only changes the* $x_1$ *parameter, we see that it increases the function value, travelling away from the minimum. Continuing with the iterative process shows that the method doesn't descend to the minimizer and fails to converge, and this is due to* $\nabla^2 f(0, 0) \not\succeq 0$*.*

Perhaps the most obvious caveat of Newton's method for optimization is regarding the Hessian matrix. In the multivariate case, if the Hessian is not invertible, then the method breaks down since we need to compute the inverse of the Hessian. In some cases, the Hessian may be difficult or expensive to compute at each iteration. To combat this, we look at the Gauss-Newton variant in the following Section 2.2.

## 2.2 Gauss-Newton variant

The Gauss-Newton method is a variant of the regular Newton method, commonly used in nonlinear least squares optimization. Its advantage and purpose is to remove the potential problems that arise when computing a difficult Hessian, and thus is attractive to use in practical applications. In fact, I will use

the Gauss-Newton method later in this thesis to optimize parameters to model a logistic curve via nonlinear least squares. First I will give two derivations stemming from different reasonings, and then I will present the algorithm.

### 2.2.1 Approach 1

The first approach is where we approximate the Hessian matrix by neglecting certain terms. We continue with the case we started with in Section 1.1, and develop it according to Nocedal and Wright (2006). We re-define the following terms:

$$\begin{aligned} (t_j, y_j) \text{ for } j = 1, \ldots, m && \text{(Observations)} \\ r_j(x) = y_j - \phi(x; t_j) && \text{(Residuals)} \\ f(x) = \frac{1}{2} \sum_{j}^{m} r_j^2(x). && \text{(Objective function)} \end{aligned}$$

We use the *residual vector* $r : \mathbb{R}^n \to \mathbb{R}^m$, where

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_m(x) \end{bmatrix}. \tag{2.13}$$

This allows us to write $f$ as $f(x) = \frac{1}{2}\|r(x)\|_2^2$. Furthermore, we define the matrix of first-order partial derivatives (of the residuals), known as the *Jacobian* $\mathcal{J}(x)$, evaluated at $x$, as follows:

$$\mathcal{J}(x) = \left[\frac{\partial r_j}{\partial x_i}\right]_{i=1,\ldots,n}^{j=1,\ldots,m} = \begin{bmatrix} \nabla r_1(x)^\top \\ \nabla r_2(x)^\top \\ \vdots \\ \nabla r_m(x)^\top \end{bmatrix}, \tag{2.14}$$

where each $\nabla r_j(x) \in \mathbb{R}^n,\ j = 1, \ldots, m$ is the gradient of $r_j$ at $x$. Thus we have that $\mathcal{J}(x) \in \mathbb{R}^{m \times n}$. The gradient and Hessian of $f$ are defined as follows:

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x) \nabla r_j(x) = \mathcal{J}(x)^\top r(x), \tag{2.15}$$

$$\begin{aligned} \nabla^2 f(x) &= \sum_{j=1}^{m} \nabla r_j(x) \nabla r_j(x)^\top + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x) \\ &= \mathcal{J}(x)^\top \mathcal{J}(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x). \end{aligned} \tag{2.16}$$

Therefore, we have that $\nabla f(x) \in \mathbb{R}^n$ and $\nabla^2 f(x) \in \mathbb{R}^{n \times n}$. In general, the Jacobian is relatively cheap to calculate, and thus the gradient of $f$ is easy to compute. On the other hand, the Hessian of $f$ can be expensive to

determine. However, there are a few advantageous aspects of $\nabla^2 f(x)$ that can be exploited. The first term, $\mathcal{J}(x)^\top \mathcal{J}(x)$, we have for free as we needed to calculate the Jacobian for $\nabla f(x)$. In the case when the $r_j$ are linear functions (linear regression), then the $\nabla^2 r_j(x)$ is zero and thus so is the second term. This is the basis for Approach 2, shown below. But even if we have a nonlinear function, near the solution we have that the residuals $r_j$ are negligible and so this means the second term is also relatively small. Hence, the second term is dominated by the first term in $\nabla^2 f(x)$, and this leads us to our approximation:

$$\nabla^2 f(x) \approx \mathcal{J}(x)^\top \mathcal{J}(x). \tag{2.17}$$

We take our iterative formula from (1.10) and substitute in (2.15) and (2.17):

$$x^{(k+1)} = x^{(k)} - \left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1} \nabla f\left(x^{(k)}\right) \tag{2.18}$$

$$\approx x^{(k)} - \left[\mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\right]^{-1} \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right). \tag{2.19}$$

### 2.2.2 Approach 2

The idea behind Approach 2, presented by Givens and Hoeting (2013) (page 44), is to linearly approximate the function by using the first-order Taylor, and then apply Ordinary Least Squares to obtain the iterative formula. I show a similar derivation below. Begin with the same setup from Approach 1 and use Taylor's linear approximation of our nonlinear function:

$$\tilde{\phi}\left(x; t_j\right) = \phi\left(x^{(k)}; t_j\right) + \nabla\phi\left(x^{(k)}; t_j\right)^\top \left[x - x^{(k)}\right]. \tag{2.20}$$

Then we have

$$r_j(x) = y_j - \phi\left(x^{(k)}; t_j\right) - \nabla\phi\left(x^{(k)}; t_j\right)^\top \left[x - x^{(k)}\right].$$

Denote the following terms:

$$\begin{aligned} \tilde{y_j} &= y_j - \phi\left(x^{(k)}; t_j\right) \\ \implies \tilde{y_j} &= r_j\left(x^{(k)}\right) \\ \implies \tilde{y} &= r\left(x^{(k)}\right) \\ \tilde{x} &= x - x^{(k)}. \end{aligned}$$

Then we obtain

$$\begin{aligned} \tilde{r_j}(x) &= \tilde{y_j} - \nabla\phi\left(x^{(k)}; t_j\right)^\top \tilde{x} \\ \implies \tilde{r}(x) &= \tilde{y} + \mathcal{J}\left(x^{(k)}\right)\tilde{x}. \end{aligned} \tag{2.21}$$

Notice that the Jacobian defined in (2.14) is the same as $-\nabla\phi\left(x^{(k)}; t_j\right)^\top$ elongated into matrix form, when evaluated at $x^{(k)}$ since $\nabla r\left(x^{(k)}\right) = \nabla\left(y - \phi\left(x^{(k)}; t_j\right)\right)$ and $y$ is independent of $x$. And finally we denote our objective function:

$$\tilde{f}(x) = \frac{1}{2}\|\tilde{r}(x)\|_2^2. \tag{2.22}$$

To find the optimal next iteration we require $\arg\min_x \tilde{f}(x)$, so we set the first-order derivative to zero and solve for $x$.

$$\begin{aligned}
\nabla \tilde{f}(x) &= \|\tilde{r}(x)\|_2 \cdot \frac{\nabla \tilde{r}(x)^\top \tilde{r}(x)}{\|\tilde{r}(x)\|_2} && (2.23)\\
\implies 0 &= \mathcal{J}\left(x^{(k)}\right)^\top \tilde{r}(x)\\
\implies 0 &= \mathcal{J}\left(x^{(k)}\right)^\top \left(\tilde{y} - \mathcal{J}\left(x^{(k)}\right)\tilde{x}\right)\\
\implies 0 &= \mathcal{J}\left(x^{(k)}\right)^\top \tilde{y} - \mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\left[x - x^{(k)}\right]\\
\implies 0 &= \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right) - \mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\left[x - x^{(k)}\right]\\
\implies x &= x^{(k)} - \left[\mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\right]^{-1} \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right). && (2.24)
\end{aligned}$$

Thus this yields the same iterative formula as (2.19):

$$x^{(k+1)} = x^{(k)} - \left[\mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\right]^{-1} \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right). \tag{2.25}$$

### 2.2.3 Global convergence of the Gauss-Newton method

Given some favourable assumptions, it can be guaranteed that the Gauss-Newton variant will converge to a minimizer. We provide the theorem below, but not its proof. For both the theorem and the proof, we refer the reader to Dennis and Schnabel (1996) page 222/223. We define $N(x, r)$ as the open neighbourhood of radius $r$ around $x$, i.e. $N(x, r) = \{\bar{x} \in \mathbb{R}^n : \|\bar{x} - x\|_2 < r\}$.

**Theorem 2.2.1** (Global convergence)**.** *Let $r : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, and let $f(x) = \frac{1}{2} r(x)^\top r(x)$ be twice continuously differentiable in an open convex set $\mathcal{D} \subset \mathbb{R}^n$. Assume that $\mathcal{J}(x)$ is Lipschitz continuous in $\mathcal{D}$ with $\|\mathcal{J}(x)\|_2 \leq \alpha$ for all $x \in \mathcal{D}$, and that there exists $x^* \in \mathcal{D}$ and $\lambda$, $\sigma \geq 0$, such that $\mathcal{J}(x^*)^\top r(x^*) = 0$, $\lambda$ is the smallest eigenvalue of $\mathcal{J}(x^*)^\top \mathcal{J}(x^*)$, and*

$$\|(\mathcal{J}(x) - \mathcal{J}(x^*))^\top r(x^*)\|_2 \leq \sigma \|x - x^*\|_2 \tag{2.26}$$

*for all $x \in \mathcal{D}$. If $\sigma < \lambda$, then for any $c \in (1, \frac{\lambda}{\sigma})$, there exists $\varepsilon > 0$ such that for all $x^{(0)} \in N(x^*, \varepsilon)$, the sequence generated by the Gauss-Newton method*

$$x^{(k+1)} = x^{(k)} - \left[\mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\right]^{-1} \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right)$$

*is well defined, converges to $x^*$, and obeys*

$$\|x^{(k+1)} - x^*\|_2 \leq \frac{c\sigma + \lambda}{2\lambda} \|x^{(k)} - x^*\|_2 < \|x^{(k)} - x^*\|_2. \tag{2.27}$$

# Chapter 3

# Growth models

## 3.1 Growth models

In the literature regarding growth mathematics, there are numerous growth models, and their analyses are an important field of study for mathematical biologist. Despite the vast number of models, there are two main categories that the majority fall into: exponential and logistic. Thomas Malthus (1798) argued in his work *An Essay on the Principle of Population* that human population grew according to a geometric progression, while other resources like food production grew linearly. This form of exponential growth is well-known and even influenced Charles Darwin in his work.

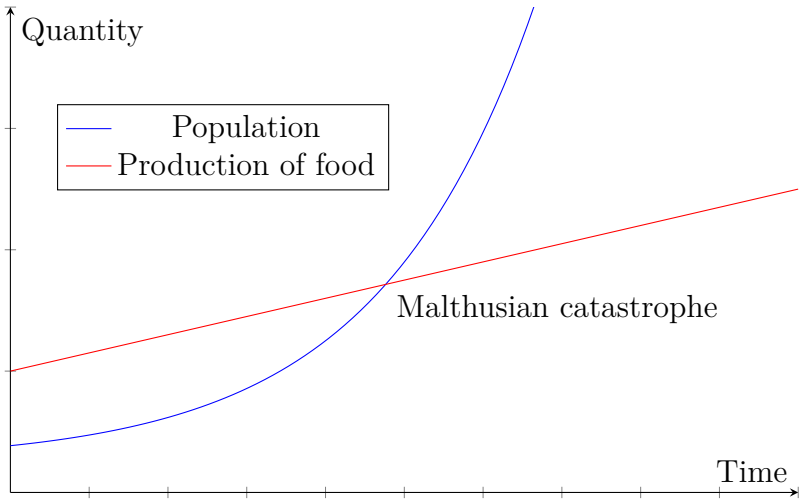


Figure 3.1: Malthusianism

Source: ©commons.wikimedia.org

Another form of growth is the logistic curve. Verhulst (1845) introduced the concept of the logistic function, also known as the sigmoid curve. It is adapted from the exponential growth pattern to include the effects of limited resources, causing a plateau. It has multiple applications, for instance it is a well known shape in statistics for the cumulative distribution function.

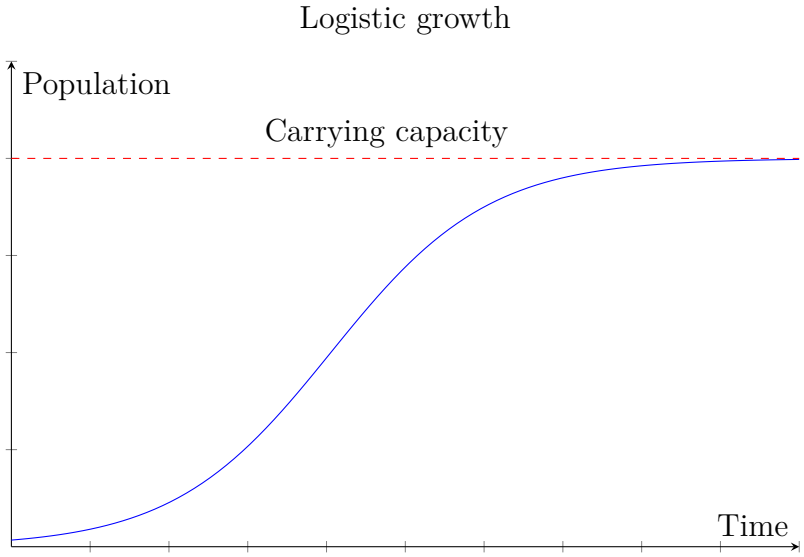


Figure 3.2: Logistic growth

Source: Environmental limits to population growth by OpenStax College

## 3.2 Logistic function

In this section we provide the formulation of the logistic function and verify its relation to the differential equation given below. We then develop the function to be used in the Newton and Gauss-Newton algorithms for nonlinear least squares regression. Verhulst (1845) proposed the following differential equation to model population growth

$$\frac{dP}{dt} = aP\left(1 - \frac{P}{K}\right), \tag{3.1}$$

where $P$ is the population, $a > 0$ is the constant of growth, and $K$ is the carrying capacity. We also have the boundary case of $P(0) = P_0$, which we use later to solve for the integration constant. The following is the solution to the differential equation in (3.1)[1], which will be shown on the next page.

$$P(t) = \frac{K}{1 + \mathrm{e}^{-(at+C)}}. \tag{3.2}$$

So we have $P(t)$ as the population at time $t$, $a$ as the constant of growth, and $K$ as the carrying capacity. We choose $C$ such that it is in accordance with

[1]For a more modern article on the logistic curve and application in ecology, see Kingsland, 1982.

the function - let $P(0) = P_0$ be the population at time $t = 0$. Then we have

$$\begin{aligned} P_0 &= \frac{K}{1 + \mathrm{e}^{-(a\cdot 0 + C)}} \\ \implies P_0 &= \frac{K}{1 + \mathrm{e}^{-C}} \\ \implies P_0\left(1 + \mathrm{e}^{-C}\right) &= K \\ \implies 1 + \mathrm{e}^{-C} &= \frac{K}{P_0} \\ \implies C &= \ln\left(\frac{P_0}{K - P_0}\right). \end{aligned} \tag{3.3}$$

It is obvious to see that this function is nonlinear[2] in terms of $a$, $K$, and $P_0$ and so when we proceed to use this function to model population growth, we can use nonlinear least squares to estimate the constants defined above. We now verify that (3.2) satisfies (3.1) by taking the derivative of (3.2):

$$\begin{aligned} \frac{dP}{dt} &= \frac{-K}{\left(1 + \mathrm{e}^{-(rt+C)}\right)^2} \cdot -r\mathrm{e}^{-(rt+C)} \\ &= \frac{rK}{1 + \mathrm{e}^{-(rt+C)}} \cdot \frac{\mathrm{e}^{-(rt+C)}}{1 + \mathrm{e}^{-(rt+C)}} \\ &= rP\left(\frac{\mathrm{e}^{-(rt+C)}}{1 + \mathrm{e}^{-(rt+C)}}\right) \\ &= rP\left(\frac{1 + \mathrm{e}^{-(rt+C)} - 1}{1 + \mathrm{e}^{-(rt+C)}}\right) \\ &= rP\left(1 - \frac{1}{1 + \mathrm{e}^{-(rt+C)}}\right) \\ \implies \frac{dP}{dt} &= rP\left(1 - \frac{P}{K}\right). \end{aligned} \tag{3.4}$$

## 3.3 Applying the Newton and Gauss-Newton methods

Now we apply this function to the case of nonlinear least squares for both the Newton and Gauss-Newton methods. We provide the explicit forms of the derivatives obtained below at the end of this section. We, again, give the

[2]In the literature, there is debate over the growth rate $a$. More recent papers have given $a$ as the instantaneous rate of natural increase per individual, while Allee (1949) and Gause (1964) used $b$ instead of $a$, where $b$ is the instantaneous birth rate per individual and $a$ is equal to $b$ minus the instantaneous death rate per individual. For a comparison we refer the reader to Jensen (1975), where in notation $r$ is used for our $a$.

following terms:

$$
\begin{aligned}
f(x) &= \frac{1}{2}\left\|r(x)\right\| \\
r_j(x) &= y_j - \phi(x, t_j) \\
r(x) &= \begin{bmatrix} r_1(x) \\ \vdots \\ r_m(x) \end{bmatrix} \\
\nabla f(x) &= \sum_{j=1}^{m} r_j(x) \nabla r_j(x) = \mathcal{J}(x)^\top r(x), \qquad (3.5) \\
\nabla^2 f(x) &= \mathcal{J}(x)^\top \mathcal{J}(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x). \qquad (3.6)
\end{aligned}
$$

For applying the logistic function, we have

$$\phi(x, t_j) = \frac{K}{1 + \frac{K - P_0}{P_0} \mathrm{e}^{-at_j}}, \tag{3.7}$$

and we denote the following; $x_1 = a$, $x_2 = K$, $x_3 = P_0$. We determine the Jacobian matrix from (2.14). Thus we obtain:

$$\mathcal{J}(x) = \begin{bmatrix} \frac{\partial r_1(x)}{\partial x_1} & \frac{\partial r_1(x)}{\partial x_2} & \frac{\partial r_1(x)}{\partial x_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial r_m(x)}{\partial x_1} & \frac{\partial r_m(x)}{\partial x_2} & \frac{\partial r_m(x)}{\partial x_3} \end{bmatrix}.$$

But since we have

$$\frac{\partial r_j(x)}{\partial x_i} = -\frac{\partial \phi(x, t_j)}{\partial x_i}, \tag{3.8}$$

this yields

$$\implies \mathcal{J}(x) = \begin{bmatrix} -\frac{\partial \phi(x, t_1)}{\partial x_1} & -\frac{\partial \phi(x, t_1)}{\partial x_2} & -\frac{\partial \phi(x, t_1)}{\partial x_3} \\ \vdots & \vdots & \vdots \\ -\frac{\partial \phi(x, t_m)}{\partial x_1} & -\frac{\partial \phi(x, t_m)}{\partial x_2} & -\frac{\partial \phi(x, t_m)}{\partial x_3} \end{bmatrix}. \tag{3.9}$$

Hence, we can now use the Gauss-Newton iterative formula:

$$x^{(k+1)} \approx x_{(k)} - \left[\mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right)\right]^{-1} \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right), \tag{3.10}$$

with $x^{(k)}$ being the set of parameters at iteration $k$:

$$x^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} = \begin{bmatrix} a^{(k)} \\ K^{(k)} \\ P_0^{(k)} \end{bmatrix}.$$

However, for the Newton method we require $\nabla^2 r_j(x)$ from (3.6). Applying our logistic function gives us

$$\nabla^2 r_j(x) = \begin{bmatrix} \frac{\partial^2 r_j(x)}{\partial x_1^2} & \frac{\partial^2 r_j(x)}{\partial x_1 \partial x_2} & \frac{\partial^2 r_j(x)}{\partial x_1 \partial x_3} \\ \frac{\partial^2 r_j(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 r_j(x)}{\partial x_2^2} & \frac{\partial^2 r_j(x)}{\partial x_2 \partial x_3} \\ \frac{\partial^2 r_j(x)}{\partial x_3 \partial x_1} & \frac{\partial^2 r_j(x)}{\partial x_3 \partial x_2} & \frac{\partial^2 r_j(x)}{\partial x_3^2} \end{bmatrix},$$

and again we use (3.8) to yield

$$\implies \nabla^2 r_j(x) = \begin{bmatrix} -\frac{\partial^2 \phi(x,t_j)}{\partial x_1^2} & -\frac{\partial^2 \phi(x,t_j)}{\partial x_1 \partial x_2} & -\frac{\partial^2 \phi(x,t_j)}{\partial x_1 \partial x_3} \\ -\frac{\partial^2 \phi(x,t_j)}{\partial x_2 \partial x_1} & -\frac{\partial^2 \phi(x,t_j)}{\partial x_2^2} & -\frac{\partial^2 \phi(x,t_j)}{\partial x_2 \partial x_3} \\ -\frac{\partial^2 \phi(x,t_j)}{\partial x_3 \partial x_1} & -\frac{\partial^2 \phi(x,t_j)}{\partial x_3 \partial x_2} & -\frac{\partial^2 \phi(x,t_j)}{\partial x_3^2} \end{bmatrix}. \tag{3.11}$$

Now we can also use the Newton iterative formula:

$$\begin{aligned} x^{(k+1)} = x^{(k)} - & \left[ \mathcal{J}\left(x^{(k)}\right)^\top \mathcal{J}\left(x^{(k)}\right) + \sum_{j=1}^{m} r_j\left(x^{(k)}\right) \nabla^2 r_j\left(x^{(k)}\right) \right]^{-1} \\ & \times \mathcal{J}\left(x^{(k)}\right)^\top r\left(x^{(k)}\right). \end{aligned} \tag{3.12}$$

We now provide the first and second order derivatives for the Jacobian and Hessian matrices in (3.9) and (3.11), respectively. The first-order derivatives of $\phi$ with respect to each of the parameters:

$$\frac{\partial \phi(x,t_j)}{\partial x_1} = \frac{\partial \phi(x,t_j)}{\partial a} = -\frac{KP_0 \cdot (P_0 - K)\, t\mathrm{e}^{at_j}}{\left(P_0\mathrm{e}^{at_j} - P_0 + K\right)^2}, \tag{3.13}$$

$$\frac{\partial \phi(x,t_j)}{\partial x_2} = \frac{\partial \phi(x,t_j)}{\partial K} = \frac{P_0^2 \cdot \left(\mathrm{e}^{at_j} - 1\right)\mathrm{e}^{at_j}}{\left(K + P_0\mathrm{e}^{at_j} - P_0\right)^2}, \tag{3.14}$$

$$\frac{\partial \phi(x,t_j)}{\partial x_3} = \frac{\partial \phi(x,t_j)}{\partial P_0} = \frac{K^2\mathrm{e}^{at_j}}{\left(\left(\mathrm{e}^{at_j} - 1\right)P_0 + K\right)^2}. \tag{3.15}$$

The second-order derivatives of $\phi$ with respect to each of the parameters (Note, due to the symmetry of the Hessian, we provide only the upper triangular part):

$$\frac{\partial^2 \phi(x,t_j)}{\partial x_1^2} = \frac{\partial^2 \phi(x,t_j)}{\partial a^2} = \frac{KP_0 \cdot (P_0 - K)\, t_j^2\mathrm{e}^{at_j} \cdot \left(P_0 \cdot \left(\mathrm{e}^{at_j} + 1\right) - K\right)}{\left(P_0 \cdot \left(\mathrm{e}^{at_j} - 1\right) + K\right)^3}, \tag{3.16}$$

$$\frac{\partial^2 \phi(x,t_j)}{\partial x_1 \partial x_2} = \frac{\partial^2 \phi(x,t_j)}{\partial a \partial K} = \frac{P_0^2 t_j \mathrm{e}^{at_j} \cdot \left(\left(2\mathrm{e}^{at_j} - 1\right)K - P_0\mathrm{e}^{at_j} + P_0\right)}{\left(K + P_0\mathrm{e}^{at_j} - P_0\right)^3}, \tag{3.17}$$

$$\frac{\partial^2 \phi(x,t_j)}{\partial x_1 \partial x_3} = \frac{\partial^2 \phi(x,t_j)}{\partial a \partial P_0} = -\frac{K^2 t_j \mathrm{e}^{at_j} \cdot \left(\left(\mathrm{e}^{at_j} + 1\right)P_0 - K\right)}{\left(\left(\mathrm{e}^{at_j} - 1\right)P_0 + K\right)^3}, \tag{3.18}$$

$$\frac{\partial^2 \phi(x,t_j)}{\partial x_2^2} = \frac{\partial^2 \phi(x,t_j)}{\partial K^2} = -\frac{2P_0^2 \cdot \left(\mathrm{e}^{at_j} - 1\right)\mathrm{e}^{at_j}}{\left(K + P_0\mathrm{e}^{at_j} - P_0\right)^3}, \tag{3.19}$$

$$\frac{\partial^2 \phi(x,t_j)}{\partial x_2 \partial x_3} = \frac{\partial^2 \phi(x,t_j)}{\partial K \partial P_0} = \frac{2K \cdot \left(\mathrm{e}^{at_j} - 1\right)\mathrm{e}^{at_j} P_0}{\left(\left(\mathrm{e}^{at_j} - 1\right)P_0 + K\right)^3}, \tag{3.20}$$

$$\frac{\partial^2 \phi(x,t_j)}{\partial x_3^2} = \frac{\partial^2 \phi(x,t_j)}{\partial P_0^2} = -\frac{2K^2 \cdot \left(\mathrm{e}^{at_j} - 1\right)\mathrm{e}^{at_j}}{\left(\left(\mathrm{e}^{at_j} - 1\right)P_0 + K\right)^3}. \tag{3.21}$$

# Chapter 4

# Data: COVID-19 Deaths

In this section we briefly discuss the data used to test the Newton and Gauss-Newton method for nonlinear least squares. The data I use is the cumulative COVID-19 deaths per million people. The data was collected from Mathieu et al., 2020. I chose cumulative deaths as it then aggregates the data generating clear sigmoid or S-shape curves representing the different waves of the coronavirus pandemic. I further decided upon two countries, namely, the United Kingdom and the Netherlands, and then both their first and second waves. Data was chosen at the national level as the effect of the individual waves are more visible, due to the nature of the spread of the pandemic. Notice also how the dates of the two waves align for the two countries. The following figure shows the cumulative confirmed COVID-19 deaths per million people, for both the UK and the Netherlands, over the entire time period.

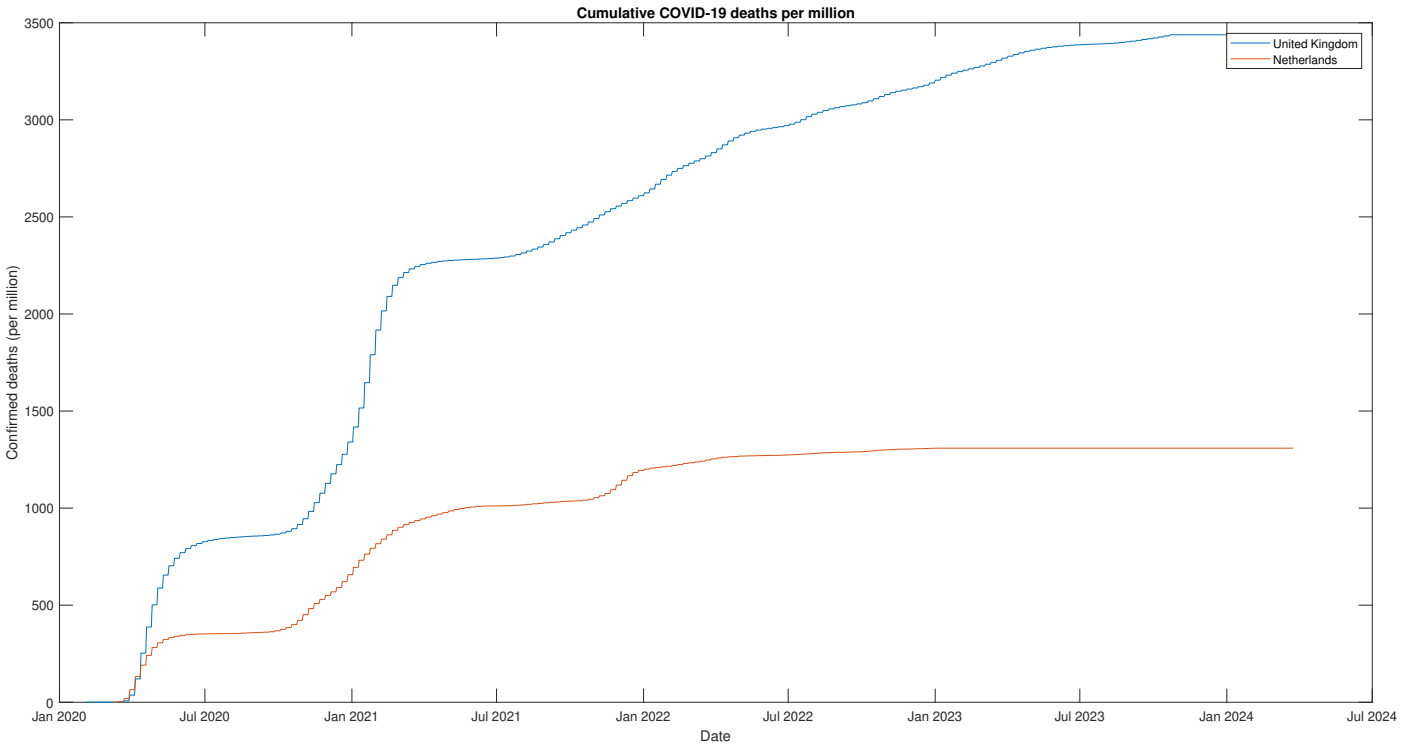


Figure 4.1: Cumulative confirmed COVID-19 deaths per million people

As can be clearly seen, the first and second waves for both the Netherlands and the UK express the logistic curve and hence will pose as good data to allow for fitting of our specified function via nonlinear least squares. We also provide the figures of each of the waves for the Netherlands and the UK below. Notice that the graphs have sharp jumps. This is because the data is given in a daily format but the figures only update weekly. Due to this format this could generate unnecessary persistent residuals that would exist even if the data fit the model perfectly. Hence, I took only the data points at the end of the week, removing the sharp jump-like features in the graphs and helping with convergence.

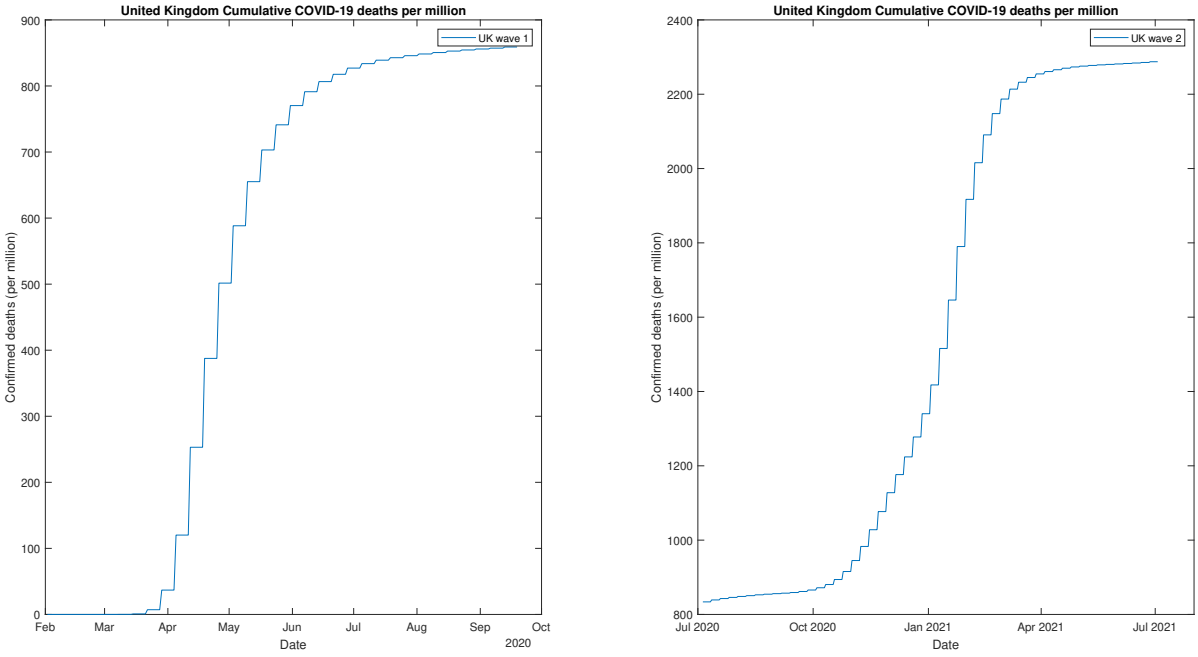


Figure 4.2: Cumulative confirmed COVID-19 deaths per million people (UK)

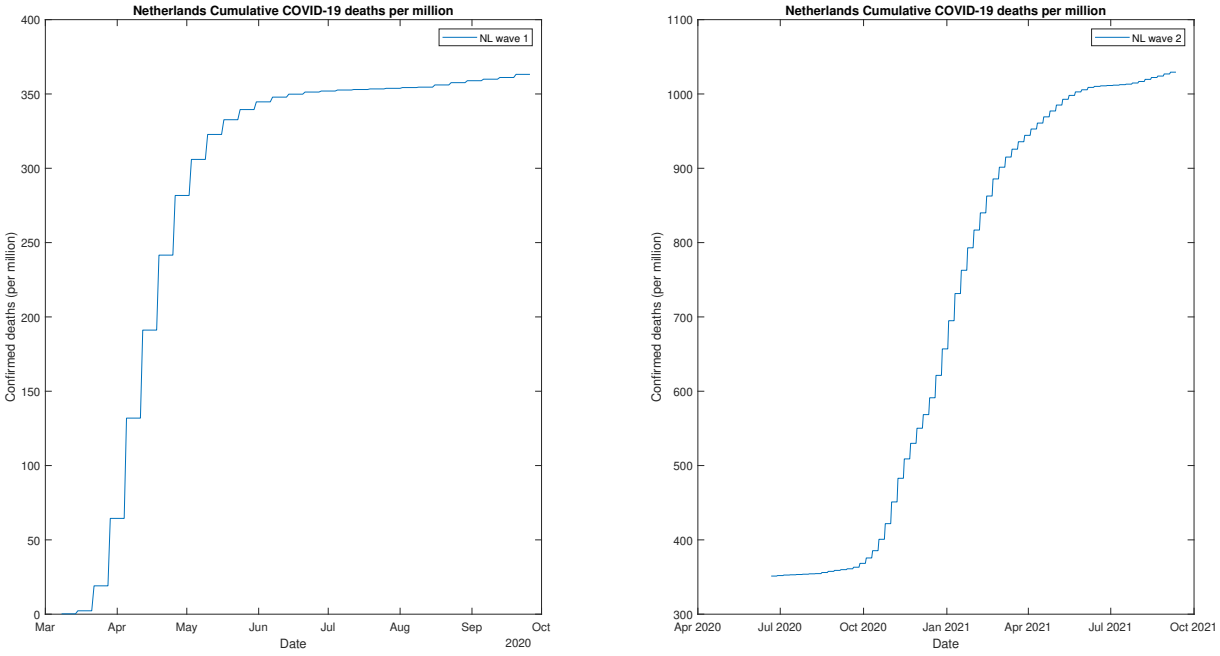


Figure 4.3: Cumulative confirmed COVID-19 deaths per million people (Netherlands)

# Chapter 5

# Numerical Experiments

In this section I will compare the Newton and Gauss-Newton methods in terms of rate of convergence and region of convergence for initial guesses, using the data provided in Chapter 4.

## 5.1 Initial Guesses

In order to initialize the algorithms, an initial input for the parameters is needed. This initial guess is important and plays a significant role in whether the methods converge or not. In Section 5.3 on the Rates of Convergence, I relax these conditions and explore other initial guesses. Before determining the initial parameters, each data set had to be shifted vertically such that it began just above zero. Thus, an extra vertical parameter $v$ was introduced. In each data set, for $v$, I use the lowest value. I set $P_0$ to one as the initial population needs to be positive for the function to work. I then set $K$ to be the largest observed data point from the (vertically shifted) data set that I am trying to fit. This then yields the same asymptotes as the data. In order to estimate a starting value for the growth parameter $a$, I used the median value. I set $t = X_{\frac{m}{2}}$ and taking the corresponding data point $y_{X_{\frac{m}{2}}} - v$, where $v$ is the vertical shift of the specific data set. This leaves us with an equation with one unknown parameter, for which we can solve:

$$\phi(x, X_{\frac{m}{2}}) = y_{X_{\frac{m}{2}}} - v = \frac{K}{1 + \frac{K-P_0}{P_0}\mathrm{e}^{-aX_{\frac{m}{2}}}} \tag{5.1}$$

$$\implies K = (y_{X_{\frac{m}{2}}} - v) + \frac{(K-P_0)(y_{X_{\frac{m}{2}}} - v)}{P_0}\mathrm{e}^{-aX_{\frac{m}{2}}} \tag{5.2}$$

$$\implies \mathrm{e}^{-aX_{\frac{m}{2}}} = \frac{(K - y_{X_{\frac{m}{2}}} + v)P_0}{(K-P_0)(y_{X_{\frac{m}{2}}} - v)} \tag{5.3}$$

$$\implies a = \frac{1}{X_{\frac{m}{2}}} \ln \frac{(K - y_{X_{\frac{m}{2}}} + v)P_0}{(K-P_0)(y_{X_{\frac{m}{2}}} - v)}. \tag{5.4}$$

Thus with this method for determining the initial parameters, the table below contains the initial parameters for each of the four data sets:

| Data set | $x_1^{(0)} = a^{(0)}$ | $x_2^{(0)} = K^{(0)}$ | $x_3^{(0)} = P_0^{(0)}$ |
|---|---|---|---|
| UK wave 1 | 0.0779 | 858.9680 | 1 |
| UK wave 2 | 0.0378 | 1,453.6 | 1 |
| NL wave 1 | 0.0980 | 362.9680 | 1 |
| NL wave 2 | 0.0328 | 678.0340 | 1 |

Table 5.1: Initial parameters

Naturally, this method provides initial plots with varying degrees of fit. Below we present the initial plot for the UK wave 1 data set. For the initial plots of the other data sets, see Appendix B.

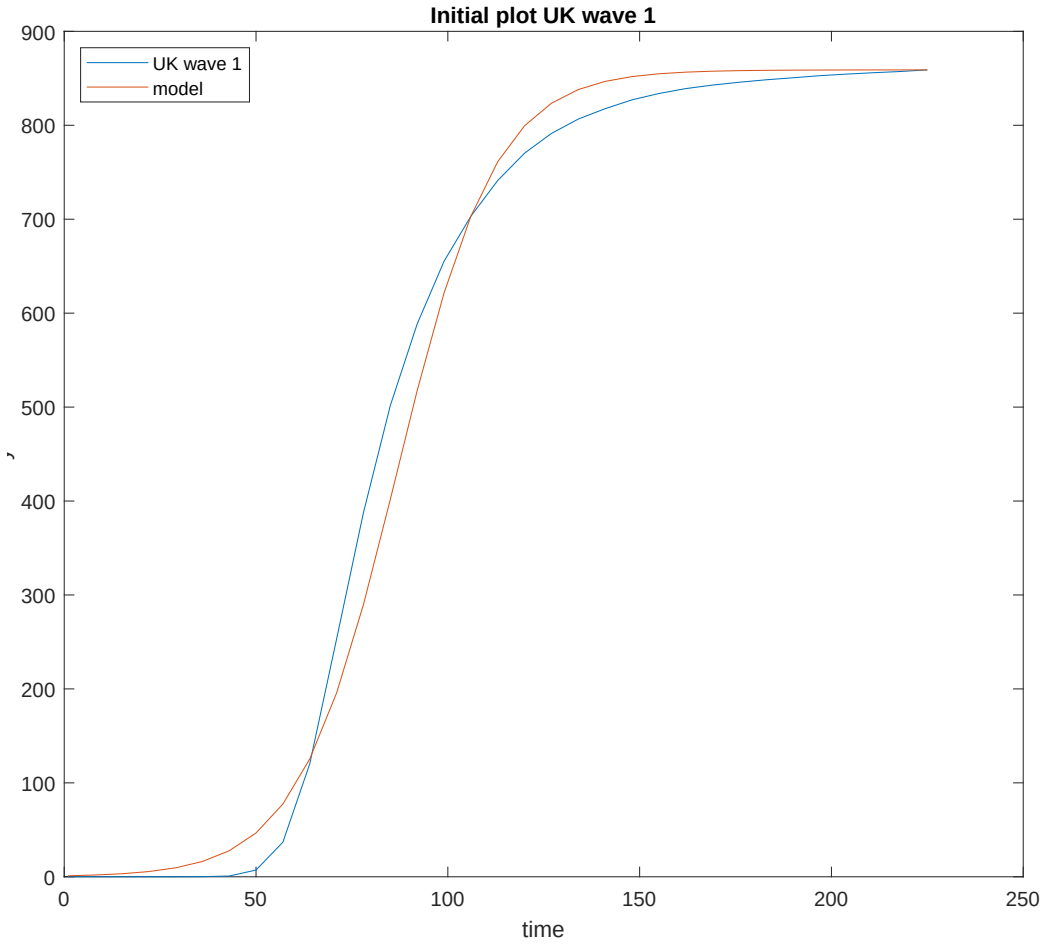


Figure 5.1: Initial plot for UK wave 1 data set

## 5.2 Rates of Convergence

Now using these initial guesses from Section 5.1, we can examine the convergence of the Newton and Gauss-Newton methods. For both algorithms, I employ four stopping criteria:

- If the absolute relative change in the objective value between iterations is less than $10^{-4}$,
- if the number of iterations exceeds 50,
- if the Hessian or Hessian approximation becomes singular,
- if the difference in the objective value between iterations is less than $-10^7$.

The size of the absolute relative change criterion was simply a modelling choice. Whereas the last stopping criterion was implemented as it shows if the method causes the iterate to move away from the minimizer, if the difference is negative. It is mainly implemented for use in Section 5.3 on Regions of Convergence. After some probing, $-10^7$ was chosen as it allowed certain initialisations to first move in the wrong direction, and then re-converge.

The following table provides the number of iterations for convergence and the final objective value for each of the methods, on each of the data sets:

| Data set | **Newton** | | |
|---|---|---|---|
| | Iterations | Parameters | Obj. value |
| UK wave 1 | 5 | 0.0843, 836.4683, 0.8168 | 8,918.833 |
| UK wave 2 | 4 | 0.0408, 1,469.4, 0.6809 | 24,388.592504 |
| NL wave 1 | 10 | 0.1028, 352.2705, 8.7588 | 978.598701 |
| NL wave 2 | 7 | 0.0284, 662.1426, 2.582 | 3,754.979548 |
| | **Gauss-Newton** | | |
| | Iterations | Parameters | Obj. value |
| UK wave 1 | 6 | 0.0843, 836.4683, 0.8174 | 8,918.846 |
| UK wave 2 | 5 | 0.0408, 1,469.4, 0.6807 | 24388.59806 |
| NL wave 1 | Singular Hessian approximation at iteration 6 | | |
| NL wave 2 | 5 | 0.0284, 662.1475, 2.5852 | 3,754.980433 |

Table 5.2: Convergence results

From the table, we see that both methods, when they converge, converge to almost exactly the same parameters for each the data sets, which gives confidence that this minima is correct. You can also see that the Newton method, in general, converges faster than the Gauss-Newton method. This makes sense according to the convergence results we obtained in Chapter 2. This difference is even more pronounced if the tolerance level of the first stopping criterion is decreased to $10^{-8}$, where the number of fewer iterations required for convergence with Newton's method grows to 3 from the 1 we see above. Furthermore, for NL wave 1, we even see that the Gauss-Newton method fails to converge under the same initial parameters. However, there is the exception for NL wave 2 where Gauss-Newton converges in one fewer iteration. This result is

even robust when decreasing the tolerance criterion. This result is likely due to favourable circumstance for the Gauss-Newton method, but it is possible that implementing the damped Newton method would rectify this. One could use a step size variable like backtracking line search, but this goes beyond the scope of this thesis. Below we present the plots representing the convergence of the Newton's method for the first data set.

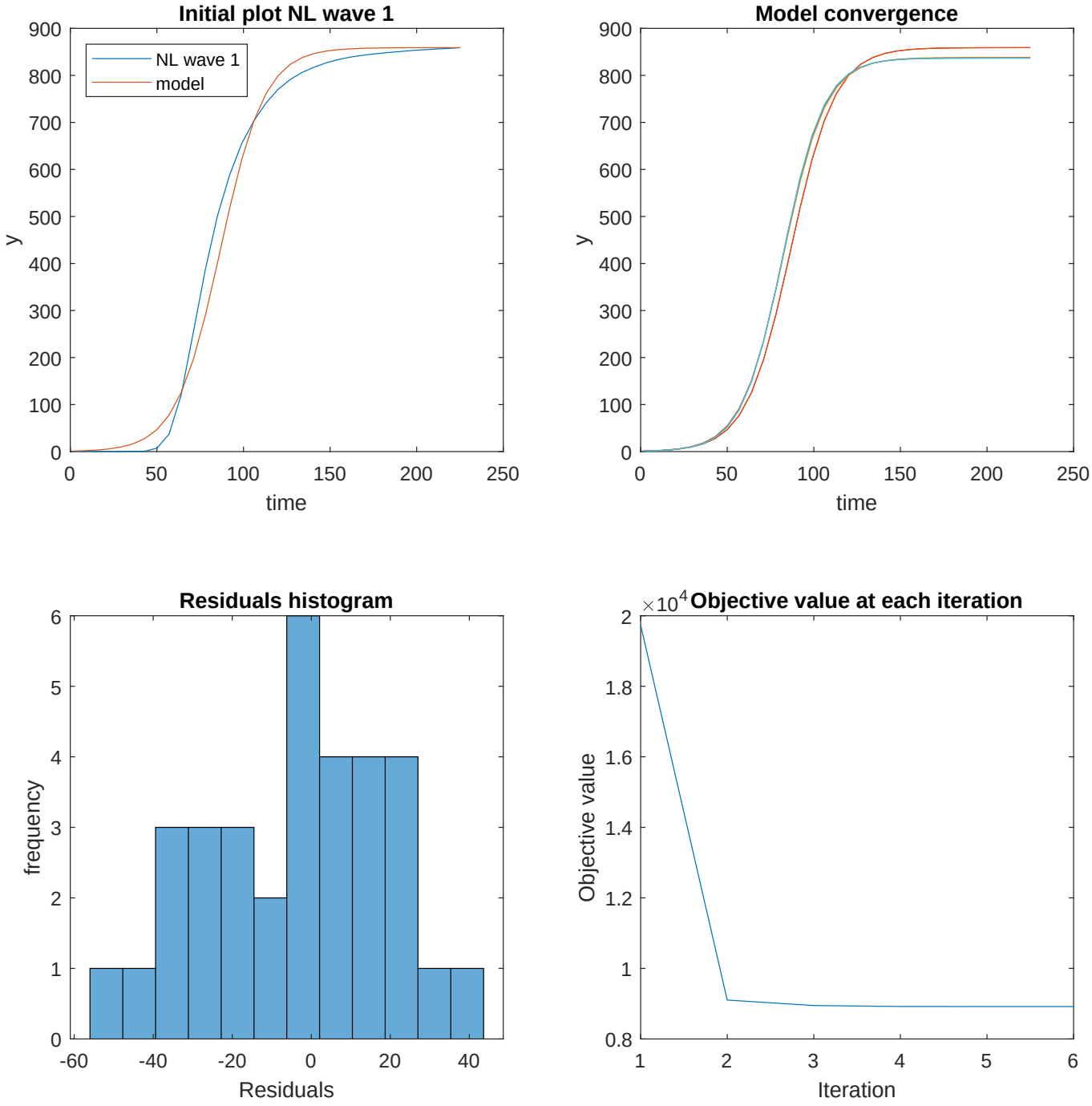


Figure 5.2: Convergence Newton's method for UK wave 1

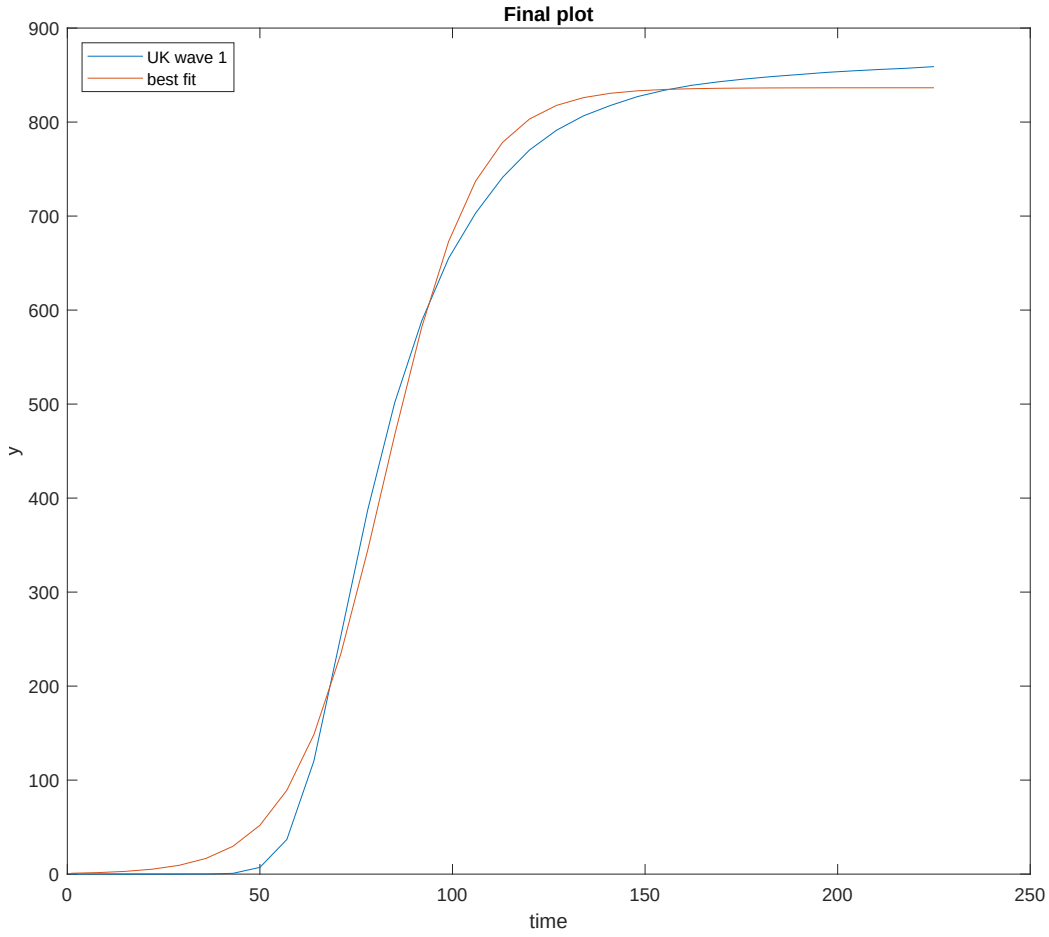


Figure 5.3: Final plot for Newton's method for UK wave 1

The residuals histogram shows that the residuals on the final plot are fairly symmetric around zero. The Objective value plot shows that the algorithm quickly yields a good fit, however the quadratic convergence rate isn't clearly shown, although it is present. This is because the initial guess is good. The Model convergence plot shows the model defined by the set of parameters at each iteration of the algorithm but again due to the good initialization the various iterations are not easily distinguishable. To highlight this we present the case of NL wave 1, where Newton's method takes ten iterations due to the poorer initial guess.

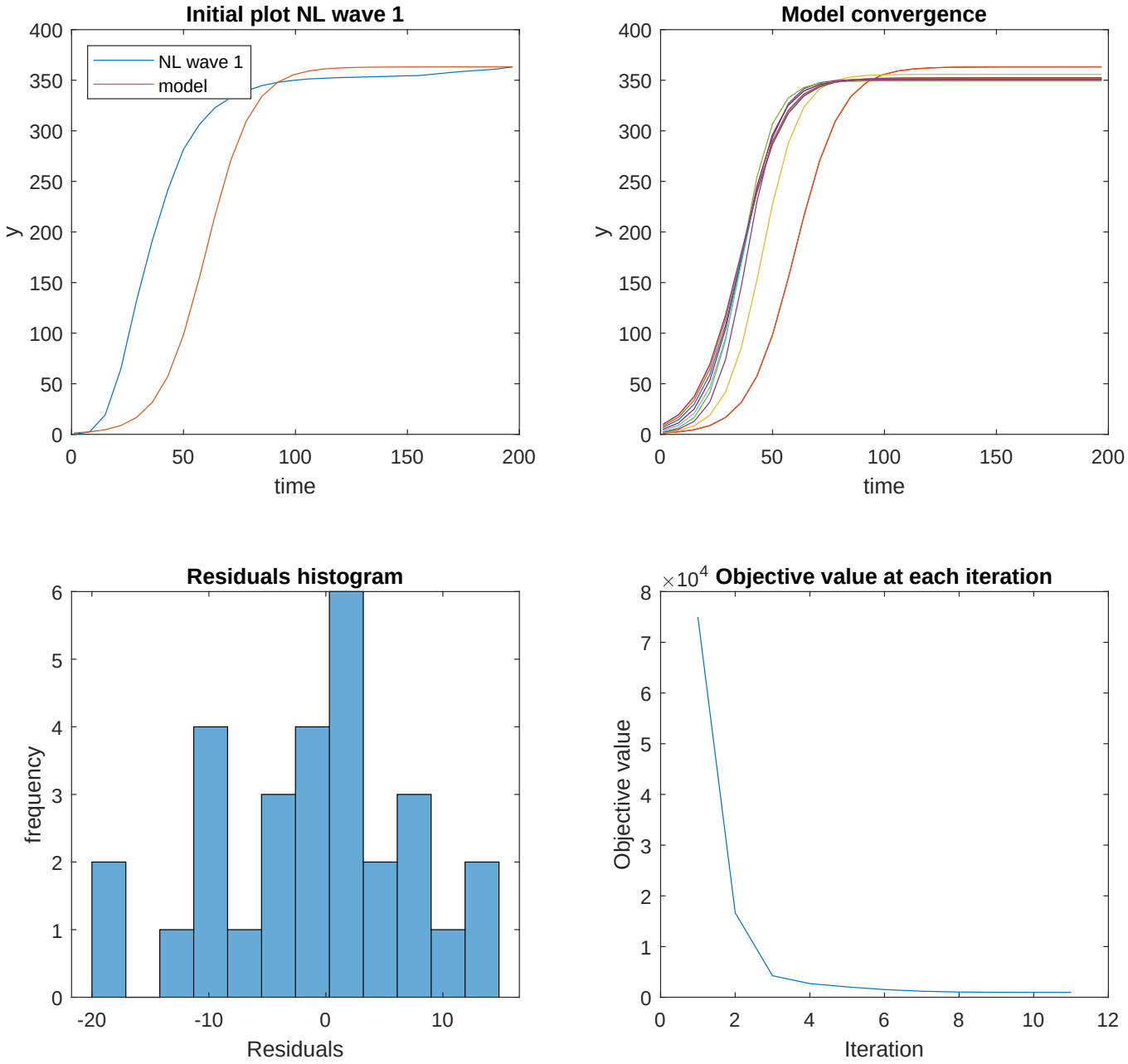


Figure 5.4: Convergence Newton's method for NL wave 1

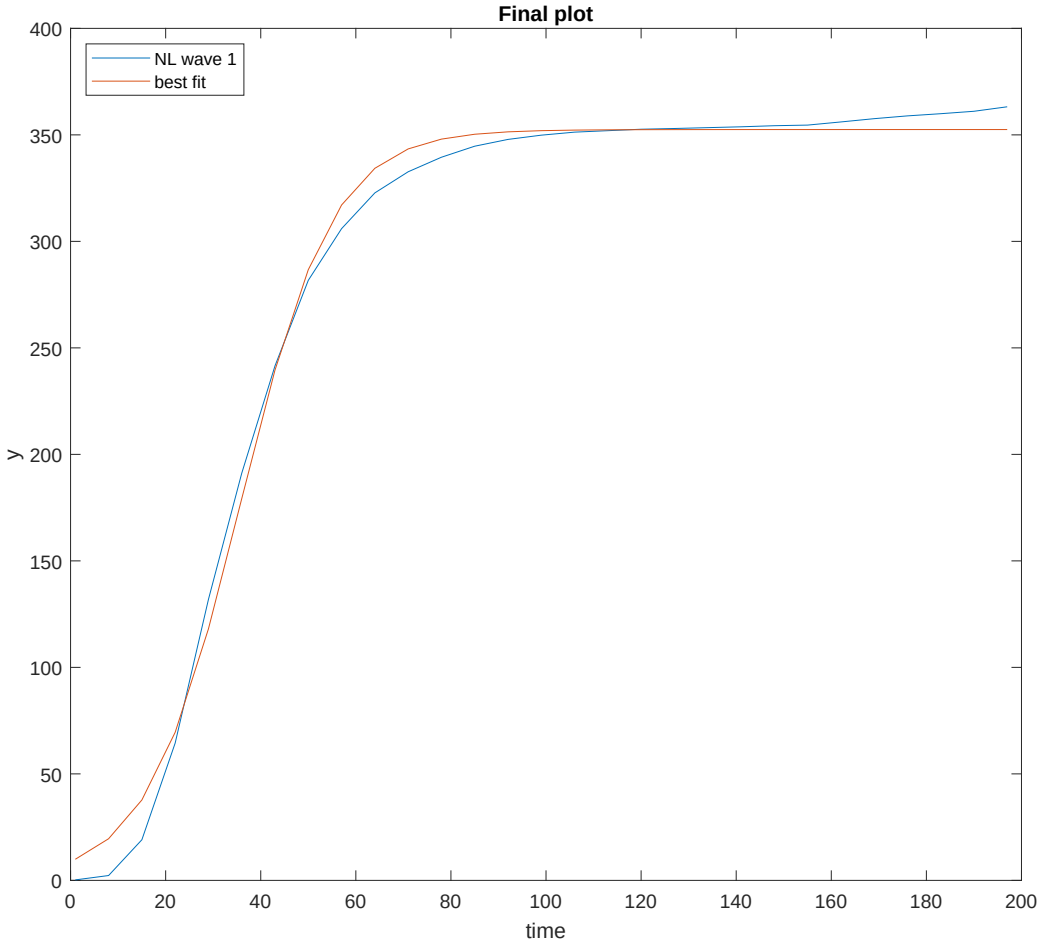


Figure 5.5: Final plot for Newton's method for NL wave 1

From this worse initial starting point, we can better see how the model changes after each iteration and converges to the model of best fit. Moreover, the objective function plot also better presents how the algorithm approaches the minimizer. For the final plots of each data set, refer to Appendix B. Here we show the initial plots, the final plots and the observed data in a single plot for each data set. As the initial guesses don't change for the two methods and they both converge to the same minima, we present only the results for the Newton method.

Comparing the methods against each other, we see that the Newton method generally outperforms the Gauss-Newton method in terms of speed of convergence. This can be seen by comparing the objective value plots. We present below the objective value plots for each method on each set of data. We remind the reader that the Gauss-Newton method diverges for wave 1 of the Netherlands, and hence is not shown.

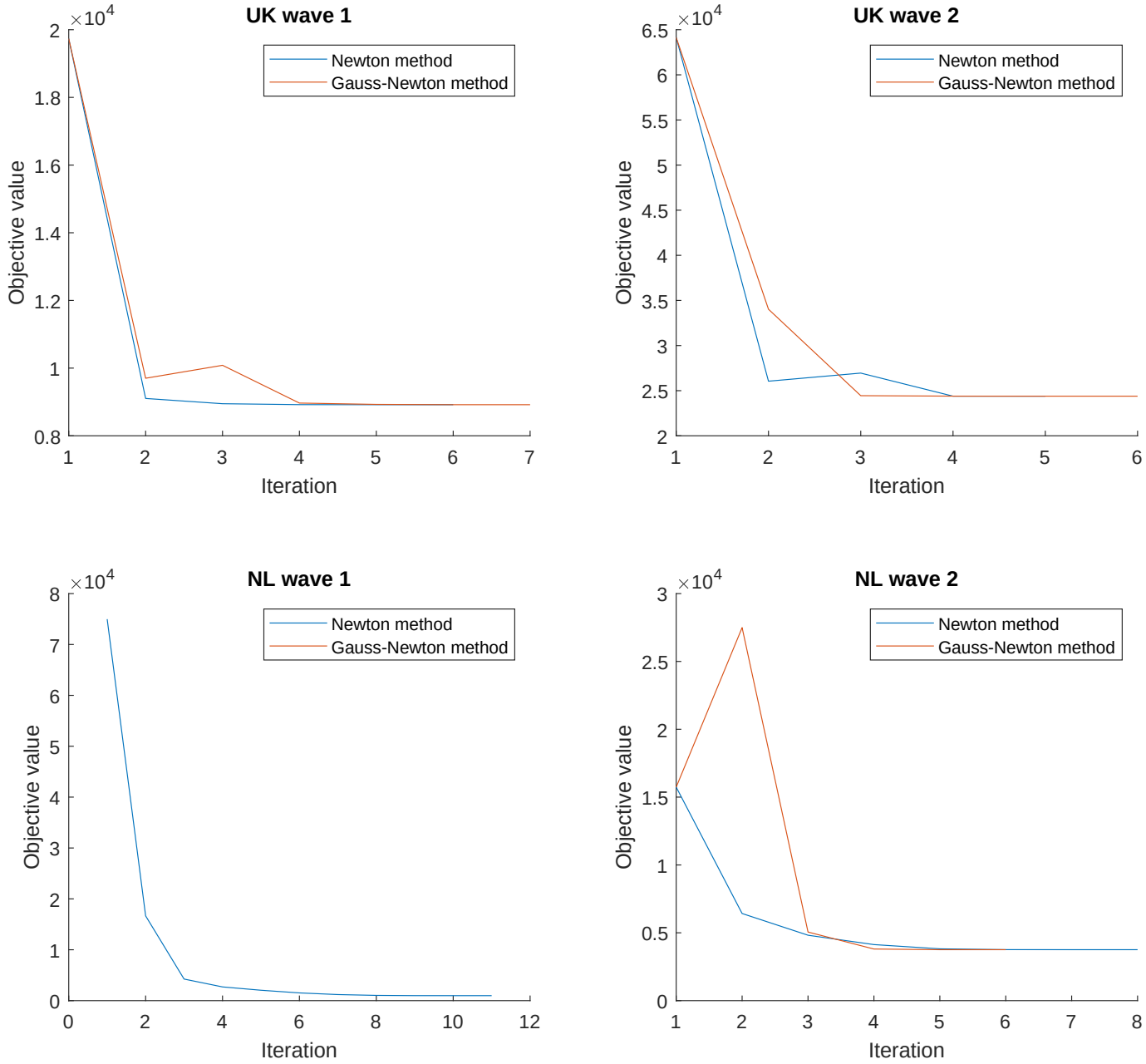


Figure 5.6: Objective value plots for each data set

## 5.3 Regions of Convergence

In this section, we briefly compare the region of convergence for the two methods. We only apply them to the UK wave 1 data set, but using the code provided in Appendix B and with some altering of starting parameters one can repeat this with the other data sets. Below we provide a figure containing a three dimensional scatter plot pertaining to the initial guesses of the three parameters, showing which points converge, whether for the Newton method, Gauss-Newton method or both.

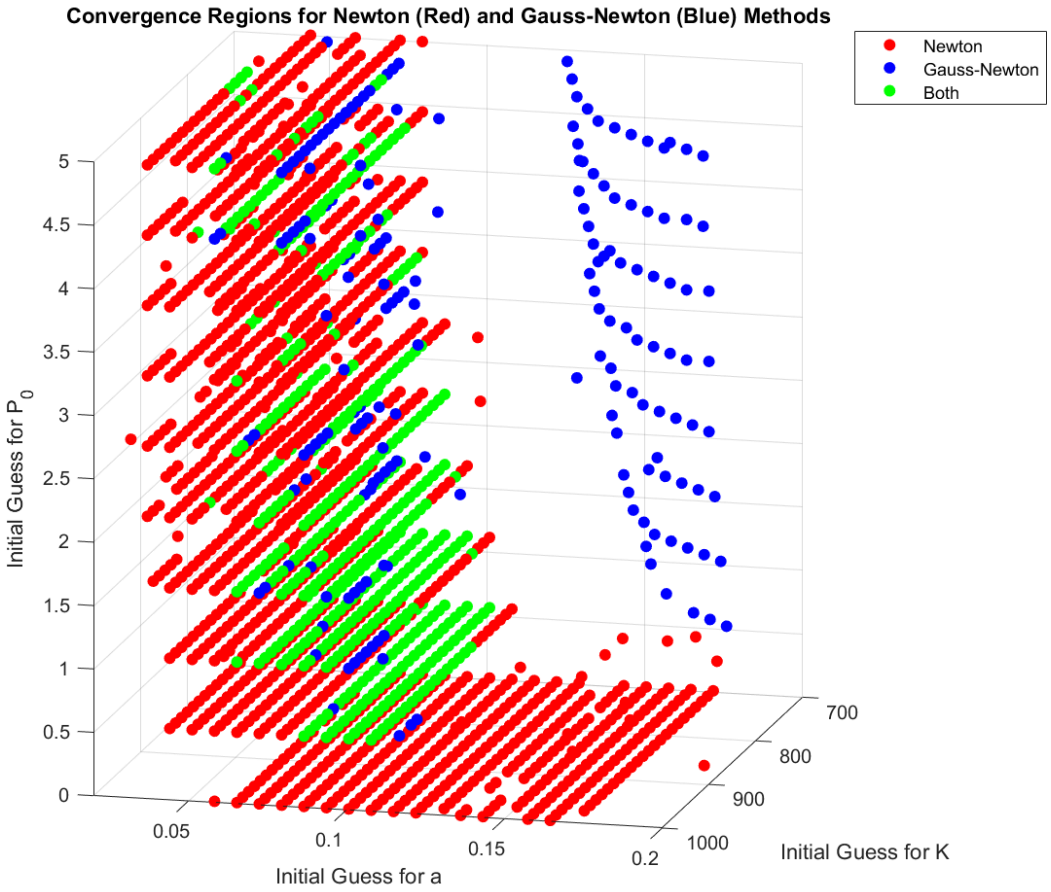


Figure 5.7: Region of convergence for UK wave 1

Of course, this does not show the exhaustive region of convergence for either algorithm. The tested region of initial parameters was:

- $a$: 25 points between 0.03 and 0.2
- $K$: 25 points between 700 and 1000
- $P_0$: 10 points between 0.01 and 5

We see that the Newton method has more initial guesses that converge than the Gauss-Newton method, which is to be expected. Naturally, we find the majority of initial guesses around the true minimum, do converge and for both methods. Interestingly, we also see that the Newton method converges better for values of $a$, lower than the true minimum, while the Gauss-Newton converges better for a few higher $a$ initialisations.

# Chapter 6

# Conclusion

At the beginning of this paper, Nonlinear Least Squares regression and the Newton method for optimization were introduced. Then, I presented a quadratic rate of convergence result under certain circumstances and discussed some of the drawbacks of the Newton method. Subsequently, two varying approaches were presented obtaining the Gauss-Newton variant of the Newton method followed by another convergence result for some satisfying conditions. Next we looked at the logistic function as a model to apply the two optimization algorithms in the case of Nonlinear Least Squares, determining the specific iterative formulas required for later implementing the methods in MATLAB. The data that were used to test convergence of the methods, were the COVID-19 cumulative deaths per million people, for the United Kingdom and the Netherlands. In this data we could clearly see the sigmoid curve being exhibited representing the different waves of the spreading of the coronavirus, making it an ideal model to test the methods. Lastly, numerical experiments were performed on the 4 data sets and the results were examined and discussed in Chapter 5.
Using the initialisation described in Section 5.1, we found that, except for one case for the Gauss-Newton method, both methods converged for each of the data sets. Furthermore, they both converged to the same minima which inspires confidence in the correctness of the implementation of the algorithms. In general, we saw that the Newton method converged slightly quicker than the Gauss-Newton method. We also witnessed that Newton had (for the tested region) a larger region of convergence compared to Gauss-Newton.
From these findings, it is clear that the Newton method would be preferred, and especially if it were developed into the damped Newton method with variable step-size. However, the main advantage of the Gauss-Newton method over the Newton method is the fact that the Hessian need not be computed in its entirety. Moreover, the Jacobian is reused in the Hessian approximation and obtained for free as it is needed as the first order derivative. Hence, the advantage of the Gauss-Newton method is that it is computationally cheaper. Although, this was neither tested nor shown in this paper since the data sets and model were small and simple respectively. Therefore, to summarize, different circumstances dictate which method would be advisable, as each has its benefits and drawbacks. But provided good initial guesses, we have seen

exceptional performance from both methods. In general, one would not expect to see results as good as the those found in this paper, given the model and the data we were estimating via nonlinear least squares regression. This fortuitous outcome should be noted when considering other models and data within nonlinear least squares regression.

# Bibliography


Ahmadi, A. A., Chaudhry, A., & Zhang, J. (2023). Higher-order newton methods with polynomial work per iteration.

Allee, W. C. (1949). *Principles of animal ecology*. W.B. Saunders, Philadelphia.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Dennis, J. E., & Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*. Philadephia: SIAM 1996.

Fletcher, R. (1988). *Practical methods of optimization*. John Wiley; Sons.

Freund, R. (2004). Newton's method for unconstrained optimization. https://ocw.mit.edu/courses/15-084j-nonlinear-programming-spring-2004/83159d56de04a7e7dc94d0348fa4ccda_lec3_newton_mthd.pdf

Gause, G. F. (1964). *The struggle for existence*. Hafner, New York.

Gilbert, W. J. (1998). Newton's method for multiple roots (chapter 50). In C. A. Pickover (Ed.), *Chaos and fractals* (pp. 327–329). Elsevier Science.

Givens, G. H., & Hoeting, J. A. (2013). *Computational statistics*. Wiley.

Gratton, S., Lawless, A. S., & Nichols, N. K. (2007). Approximate gauss-newton methods for nonlinear least squares problems. *SIAM Journal on Optimization*, *18(1)*, 106–132.

Hanzely, S., Kamzolov, D., Pasechnyuk, D., Gasnikov, A., Richtárik, P., & Takáč, M. (2022). A damped newton method achieves global $O\left(\frac{1}{k^2}\right)$ and local quadratic convergence rate.

Jensen, A. L. (1975). Comparison of logistic equations for population growth. *Biometrics, Vol. 31, No. 4*, 853–862.

Kingsland, S. (1982). The refractory model: The logistic curve and the history of population ecology. *The Quarterly Review of Biology, Vol. 57, no. 1*, 29–52 (page 30).

Levenberg, K. (1944). A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, *2(2)*, 164–168.

Lin, C.-J., Weng, R. C., & Keerthi, S. S. (2008). Trust region newton method for large-scale logistic regression. *Journal of Machine Learning Research*, *9*, 627–650.

Malthus, T. R. (1798). *An essay on the principle of population*. London, J. Johnson.

Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, *11(2)*, 431–441.

Mathieu, E., Ritchie, H., Rodés-Guirao, L., Appel, C., Giattino, C., Hasell, J., Macdonald, B., Dattani, S., Beltekian, D., Ortiz-Ospina, E., & Roser,

M. (2020). Coronavirus pandemic (covid-19) [https://ourworldindata.org/coronavirus]. *Our World in Data*.

Mishchenko, K. (2023). Regularized newton method with global $\boldsymbol{\mathcal{O}(1/k^2)}$ convergence. *SIAM Journal on Optimization*, *33(3)*, 1440–1462.

Nesterov, Y., & Polyak, B. (2006). Cubic regularization of newton method and its global performance. *Mathematical Programming*, *108(1)*, 177–205.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization*. New York: Springer.

Verhulst, P. F. (1845). *Recherches mathématiques sur la loi d'accroissement de la population*. Nouveaux Mémoires de l'Academie Royale des Sciences, des Lettres et des Beaux-Arts de Belgique 20: 1-32.

# Appendix A

# Mathematical Background

## A.1 Convergence requisites

The following results are taken from Nocedal and Wright, 2006, with pages given.

**Theorem A.1.1** (Second-Order Sufficient Conditions)**.** *(p. 16)*
*Suppose that $\nabla^2 f$ is continuous in an open neighbourhood of $x^*$ and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then $x^*$ is a strict local minimizer of $f$.*

Consider the general case of $f : \mathcal{D} \to \mathbb{R}^m$ where $\mathcal{D} \subset \mathbb{R}^n$ for general $m$ and $n$.

**Definition A.1.1** (Lipschitz Continuity)**.** *(p. 624)*
*The function $f$ is said to be Lipschitz continuous on some set $\mathcal{N} \subset \mathcal{D}$ if there is a constant $L > 0$ such that*

$$\|f(x_1) - f(x_0)\| \leq L\|x_1 - x_0\|, \quad \text{for all } x_0, x_1 \in \mathcal{N} \tag{A.1}$$

**Theorem A.1.2** (Taylor's Theorem)**.** *(p. 14)*
*Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and that $p \in \mathbb{R}^n$. Then we have that*

$$f(x + p) = f(x) + \nabla f(x + tp)^\top p, \tag{A.2}$$

*for some $t \in (0, 1)$. Moreover, if $f$ is twice continuously differentiable, we have that*

$$\nabla f(x + p) = \nabla f(x) + \int_0^1 \nabla^2 f(x + tp)p \, dt, \tag{A.3}$$

*and that*

$$f(x + p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x + tp)p, \tag{A.4}$$

*for some $t \in (0, 1)$.*

## A.2 Strong Convexity

**Definition of strong convexity and associated definitions as given in Boyd and Vandenberghe, 2004**.

We denote the *domain* of $f : \mathbb{R}^p \to \mathbb{R}^q$ as **dom** $f$. We define convexity as follows: A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if the domain of $f$ is a convex set and if for all $x$, $y \in$ domain of f, and $\theta$ with $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y). \tag{A.5}$$

Before defining strong convexity, we first briefly mention sublevel sets; the *$\alpha$-sublevel set* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$S_\alpha = \{x \in \textbf{dom } f \mid f(x) \leq \alpha\}. \tag{A.6}$$

Sublevel sets of a convex function are convex, for any value of $\alpha$.

Without loss of generality, we consider a suitable starting point $x^{(0)}$. We define a function on sublevel set

$$S = \{x \in \textbf{dom } f \mid f(x) \leq f(x^{(0)})\} \tag{A.7}$$

to be *strongly convex* if there exists an $m > 0$ such that

$$\nabla^2 f(x) \succeq mI \tag{A.8}$$

for all $x \in S$.

# Appendix B

# Extra Tables and Figures

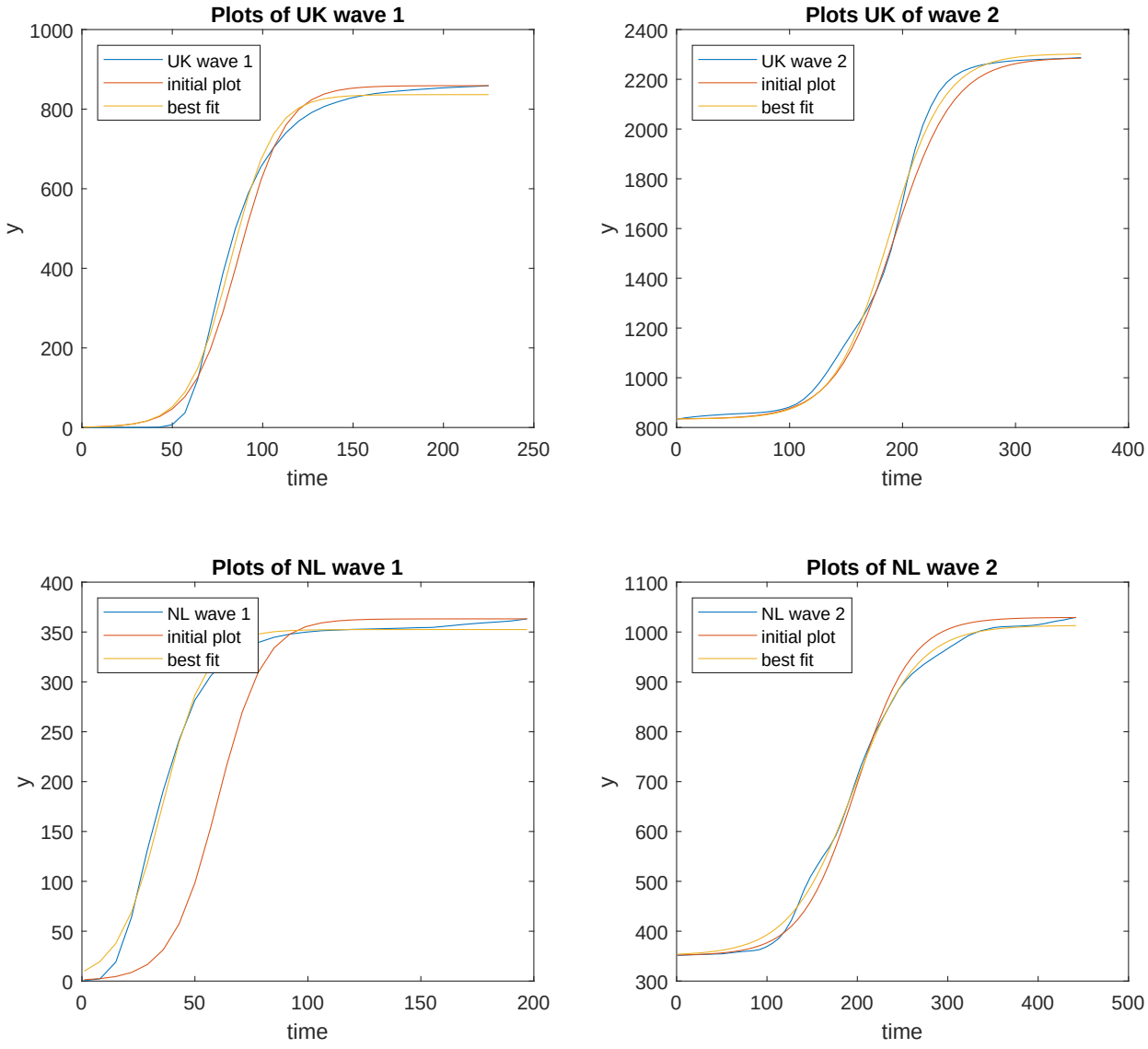


Figure B.1: Initial, best fit, and model plots for each data set

Here, I present some of the MATLAB code used in this paper to obtain the numerical results.

```
clc
clear
close all

%% Data set and method choice:
% For data, choose for dataNum 1-4 where 1,2,3,4 is
   UK_wave1, UK_wave2, NL_wave1, NL_wave2
% respectively.
% For method, choose 1 for Newton method, and 2 for Gauss
   -Newton
dataNum = 1;
method = 1;


%% Rest of code
% Extracting data
opts = detectImportOptions('total_deaths_per_million.csv'
   );
opts.VariableNamingRule = 'preserve';
table = readtable('total_deaths_per_million.csv', opts);
UK = table2array(table(:,235));
NL = table2array(table(:, 156));
daily = 0;


if daily == 1
    UK_wave1 = UK(29:259);
    UK_wave2 = UK(183:546);
    NL_wave1 = NL(64:266);
    NL_wave2 = NL(169:616);
    time_cell = {29:259, 183:546, 64:266, 169:616};
else
    UK_wave1 = UK(35:7:259);
    UK_wave2 = UK(189:7:546);
    NL_wave1 = NL(70:7:266);
    NL_wave2 = NL(175:7:616);
    time_cell_ = {35:7:259, 189:7:546, 70:7:266,
       175:7:616};
    time_cell = {1:7:231, 1:7:364, 1:7:203, 1:7:448};
end
data_cell = {UK_wave1, UK_wave2, NL_wave1, NL_wave2};



% Logistic function
phi = @(t, a, K, P_0, v) (K) ./ (1 + ((K - P_0)/ P_0) *
   exp(-a * t)) + v;
```

```
% First order derivatives
Dr = @(t, a, K, P_0) -(K.*P_0.*(P_0-K).*t.*exp(t.*a))./(P_0.*exp(t.*a)-P_0+K).^2;
DK = @(t, a, K, P_0) (P_0.^2.*(exp(a.*t)-1).*exp(a.*t))./(K+P_0.*exp(a.*t)-P_0).^2;
DP_0 = @(t, a, K, P_0) (K.^2.*exp(a.*t))./((exp(a.*t)-1).*P_0+K).^2;
% Second order derivatives
Dr2 = @(t, a, K, P_0) (K.*P_0.*(P_0-K).*t.^2.*exp(t.*a).*(P_0.*exp(t.*a)+P_0-K))./(P_0.*exp(t.*a)-P_0+K).^3;
DrDK = @(t, a, K, P_0) (P_0.^2.*t.*exp(a.*t).*((2.*exp(a.*t)-1).*K-P_0.*exp(a.*t)+P_0))./(K+P_0.*exp(a.*t)-P_0).^3;
DrDP_0 = @(t, a, K, P_0) -(K.^2.*t.*exp(a.*t).*((exp(a.*t)+1).*P_0-K))./((exp(a.*t)-1).*P_0+K).^3;
DK2 = @(t, a, K, P_0) -(2.*P_0.^2.*(exp(a.*t)-1).*exp(a.*t))./(K+P_0*exp(a.*t)-P_0).^3;
DP_0DK = @(t, a, K, P_0) (2.*P_0.*(exp(a.*t)-1).*exp(a.*t).*K)./(K+P_0.*exp(a.*t)-P_0).^3;
DP_02 = @(t, a, K, P_0) -(2.*K.^2.*(exp(a.*t)-1).*exp(a.*t))./((exp(a.*t)-1).*P_0+K).^3;

% Jacobian, nabla_r, and nabla2_r
Jac = @(t, a, K, P_0) [-Dr(t, a, K, P_0)', -DK(t, a, K, P_0)', -DP_0(t, a, K, P_0)'];
r_j = @(y, t, a, K, P_0,v) y - phi(t,a,K,P_0,v);
nabla2_r_j = @(t, a, K, P_0) [-Dr2(t,a,K,P_0), -DrDK(t,a,K,P_0), -DrDP_0(t,a,K,P_0); -DrDK(t,a,K,P_0), -DK2(t,a,K,P_0), -DP_0DK(t,a,K,P_0); -DrDP_0(t,a,K,P_0), -DP_0DK(t,a,K,P_0), -DP_02(t,a,K,P_0)];


% Data-specific parameters for starting:
time = time_cell{dataNum};
data = data_cell{dataNum};
v = data(1);
x_k = [0; data(end)-v; 1];

m = length(time);
time_m = time(floor(m/2));
median_ind = floor(m/2);
median = data(median_ind);
a = (1/time_m)*log(((x_k(2) - x_k(3))* (median - v))/(x_k(2) - median + v) * x_k(3)));
x_k(1) = a;

% Algorithm-specific paramters:
tolerance = 0.1;

```

```
% Plot flag: 1 to finish plots, 0 to stop (due to
    divergence)
plots = 1;

figure;
subplot(2,2,1);
plot(time, data)

initial = @(time) phi(time, x_k(1), x_k(2), x_k(3), v);
title('Initial plot NL wave 1')
xlabel('time')
ylabel('y')
hold on
plot(time, initial(time))
legend({'NL wave 1', 'model'}, 'Location', 'northwest')



subplot(2,2,2);
plot(time, phi(time, x_k(1), x_k(2), x_k(3), v), 'red')
title('Model convergence')
xlabel('time')
ylabel('y')
hold on
plot(time, initial(time))
residual = data - phi(time, x_k(1), x_k(2), x_k(3), v)';
objective = 0.5*sum(residual.^2);


% Following for-loop performs the iterative algorithms,
    with stopping
% conditions.
parameters = [];
counter = 0;
while 1
    counter = counter + 1;
    % Saving each iteration of parameters
    parameters = [parameters, x_k];
    if method == 1
        [x_k, singular_check] = N_iteration(time, data,
            x_k, v, phi, Jac, r_j, nabla2_r_j, tolerance);
    else
        [x_k, singular_check] = GN_iteration(time, data,
            x_k, v, phi, Jac);
    end
    residual = data - phi(time, x_k(1), x_k(2), x_k(3), v
        )';
    objective_residual = 0.5*sum(residual.^2);
    objective = [objective, objective_residual];
```

```
121     change = abs((objective(counter+1) - objective(
            counter)) / objective(counter));
122     change2 = (objective(counter) - objective(counter+1))
            ;
123     if counter ~= 1 && change < 10^-4
124         fprintf('Number of iterations: %i \n', counter)
125         fprintf('Objective value: %f \n', objective(end))
126         break
127     elseif counter >= 50
128         fprintf('Exceeded 50 iterations \n')
129         break
130     elseif any(isnan(singular_check), 'all')
131         fprintf('Singular Hessian approximation at
                iteration %i \n', counter)
132         break
133     elseif change2 < -10^7
134         fprintf('Divergent case at iteration %i \n',
                counter)
135         plots = 0;
136         break
137     end
138
139     hold on
140     plot(time, phi(time, x_k(1), x_k(2), x_k(3), v))
141
142 end
143
144 if plots == 1
145     %residuals
146     subplot(2,2,3);
147     r_xk = data - phi(time, x_k(1), x_k(2), x_k(3), v)';
148     histogram(r_xk, 12)
149     title('Residuals histogram')
150     xlabel('Residuals')
151     ylabel('frequency')
152     subplot(2,2,4);
153     plot(1:counter+1, objective)
154     title('Objective value at each iteration')
155     xlabel('Iteration')
156     ylabel('Objective value')
157
158     % Final plot
159     figure;
160     plot(time, data)
161     title('Final plot')
162     xlabel('time')
163     ylabel('y')
164
165     final = @(time) phi(time, x_k(1), x_k(2), x_k(3), v);
```

```
166     hold on
167     plot(time, final(time))
168     legend({'NL wave 1', 'best fit'}, 'Location','
           northwest')
169
170     % Plot showing quadratic convergence
171     distances = parameters - repmat(x_k,1,length(
           parameters));
172     norms = zeros(1, length(parameters));
173     for i=1:length(parameters)
174         norms(i) = norm(distances(:,i),2);
175     end
176     figure;
177     plot(1:counter, norms)
178     title('Distance to x* at each iteration')
179     xlabel('Iteration')
180     ylabel('Distance to x*')
181 end
182
183
184 function [x_k1, singular_check] = GN_iteration(t, y, x_k,
        v, phi, J)
185     % Function that takes time vector t, data vector y,
           vector of
186     % parameters at current iterate x_k, vertical shift
           parameter v,
187     % function of logistic model phi and Jacobian
           function J. It computes
188     % the new set of parameter iterates and returns also
           the singularity
189     % check of the approximate Hessian of f.
190
191     r_xk = y - phi(t, x_k(1), x_k(2), x_k(3), v)';
192
193     singular_check = inv(J(t, x_k(1), x_k(2), x_k(3))' *
           J(t, x_k(1), x_k(2), x_k(3)));
194     singular_check2 = cond(J(t, x_k(1), x_k(2), x_k(3))'
           * J(t, x_k(1), x_k(2), x_k(3)));
195
196     x_k1 = x_k - (J(t, x_k(1), x_k(2), x_k(3))' * J(t,
           x_k(1), x_k(2), x_k(3))) \ (J(t, x_k(1), x_k(2),
           x_k(3))' * r_xk);
197
198 end
199
200 function [x_k1, singular_check] = N_iteration(t, y, x_k,
       v, phi, J, r_j, nabla2_r_j, tolerance)
201     % Function that takes time vector t, data vector y,
           vector of
```

```
    % parameters at current iterate x_k, vertical shift parameter v,
    % function of logistic model phi, Jacobian function J,
    % residual function (for observation j) r_j, Hessian of the
    % residuals function (for observation j) nabla2_r_j, and tolerance
    % parameter tolerance. Computes the new set of parameter iterates,
    % checking the singularity of the Hessian of f. It returns the new
    % iterates and the singularity check.

    r_xk = y - phi(t, x_k(1), x_k(2), x_k(3), v)';
    rr_xk = kron(eye(3), r_xk);

    nabla_f_sum = zeros(3,3);
    for i=1:length(t)
         nabla_f_sum = nabla_f_sum + r_j(y(i), t(i), x_k(1), x_k(2), x_k(3), v).*nabla2_r_j(t(i), x_k(1), x_k(2), x_k(3));
    end

    nabla_f = J(t, x_k(1), x_k(2), x_k(3))' * r_xk;
    nabla2_f = J(t, x_k(1), x_k(2), x_k(3))' * J(t, x_k(1), x_k(2), x_k(3)) + nabla_f_sum;
    singular_check = inv(nabla2_f);
    singular_check2 = cond(nabla2_f);
    decrement = nabla_f' * (nabla2_f \ nabla_f);
    if abs(decrement/2) < tolerance
        x_k1 = x_k;
        return
    end

    newton_step = - inv(nabla2_f) * nabla_f;
    x_k1 = x_k + newton_step;
end
```

Listing B.1: MATLAB code 2

The following code is used to determine the region of convergence plot, some parts required in the code have been omitted as they can be taken from the code given above.

```
clc
clear
close all

%% Data set and method choice:
```

```
% For data, choose for dataNum 1-4 where 1,2,3,4 is
    UK_wave1, UK_wave2, NL_wave1, NL_wave2
% respectively.
% For method, choose 1 for Newton method, and 2 for Gauss
    -Newton

newton_convergence = zeros(25, 25, 10);
gauss_newton_convergence = zeros(25, 25, 10);
for x=1:2
    dataNum = 1;
    method = x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Use the same code here as from code 1 above
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Data-specific parameters for starting:
    time = time_cell{dataNum};
    data = data_cell{dataNum};
    v = data(1);
    x_k = [0; data(end)-v; 1];


    % Algorithm-specific paramters:
    tolerance = 0.1;


    % Initial guesses test region
    initial_guesses_a = linspace(0.03, 0.2, 25);
    initial_guesses_K = linspace(700, 1000, 25);
    initial_guesses_P0 = linspace(0.01, 5, 10);

    residual = data - phi(time, x_k(1), x_k(2), x_k(3), v
       )';
    objective = 0.5*sum(residual.^2);


    for i=1:25
        for j=1:25
            for k=1:10
                counter = 0;
                x_k = [initial_guesses_a(i);
                   initial_guesses_K(j);
                   initial_guesses_P0(k)];
                while 1
                    counter = counter + 1;
```

```
50
51                  if method == 1
52                      [x_k, singular_check] =
                            N_iteration(time, data, x_k, v
                            , phi, Jac, r_j, nabla2_r_j,
                            tolerance);
53                  else
54                      [x_k, singular_check] =
                            GN_iteration(time, data, x_k,
                            v, phi, Jac);
55                  end
56                  residual = data - phi(time, x_k(1),
                        x_k(2), x_k(3), v)';
57                  objective_residual = 0.5*sum(residual
                        .^2);
58                  objective = [objective,
                        objective_residual];
59                  change = abs((objective(counter+1) -
                        objective(counter)) / objective(
                        counter));
60                  change2 = (objective(counter) -
                        objective(counter+1));
61                  if counter ~= 1 && ((8918 <
                        objective_residual) && (
                        objective_residual < 9000))
62                      fprintf('Number of iterations: %i
                            \n', counter)
63                      fprintf('Objective value: %f \n',
                            objective(end))
64                      converged = 1;
65                      break
66                  elseif counter >= 50
67                      fprintf('Exceeded 50 iterations \
                            n')
68                      converged = 0;
69                      break
70                  elseif any(isnan(singular_check), '
                        all')
71                      fprintf('Singular Hessian
                            approximation at iteration %i
                            \n', counter)
72                      converged = 0;
73                      break
74                  end
75              end
76              if method == 1
77                  newton_convergence(i,j,k) = converged
                        ;
78              else
```

```
                    gauss_newton_convergence(i,j,k) =
                        converged;
                end
            end
        end
    end
end
% Visualize convergence regions in 3D
[x_param1, y_param2, z_param3] = meshgrid(
    initial_guesses_K, initial_guesses_a,
    initial_guesses_P0);


both_converged = newton_convergence &
    gauss_newton_convergence;
figure;
scatter3(x_param1(newton_convergence==1 & ~both_converged
    ), y_param2(newton_convergence==1 & ~both_converged),
    z_param3(newton_convergence==1 & ~both_converged), 'r'
    , 'filled');
hold on;
scatter3(x_param1(gauss_newton_convergence==1 & ~
    both_converged), y_param2(gauss_newton_convergence==1
    & ~both_converged), z_param3(gauss_newton_convergence
    ==1 & ~both_converged), 'b', 'filled');
hold on;
scatter3(x_param1(both_converged), y_param2(
    both_converged), z_param3(both_converged), 'g', '
    filled');
title('Convergence Regions for Newton (Red) and Gauss-
    Newton (Blue) Methods');
ylabel('Initial Guess for a');
xlabel('Initial Guess for K');
zlabel('Initial Guess for P_0');
legend('Newton', 'Gauss-Newton', 'Both');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   ... Functions from code 1
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing B.2: MATLAB code 2