Semi-parametrische Dichteschätzung Minimaler Hemm-Konzentrationen

Masterarbeit

vorgelegt von

Laura Reuter

angefertigt am
Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
im Studiengang Wirtschaftsmathematik

betreut von
Prof. Dr. Markus Bibinger

Marburg, den 3. April 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

AIC	Akaike Informationskriterium
B-Spline	Basis-Spline
CLM	Composite Link Model
ECOFF	Epidemiological cut-off
E. coli	Escherichia coli
EUCAST	European Committee on Antimicrobial Susceptibility Testing
GLM	Generalized linear model
MALA	Metropolis-adjusted Langevin algorithm
MCMC-Methode	Markov-Chain-Monte-Carlo-Methode
MIC	Minimum Inhibitory Concentration
MSE	Mean Squared Error

1 Einleitung

Der Einfluss des Menschen auf die Umwelt ist nicht zu übersehen. Seine Kontrolle über einen Großteil des verfügbaren Süßwassers sowie der Wald- und Nutzflächen hat direkte evolutionäre Auswirkungen auf viele Spezies. Beispielsweise ist zu beobachten, dass Fische in stark befischten Gebieten mit der Zeit einen schmaleren Körperbau entwickeln, um Fangnetzen besser entgehen zu können. Solche Veränderungen entsprechend der *natürlichen Selektion* sind jedoch nicht nur bei Tieren zu beobachten, sondern vor allem auch bei Bakterien und Viren. Sobald Bakterien und Viren resistent gegen Medikamente werden, stellt dies eine ernstzunehmende Gefahr für die menschliche Gesundheit dar.
Resistenzen entwickeln sich unter anderem durch die ineffiziente und großflächige Nutzung von Antibiotika und Pestiziden auf Ackerflächen sowie die oftmals unnötige ärztliche Verschreibung von Antibiotika und den vorzeitigen Behandlungsabbruch durch die Patienten. Im Falle einer nicht vollendeten Behandlung überleben einige Erreger und können durch Mutationen resistent gegen das verwendete Medikament werden und sich so ungehindert vermehren.
Seit der zufälligen Entdeckung des Penicillin im Jahre 1928 durch Sir Alexander Fleming[1] ist ein Großteil der Krankheiten, die durch das Bakterium *Staphylococcus aureus* verursacht werden, Penicillin-resistent geworden. Erkrankungen, die einst mit geringen Dosen dieses Medikaments behandelt werden konnten, benötigen nun eine sehr viel höhere Dosis oder ein stärkeres Mittel. Diese Alternativen helfen jedoch nur für eine begrenzte Dauer, da bereits im Jahr 2001 bis zu 50 % der gegen Penicillin resistenten Bakterien auch gegen das stärkere Mittel *Methicillin* Resistenzen entwickelt haben (Beispiel: MRSA: Methicillin-resistenter Staphylococcus aureus).
Diese Entwicklung verursacht so hohe Kosten für Entwicklung, Tests und Anwendung neuer Medikamente, dass eine effektive Therapie eigentlich behandelbarer Krankheiten in naher Zukunft für viele Menschen ökonomisch gesehen nicht mehr erreichbar sein wird.[2]
Zur Untersuchung von Antibiotika-Resistenzen wird die *Minimale Hemm-Konzentration* (basierend auf dem englischsprachigen Begriff *Minimum Inhibitory Concentration*, im Folgenden als MIC abgekürzt) angegeben, die als die geringste Konzentration eines Antibiotikums definiert ist, welche das sichtbare Wachstum eines Mikroorganismus über Nacht eindämmt. Die Messung der MIC wird bei neuen Medikamenten außerdem zur Untersuchung der *in vitro* Aktivität verwendet.[3]
Es ist von besonderer Wichtigkeit, die Verteilung der Bakterien zu bestimmen, da un-

[1] vgl. Chemie.de [online].
[2] vgl. Palumbi, S. R. (2001), S. 1786 ff.
[3] vgl. Andrews, J. M. (2001), S. 5.

erkannte Resistenzen große Gefahren für die menschliche Gesundheit bergen. Deshalb müssen Erkenntnisse über den resistenten Anteil gewonnen werden, die zur Entwicklung neuer Therapien genutzt werden können.

1.1 Motivation der Arbeit

MICs werden zur Messung der Reaktion von Bakterien auf Antibiotika verwendet. Diese Informationen sind in Hinblick auf die Behandlung von Krankheiten sehr wichtig, um die richtige Therapie wählen zu können. Mithilfe der ermittelten Kennzahlen kann die Verteilung der Bakterien bestimmt werden sowie die Bakterienkultur in die Anteile *Wild-Type* und *Non-Wild-Type* kategorisiert werden. *Wild* bezeichnet in diesem Fall denjenigen Teil der Bakterienpopulation, der keine Resistenzen entwickelt hat und somit in seiner „wilden“ bzw. ursprünglichen Form vorliegt. *Non-wild* beschreibt entsprechend den Teil, der resistent gegen das untersuchte Medikament ist.
Das Vorgehen zur Messung der MIC ist standardisiert: Zunächst wird definiert, welchen Umfang das *Dilution-Experiment* haben soll. Es werden dann beispielsweise 13 gleiche Behälter verwendet, in die jeweils eine festgelegte Menge des Isolats (Bakterium in seiner reinen Form) gegeben wird. Ein Behälter bleibt dabei frei von Antibiotika, um das Wachstum ohne Behandlung beobachten zu können. Die anderen Behälter werden mit Antibiotikum-Konzentrationen von $^1/_8$ mg/l, $^1/_4$ mg/l, $^1/_2$ mg/l bis 512 mg/l behandelt, wobei die Konzentration jeweils verdoppelt wird.
Nach einer definierten Inkubationszeit wird das Experiment ausgewertet: Die MIC ist definiert als geringste Konzentration, bei der kein sichtbares Wachstum stattgefunden hat.[4] Wenn beispielsweise ein Wachstum in den Proben unterhalb einer Konzentration von 2 mg/l beobachtet werden kann und in allen Proben ab dieser kein Wachstum aufgetreten ist, wird 2 mg/l als MIC-Wert festgelegt. Die wahre Konzentration, ab der kein Wachstum mehr stattfindet, wird jedoch im Bereich zwischen 1 mg/l und 2 mg/l liegen. Die wahren MIC-Werte liegen also in Intervallen vor und sind somit *Intervallzensiert*.
Die in dieser Arbeit verwendeten Daten stammen von dem *European Committee on Antimicrobial Susceptibility Testing* (EUCAST), das sich seit dem Jahr 1997 mit Breakpoints von MICs befasst und dafür die Ergebnisse von Experimenten aus weltweiten Quellen vereinheitlicht und aufbereitet.[5]

In der vorliegenden Arbeit wird die Kombination Ampicillin gegen Escherichia coli (E. coli) betrachtet. „Escherichia coli ist ein natürlich vorkommender Keim [...] im Darm

[4] vgl. Andrews, J. M. (2001), S. 10.
[5] vgl. EUCAST (2016) [online].

von Vögeln und warmblütigen Säugetieren. Ebenso ist er Bestandteil der Darmflora des Menschen. Bestimmte Stämme von Escherichia coli können bei Tieren und Menschen schwerwiegende Erkrankungen hervorrufen.“[6]
E. coli-Bakterien gelten als die häufigsten Verursacher von bakteriellen Harnwegsinfektionen sowie Blutvergiftungen und Krankenhausinfektionen. In Ländern warmer Klimazonen mit geringem Hygienestandard sind E. coli-Bakterien außerdem für viele Erkrankungen des Magen-Darm-Traktes des Menschen verantwortlich. Der E. coli-Stamm EHEC beispielsweise kommt in der Natur im Darm von Rindern, Schafen und Ziegen vor und hat im Jahr 2011 in Deutschland zu einem großen EHEC-Ausbruch geführt, „der von mit EHEC-Keimen verunreinigten Bockshornkleesamen, die in der Sprossenproduktion eingesetzt wurden“[7], verursacht wurde.

Die folgende Abbildung zeigt die verwendeten Daten.

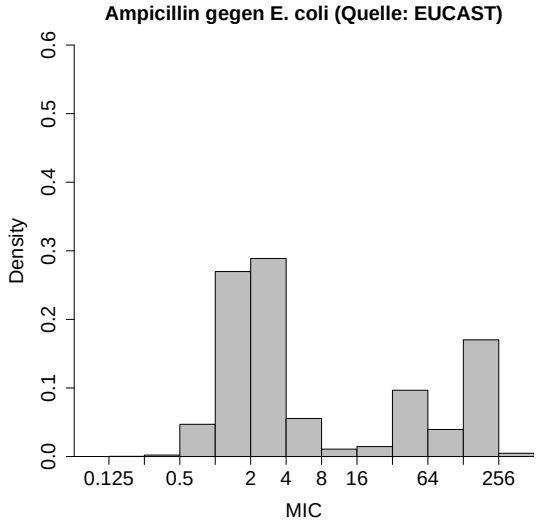


Abbildung 1.1: Ampicillin gegen E. coli.[8]

Der Wild-Type-Anteil der Population ist hier, wie in den meisten Fällen, im linken Bereich der MIC-Verteilung zu finden, wobei man davon ausgeht, dass dieser keine Resistenzen gegen das Medikament aufweist. Dieser Teil kann durch eine parametrische unimodale Verteilung, wie beispielsweise eine Lognormal- oder Gammaverteilung, modelliert werden.[9] Der Non-Wild-Teil ist auf der rechten Seite angesiedelt und ist oft multimodal, wobei angenommen wird, dass dieser Teil wiederum aus zwei oder mehr verschiedenen Non-Wild-Populationen besteht, deren entwickelte Resistenz unterschiedlich ausgeprägt ist.[10] Für die Schätzung des Non-Wild-Anteils wird häufig ein

[6] BfR (2016) [online].
[7] Ebenda [online].
[8] eigene Darstellung in Anlehnung an MIC-EUCAST (2017).
[9] vgl. Turnidge, J. et al. (2006), S. 419 ff.
[10] vgl. Jaspers, S. et al. (2014a), S. 290.

nicht-parametrischer Ansatz gewählt.
Ein Isolat wird der Klasse Wild bzw. Non-Wild auf Basis des sogenannten *epidemiological cut-off value* (ECOFF) zugeordnet. Dieser cut-off-Wert ist die obere Schranke der Wild-Type-Verteilung und wird in den meisten Fällen durch visuelle Betrachtung des Histogramms des Dilution-Experimentes abgelesen. Für eine modellbasierte Einordnung des Isolats jedoch wird dieses dem Wild-Type-Anteil mithilfe des 99,9 %-Quantils der parametrischen Verteilung zugeordnet. Deshalb ist es wichtig, einen semi-parametrischen Ansatz der Dichteschätzung zu wählen.

1.2 Zielsetzung und Aufbau

Jaspers, S. et al. (2016) stellt in „*A Bayesian approach to the semi-parametric estimation of a MIC distribution*“ eine Methode zur Schätzung einer MIC-Verteilung vor.[11] Dazu wird ein Ansatz verfolgt, der in Lambert, P., Eilers, P. H. C. (2009) beschrieben wird und Bayes-Methoden zur Schätzung der Verteilung verwendet.[12] Der Bayes-Ansatz zur Schätzung gruppierter stetiger Daten wird erweitert, indem ein Strafterm zur Glättung der geschätzten Verteilung eingeführt wird.
Das Ziel dieser Arbeit ist es, die in Jaspers, S. et al. (2016) beschriebene Methode nachzuvollziehen und an den von EUCAST vorliegenden Daten anzuwenden. Dazu werden im ersten Schritt benötigte mathematische Grundlagen wie B-Splines, das Composite Link Model und nicht-lineare Regression bzw. insbesondere das Verfahren von Turnidge, J. et al. (2006) erklärt. In Kapitel 3 wird das Vorgehen von Jaspers, S. et al. (2016) erläutert und die Erweiterung des Modells dargelegt. Kapitel 4 befasst sich mit Markov-Chain-Monte-Carlo-Methoden, die zur Stichprobenerzeugung von Verteilungen verwendet werden können. Dafür werden zunächst allgemein Markov-Ketten sowie die Acceptance-Rejection-Methode eingeführt und darauf aufbauend der Metropolis-Hastings- und der Langevin-Hastings-Algorithmus detailliert betrachtet, welche dann in Kapitel 5 angewendet werden. Dort werden die vorliegenden Daten beschrieben und beide Bestandteile der Verteilung geschätzt. In Kapitel 5.4 wird außerdem eine alternative Berechnung der Gesamtdichte aus den zwei einzelnen Teilen vorgestellt. In Kapitel 6 wird schließlich die entwickelte Methode auf simulierte Daten angewendet, um die Performance zu bewerten. Eine Zusammenfassung inklusive Ausblick sind in Kapitel 7 zu finden. Alle Berechnungen wurden mit der Statistiksoftware `R` durchgeführt und sind im Anhang in Kapitel 8 mit kurzen Erklärungen zusammengefasst.

[11] vgl. Jaspers, S. et al. (2016).
[12] vgl. Lambert, P., Eilers, P. H. C. (2009).

2 Mathematische Grundlagen

Im folgenden Kapitel werden die für diese Arbeit relevanten mathematischen Grundlagen eingeführt. Der erste Teil widmet sich der Theorie von Splines sowie den sogenannten B-Splines. Im zweiten Teil werden die Grundlagen der nicht-linearen Regression wie auch das Verfahren von Turnidge, J. et al. (2006) vorgestellt, welches im praktischen Teil dieser Arbeit angewendet wird. Das Composite Link Model, mit dem Verteilungen aus gruppierten Daten geschätzt werden können, wird im abschließenden Teil dieses Kapitels erläutert.

2.1 B-Splines als Hilfsmittel zur Dichteschätzung

Im Folgenden werden Splines als Möglichkeit zur Schätzung stetiger Kurven aus diskretisierten Daten erläutert. Dazu wird zunächst die allgemeine Theorie der Spline-Approximation eingeführt. Anschließend folgt die Beschreibung von B-Splines, die einen Spezialfall von Splines darstellen. Zuletzt wird das sogenannte Bias-Varianz-Dilemma veranschaulicht, das bei Dichteschätzungen vorliegt.

Allgemeines Konzept von Splines

Es seien n Paare (x_i, y_i) gegeben, die durch eine glatte Kurve verbunden werden sollen. Gesucht ist also eine *interpolierende Kurve*, die an allen *Knoten* x_i die Werte y_i annimmt. Ein Ansatz dafür ist, ein Polynom mit der Form $p_n(x) := \sum_{i=0}^{n-1} a_i x^i$ zu definieren, das *Interpolationspolynom von Grad n*, für welches an allen Stützstellen $p_n(x_i) = y_i$ gilt. Man erwartet, dass das Interpolationspolynom die zugrundeliegende Funktion, durch die sich die Stützstellen mit $f(x_i) = y_i$ ergeben, auch an allen Zwischenpunkten gut approximiert. Eine weitere Vermutung in diesem Zusammenhang ist, dass die Approximation desto besser ist, je mehr Stützstellen es gibt. Diese Erwartungen werden jedoch im Allgemeinen nicht erfüllt, wie das sogenannte *Runge-Phänomen* zeigt: Für die Interpolation der Runge-Funktion $f(x) = \frac{1}{1+x^2}$ im Bereich $x \in [-5, 5]$ mit $n + 1$ äquidistanten Stützpunkten kann gezeigt werden, dass[13]

$$\lim_{n\to\infty} \max_{x\in[-5,5]} |p_n(x) - f(x)| = \infty.$$

Die Interpolation der glatten Runge-Funktion oszilliert extrem, besonders am Rand des Intervalls. Dieses Gegenbeispiel zeigt, dass eine Interpolation mit hohem Grad n und sehr vielen Stützstellen im Allgemeinen keine gute Approximation an die zugrundeliegende Funktion darstellt. Um dieses Problem zu umgehen, soll die zu approximierende

[13] vgl. Runge, C. (1901), S. 243.

Funktion f global interpoliert werden. Dazu kann eine Interpolation mit lokalen und stückweisen Polynomen vorgenommen werden, die als *Spline-Funktionen* oder kurz *Splines* bezeichnet werden. Ein Spline $S(x)$ ist somit eine Funktion aus einzelnen Polynomen, die zu einer glatten Kurve zusammengesetzt sind und alle Knotenpunkte enthalten. Ein Spline von Grad 1 ist eine stückweise lineare Funktion und ist in Abbildung 2.1 mit einer grünen Linie dargestellt. Außerdem zeigt die Abbildung einen Spline von Grad 3 (blaue Kurve), der aus kubischen Polynomen zusammengesetzt ist. Man kann erkennen, dass die Richtungsänderungen der Funktion von Grad 1 an den Knoten sehr abrupt sind und keine glatte Kurve vorliegt. Für Splines ab Grad 3 ist der Kurvenverlauf glatt, weshalb kubische Splines für die meisten Anwendungen verwendet werden.[14]

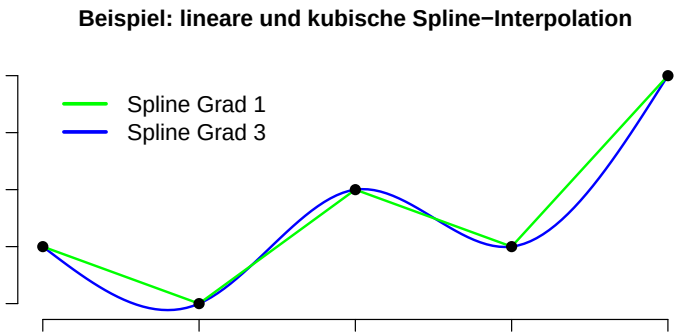


Abbildung 2.1: Lineare und kubische Spline-Interpolation.[15]

B-Splines als effiziente Möglichkeit zur Dichteschätzung

Mithilfe numerischer Methoden Splines zu bestimmen, ist sehr aufwendig. Aus diesem Grund wurden verschiedene Spline-Funktionen entwickelt, um die Berechnung effizienter zu gestalten. Eine Möglichkeit sind die sogenannten *Basis-Splines* bzw. kurz *B-Splines*, welche eine Basis der Menge der Splines darstellen, die zur Interpolation verwendet werden.[16]

Sei dazu $(t_i)_{i=0,\dots,n}$ eine Menge von Knoten mit $t_0 < t_1 < \cdots < t_n$. Der i-te B-Spline von Grad 0 ist definiert als

$$B_i^0(x) = \begin{cases} 1 & t_i \leq x < t_{i+1} \\ 0 & \text{sonst}\,. \end{cases}$$

[14] vgl. Efromovich, S. (1999), S. 343 ff.
[15] eigene Darstellung.
[16] vgl. Efromovich, S. (1999), S. 346.

Mit $B_i^0(x)$ als Startfunktion erhält man B-Splines höheren Grades $q \geq 1$ rekursiv mit folgender Formel:[17]

$$B_i^q(x) = \frac{x - t_i}{t_{i+q} - t_i} B_i^{q-1}(x) + \frac{t_{i+q+1} - x}{t_{i+q+1} - t_{i+1}} B_{i+1}^{q-1}(x). \tag{2.1}$$

Daraus folgt, dass ein Basis-Spline von Grad q aus $q+1$ Polynomen von jeweils Grad q besteht und über $q+2$ Knoten verbunden ist. Somit enthält ein B-Spline von Grad q exakt q Knoten.[18] In Abbildung 2.2 sind B-Splines der Grade 1 bis 4 dargestellt. Es ist zu erkennen, dass die Basis-Funktionen nur auf ihrem Träger positiv sind und sonst auf der Nulllinie verlaufen.

In der vorliegenden Arbeit werden B-Splines von Grad 3 verwendet, daher setze $B_i^3(x) := B_i(x)$.

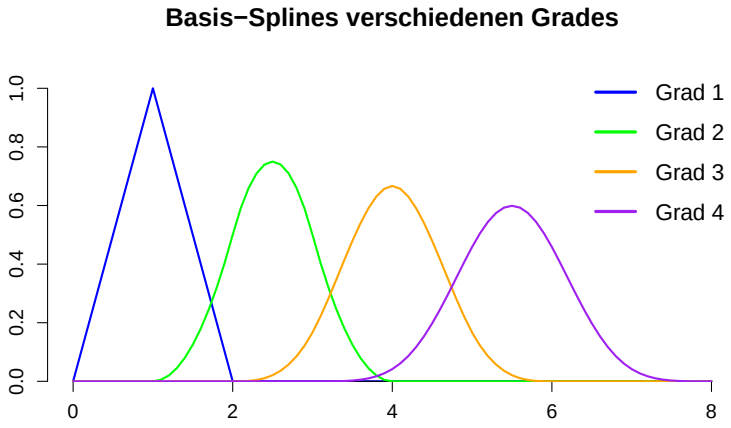


Abbildung 2.2: B-Splines von Grad 1 bis Grad 4.[19]

Sei nun also $B_i(x)$ der Wert des i-ten B-Splines an der Stelle x auf einem Raster von äquidistanten Knoten. Die an die Werte (x_i, y_i) angepasste Kurve ergibt sich aus der Linearkombination von B-Splines

$$\hat{f}(x) = \sum_{i=1}^{n} \phi_i B_i(x)$$

mit Koeffizienten-Vektor $\phi = (\phi_1, \ldots, \phi_n)$.

Die Berechnung der B-Splines zeigt Quellcode 2.1. Dort wird zunächst die Schrittweite der Stützstellen berechnet, wobei dieser Vektor um $q+1$ Stellen verlängert wird, um

[17] vgl. Boor, C. de (1993), S. 2.
[18] vgl. Eilers, P. H. C., Marx, B. D. (1996), S. 90.
[19] eigene Darstellung.

auch am Rand des Intervalls die Berechnung durchführen zu können. Danach werden entsprechend Formel (2.1) die B-Splines bestimmt.

```
B <- function(x, i, q, t)
{
  delta <- t[2]-t[1]
  t <- c(t, max(t)+delta*1:(q+1))

  if(q == 0)
    (1) * (x >= t[i] & x < t[i+1]) + (0) * (x >= t[i+1] & x < t[i])
  else
    (x-t[i])/(t[i+q] - t[i])*B(x, i, q-1, t) +
      (t[i+q+1] - x)/(t[i+q+1]-t[i+1]) * B(x, i+1, q-1, t)
}
```

Quellcode 2.1: Berechnung der B-Spline-Basis.[20]

Die Funktion $\hat{f}(x)$ soll eine Dichte repräsentieren. Deshalb muss $\int \hat{f}(x)\ \mathrm{d}x = 1$ gelten, wodurch sich für die Spline-Koeffizienten folgende Bedingung ergibt:

$$\begin{aligned} 1 = \int \hat{f}(x)\mathrm{d}x &= \int \sum_{i=1}^{n} \phi_i B_i(x)\ \mathrm{d}x \\ &= \sum_{i=1}^{n} \int \phi_i B_i(x)\ \mathrm{d}x \\ &= \sum_{i=1}^{n} \phi_i \underbrace{\int B_i(x)\ \mathrm{d}x}_{=1,\ B_i(x)\text{normiert}} \\ &= \sum_{i=1}^{n} \phi_i. \end{aligned} \tag{2.2}$$

[21]

Bias-Varianz-Dilemma

Bei der Dichteschätzung herrscht das sogenannte *Bias-Varianz-Dilemma*. Bei einer Annäherung $\hat{f}$ an f sollen sowohl die Verzerrung bzw. der Bias $\left(\hat{f}(x_i)\right) = \mathrm{E}\left[\hat{f}(x_i) - f(x_i)\right]$ als auch die Varianz $\mathrm{Var}\left(\hat{f}(x_i)\right) = \mathrm{E}\left[\left(\hat{f}(x_i) - f(x_i)\right)^2\right]$ möglichst klein sein.
Dabei ist es leicht, Kurven als Schätzung anzunehmen, die jeweils eines dieser Kriterien erfüllen. Einerseits hat beispielsweise eine Kurve, die jeden gegebenen Punkt durchläuft, einen Bias von 0, aber im Allgemeinen eine sehr hohe Varianz. Andererseits hat eine horizontale Kurve durch die Daten eine sehr geringe Varianz, dafür eine sehr große Verzerrung. Das Bias-Varianz-Dilemma ist in Abbildung 2.3 schematisch dargestellt.

[20] eigene Darstellung.
[21] vgl. Boor, C. de (1993), S. 2.

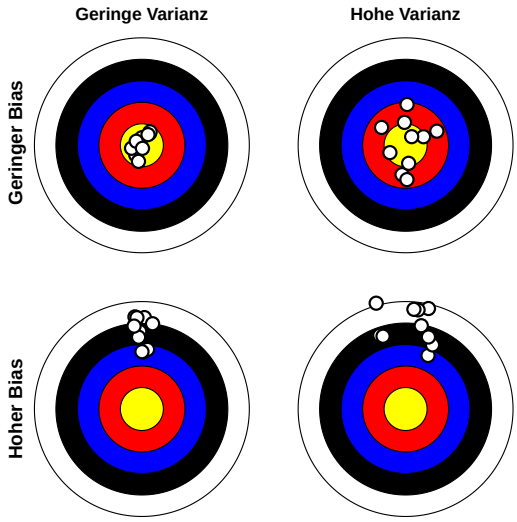
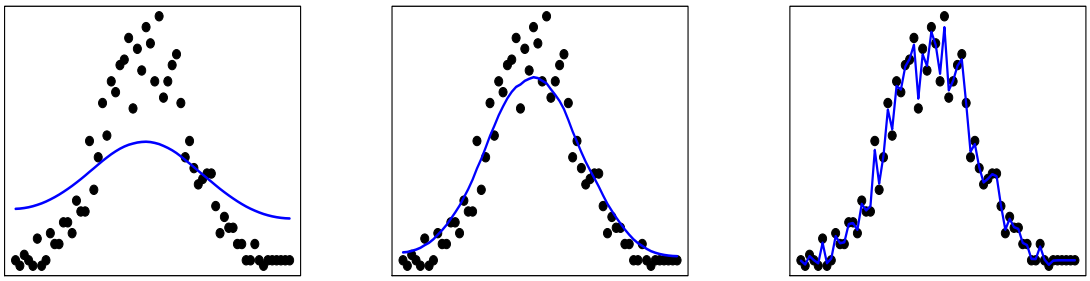


Abbildung 2.3: Bias-Varianz-Dilemma.[22]

Der Trade-off zwischen Bias und Varianz kann mit einem Strafterm (*Penalty*) gesteuert werden. Die folgende Abbildung soll eine Idee geben, welchen Einfluss der Penalty-Term auf die Schätzung haben kann.



(a) Underfitting. (b) Gute Schätzung. (c) Overfitting.

Abbildung 2.4: Dichteschätzung mit verschiedenen Penalty-Werten.[23]

Dargestellt ist dafür eine Stichprobe von 1.000 normalverteilten Zufallsvariablen, die in 50 Bins zusammengefasst und somit zensiert wurde. Die blaue Linie ist eine Schätzung der Dichte mittels Splines, wobei verschiedene Werte als Penalty eingesetzt wurden. Dabei ist in Abbildung 2.4(a) eine zu große Penalty verwendet worden (*underfitting*), wodurch die Daten nicht geeignet beschrieben werden sowie ein sehr großer Bias und eine kleine Varianz vorliegen. Im Plot (c) ist dagegen ein zu kleiner Strafwert eingesetzt worden (*overfitting*). Es handelt sich dann um eine komplexe Funktion mit hoher Varianz und geringem Bias, die fast jeden Punkt trifft. Die mittlere Grafik (b) stellt einen guten Kompromiss zwischen Bias und Varianz dar.

[22] eigene Darstellung in Anlehnung an Fortmann-Roe, S. (2012) [online].
[23] eigene Darstellungen.

Es gibt verschiedene Möglichkeiten, den Wert der Penalty optimal zu wählen, welche im Zusammenhang mit der Parameterschätzung im Composite Link Model beschrieben werden.

2.2 Nicht-lineare Regression

In diesem Kapitel werden die Theorie der linearen und insbesondere der nicht-linearen Regression erläutert. Zusätzlich wird die Methode beschrieben, die im späteren Verlauf der Arbeit zur Berechnung der Startwerte für die bayesianischen Schätzmethoden verwendet wird. Diese beruht auf dem Verfahren, das Turnidge, J. et al. (2006) zur Charakterisierung des ECOFF vorstellt.

Regressionsanalyse

In der Regressionsanalyse soll der Einfluss mehrerer erklärender Variablen $x_1, ..., x_r$, sogenannte *Kovariablen*, auf eine abhängige Variable y, die *Zielvariable*, modelliert bzw. geschätzt werden. Dieser Zusammenhang drückt sich in Form einer Funktion $y = f(x_1, ..., x_r)$ aus, wobei jedoch nicht angenommen wird, dass diese Beziehung exakt gilt. Vielmehr ist sie durch zufällige Störgrößen überlagert, also es gilt[24]

$$y = f(x_1, ..., x_r) + \epsilon.$$

Bei der linearen Regressionsanalyse nimmt man einen linearen Einfluss der Kovariablen an, wobei gegebenenfalls vorher geeignet transformiert werden muss. Das lineare Modell hat somit die Form

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_r x_r + \epsilon$$

mit unbekannten Parametern $\beta_0, \beta_1, \ldots, \beta_r$, die geschätzt werden müssen. Für jede Beobachtung soll also an die Werte y_i und den $(r+1)$-dimensionalen Vektor $\boldsymbol{x} = (1, x_{i1}, \ldots, x_{ir})^T,\ i = 1, ..., n$ ein Modell der Form

$$y_i = \beta_0 + x_{i1}\beta_1 + ... + x_{ir}\beta_r + \epsilon_i = \boldsymbol{x}^T\boldsymbol{\beta} + \epsilon \tag{2.3}$$

mit unbekanntem $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_r)$ angepasst werden. Mit den Vektoren $\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$ und $\boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$ sowie der *Designmatrix* $\boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1r} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{nr} \end{pmatrix}$ können die n

[24] vgl. Fahrmeir, L. et al. (2009), S. 59 ff.

Gleichungen aus Formel (2.3) kompakt in Matrixnotation geschrieben werden:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

Als Standardmethode zur Ermittlung des Parametervektors $\boldsymbol{\beta}$ wird die *Kleinste-Quadrate-Schätzung* verwendet, bei der die quadratische Abweichung

$$S = \sum_{i=1}^{n} \Big[y_i - f(x_{i1}, ..., x_{ir}) \Big]^2 \tag{2.4}$$

in $\boldsymbol{\beta}$ minimiert wird. Im Falle der linearen Regression unter der Annahme normalverteilter Fehler $\boldsymbol{\epsilon}$ ergibt sich[25]

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

Bei Daten, die keinem linearen Zusammenhang folgen, wird dementsprechend keine lineare Funktion f bei der Minimierung in Formel (2.4) verwendet. Im vorliegenden Fall werden die MIC-Kategorien als Kovariablen und die kumulierten Beobachtungen in den einzelnen Kategorien als Zielvariable angenommen. Abbildung 2.5 zeigt die Daten, die augenscheinlich nicht durch eine lineare, sondern eher durch eine S-förmige Funktion dargestellt werden können.

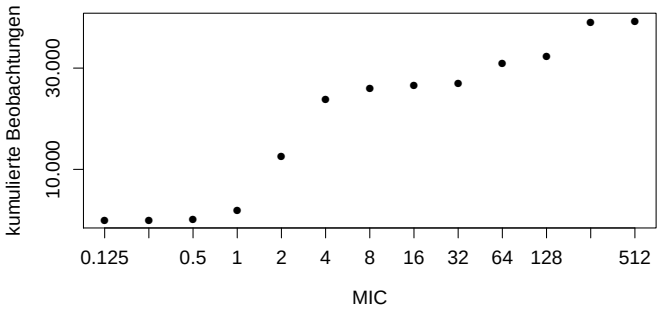


Abbildung 2.5: Kumulierte Beobachtungen der EUCAST-Daten.[26]

Verfahren von Turnidge et al. (2006)

Turnidge, J. et al. (2006)[27] stellt eine Möglichkeit zur Schätzung des Wild-Type-Anteils von MIC-Verteilungen mittels nicht-linearer Regression vor. Es wird davon ausgegangen, dass der Wild-Type-Anteil lognormalverteilt ist. Entsprechend kann auf der $\log_2$-Skala eine Normalverteilung angenommen werden, um den Zusammenhang zwischen

[25] vgl. Fahrmeir, L. et al. (2009), S. 92.
[26] eigene Darstellung.
[27] vgl. Turnidge, J. et al. (2006).

den kumulierten Beobachtungen und der MIC-Kategorie zu modellieren. Da neben dem Wild-Type-Teil immer auch noch der Non-Wild-Type-Anteil vorliegt, werden Regressionen sukzessive mit immer größer werdenden Teilmengen der Daten vorgenommen. So wird verhindert, dass im Vorfeld Annahmen über die Lage der Grenze zwischen den beiden Teilen getroffen werden müssen. Das Vorgehen wird in Abbildung 2.6 veranschaulicht, wobei dort die Beobachtungen der Einfachheit halber nicht kumuliert sind.

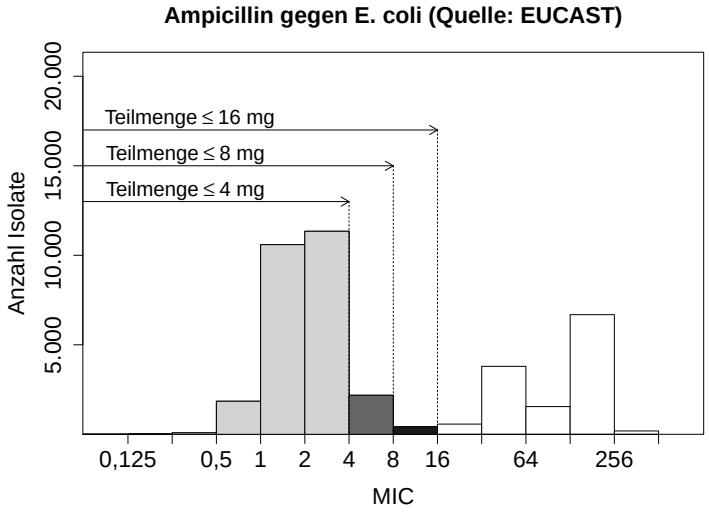


Abbildung 2.6: Sukzessives Verfahren nach Turnidge, J. et al. (2006).[28]

Gestartet wird in dem Verfahren mit derjenigen Teilmenge, die alle MIC-Kategorien beinhaltet, die entweder auf der Höhe des ersten Modalwertes oder eine Kategorie weiter liegen (hellgraue Balken). In den nächsten Schritten wird je eine weitere MIC-Kategorie eingefügt, wie die dunkler werdenden grauen Balken zeigen. So wird die Datenmenge Schritt für Schritt vergrößert.
Für die Daten werden dann jeweils drei Parameter geschätzt: der Mittelwert μ und die Standardabweichung σ der kumulativen Normalverteilung sowie die Gesamtsumme N der Beobachtungen. Der Wert N wird hier geschätzt statt als Konstante angenommen, damit nicht festgelegt ist, dass N nur Wild-Type-Isolate enthält. Das Regressionsmodell hat also die Form

$$y_i = N \cdot \Phi\left(\frac{x_i - \mu}{\sigma}\right) + \epsilon_i,$$

wobei y_i die beobachtete kumulierte Anzahl der Isolate bis zur logarithmierten MIC-Kategorie x_i und Φ die Verteilungsfunktion der Normalverteilung ist. Die in den Para-

[28] eigene Darstellung in Anlehnung an Turnidge, J. et al. (2006), S. 420.

metern N, μ und σ zu minimierende Funktion lautet somit

$$h(N, \mu, \sigma) = \sum_i \left[y_i - N \cdot \Phi \left(\frac{x_i - \mu}{\sigma} \right) \right]^2 .$$

Nun wird mit der ersten Teilmenge der Daten die Regression durchgeführt und anschließend sukzessive je eine MIC-Kategorie ergänzt, bis entweder eine optimale Schätzung gefunden oder alle Kategorien betrachtet wurden. Welche der Teilmengen nun die *beste* Schätzung liefert, kann mithilfe verschiedener Gütekriterien bestimmt werden. Eine Möglichkeit ist die Betrachtung des *Akaike Informationskriterium* (AIC), das als $AIC = 2k - 2l$ definiert ist. Die Anzahl der zu schätzenden Parameter wird als k bezeichnet und l ist die Loglikelihood des geschätzten Modells.[29] Das AIC bewertet die Güte der Anpassung eines Modells relativ zu dessen Komplexität.[30] Beim Vergleich zweier Regressionsmodelle wird das mit dem kleineren AIC als das Bessere angesehen, wobei die absoluten Werte des AIC keine Bedeutung haben, sondern nur relativ zum Vergleich verschiedener Modelle verwendet werden.
Eine zweite Möglichkeit, die Turnidge, J. et al. (2006) verwendet, ist der Vergleich zwischen wahrem n und geschätztem Wert N. In der Teilmenge, bei der die Differenz zwischen diesen beiden Werten am kleinsten ist, sind die geschätzten Parameter des Wild-Type-Anteils den wahren Beobachtungen am nächsten.
Mithilfe der Parameterschätzung kann anschließend der ECOFF bestimmt werden. Dieser ist derjenige MIC-Wert, bei dem die Wild-Type-Verteilung endet und welcher mithilfe des 99,9 %-Quantils ermittelt wird. Dazu wird das Quantil der geschätzten Verteilung berechnet und zur nächsten MIC-Kategorie hin gerundet.

2.3 Composite Link Model und indirekte Beobachtungen

Das Ziel bei der Anwendung des *Composite Link Model* (CLM) ist, die zugrundeliegende Verteilung gruppierter Daten zu schätzen, wie sie beispielsweise bei einem Histogramm vorliegen. Wenn die zur Erzeugung eines Histogrammes verwendeten Rohdaten bekannt sind, kann aus diesen *direkten Beobachtungen* die Verteilung hergeleitet werden. In vielen Fällen stehen diese Daten, wie im vorliegenden Fall, jedoch nicht zur Verfügung, wenn zum Beispiel Daten wegen zu geringer Auflösung eines Messgeräts nur in Intervallen gemessen werden können. Dann handelt es sich um *indirekte Beobachtungen*, da die beobachteten Werte als lineare Kompositionen wie Teilsummen oder gewichtete Summen vorliegen. Eine elegante Möglichkeit, aus indirekten Beobachtungen Verteilungen zu schätzen, ist das CLM, welches in Thompson, R., Baker,

[29] vgl. Snipes, M., Taylor, D. C. (2014), S. 4.
[30] vgl. Burnham, K., Anderson, D. (2002), S. 62.

R. J. (1981) als eine Erweiterung von verallgemeinerten linearen Modellen vorgestellt wird.[31]
In diesem Kapitel erfolgt zunächst eine allgemeine Beschreibung sowohl des verallgemeinerten linearen Modells als auch des Composite Link Models. Anschließend werden die Methodik sowie Lösungsmöglichkeiten des CLM erläutert.

Verallgemeinerte lineare Modelle

Verallgemeinerte lineare Modelle (basierend auf dem englischsprachigen Begriff Generalized Linear Model, im Folgenden als GLM abgekürzt) sind eine von McCullagh P., Nelder J. A. (1989)[32] eingeführte Verallgemeinerung der klassischen linearen Regression. Bei dieser werden unabhängige normalverteilte Fehler $\boldsymbol{\epsilon}$ angenommen, wohingegen beim GLM die Fehlerterme einer Verteilung aus der Klasse der exponentiellen Familie folgen können. Ein GLM besteht nach McCullagh P., Nelder J. A. (1989) aus den folgenden drei Komponenten:[33]

1. Zufallskomponente: $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ sei eine Realisierung einer Zufallsvariablen $Y = (Y_1, \ldots, Y_n)$, deren Komponenten unabhängig entsprechend der Exponentialfamilie mit der Form

$$f_Y(y, \nu, \rho) = \exp\left(\frac{y\nu - b(\nu)}{a(\rho)} + c(y, \nu)\right)$$

   verteilt sind.

2. Systematische Komponente: Sei $\boldsymbol{x_i} = (x_{0i}, x_{1i}, \ldots, x_{ri})$ der $(r+1)$-dimensionale Vektor der unabhängigen Variablen mit $x_{0i} = 1$ und $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_r)$ der unbekannte Parametervektor. Dann heißt

$$\eta_i = \sum_{j=0}^{r} x_{ji}\beta_j = \boldsymbol{x_i}^T\boldsymbol{\beta}$$

   der *lineare Prädiktor*.

3. Link-Funktion: Sei $\mathrm{E}(Y_i) = \mu_i$. Dann wird eine Funktion $g : \mathbb{R} \to \mathbb{R}$ mit der Eigenschaft

$$\nu_i = g(\mu_i)$$

   Link-Funktion genannt. Oft werden dabei Modelle mit kanonischen Link-Funktionen

[31] vgl. Thompson, R., Baker, R. J. (1981).
[32] vgl. McCullagh P., Nelder J. A. (1989).
[33] vgl. ebenda, S. 27.

verwendet. Darunter versteht man solche Links, bei denen

$$\nu_i = \eta_i = g(\mu_i)$$

gilt. Die Link-Funktion bildet in diesem Fall also den natürlichen Parameter ν der Exponentialfamilie ab.

Allgemeines Konzept vom Composite Link Model

Daten, die nur gruppiert vorliegen, machen eine detaillierte Analyse schwierig bis unmöglich. In vielen Fällen ist es deshalb notwendig, die Gruppierung wieder aufzuheben, um die Verteilung der ursprünglichen Daten modellieren zu können. Dabei handelt es sich um sogenannte *inverse Probleme*, die schlecht konditioniert sind, da sie zu viele Freiheitsgrade haben, um exakt gelöst werden zu können.
Ein Beispiel: Es liegt ein Histogramm mit zehn Klassen und somit zehn beobachteten Werten vor. Die Verteilung soll aber nun auf einem viel feineren Raster geschätzt werden, zum Beispiel soll jede Klasse in fünf kleinere Klassen aufgegliedert werden. Es gibt jedoch keine eindeutige, sondern sehr viele Möglichkeiten, die Werte des groben Rasters auf das feine aufzuteilen.
Wenn es nicht möglich ist, eine eindeutige Lösung zu finden, müssen Beschränkungen, Annahmen und Vorkenntnisse über die zugrundeliegenden Daten bei der Schätzung eingebracht werden. So bietet es sich beispielsweise an, eine gewisse Glattheit der Verteilung zu fordern, da Kurven mit einem glatten Verlauf einen geringen Detailgrad aufweisen und es nicht möglich ist, aus wenigen gruppierten Daten feine Details herauszulesen. Diese Annahme kann durch Penalisierung umgesetzt werden.

Methodik des Composite Link Models

Es sei $x_1, \ldots, x_I$ eine Folge von Werten (hier beispielsweise I MIC-Werte) und ψ_i $(i = 1, \ldots, I)$ die zugehörigen erwarteten Anzahlen, welche die Verteilung der x_i bilden. Bei einer Stichprobengröße von n sei p_i die Wahrscheinlichkeit des Wertes x_i und damit $\psi_i = np_i$. Lägen die Daten nicht gruppiert vor, so folgte die Anzahl der Beobachtungen der x_i Poisson-Verteilungen mit Erwartungswerten ψ_i.
Es soll jedoch die Verteilung $\psi = (\psi_1, \ldots, \psi_I)$ geschätzt werden, wobei die tatsächlich beobachteten Anzahlen $m = (m_1, \ldots, m_J)$ Realisationen von unabhängigen Poisson-Variablen $\mathcal{D}_j, j = 1, \ldots, J$ mit Erwartungswerten $\mu_j = \mathrm{E}(\mathcal{D}_j)$ seien, die sich durch Gruppierung der originalen Verteilung in $J < I$ Klassen ergibt. Dieser Zusammenhang ist in Abbildung 2.7 dargestellt.

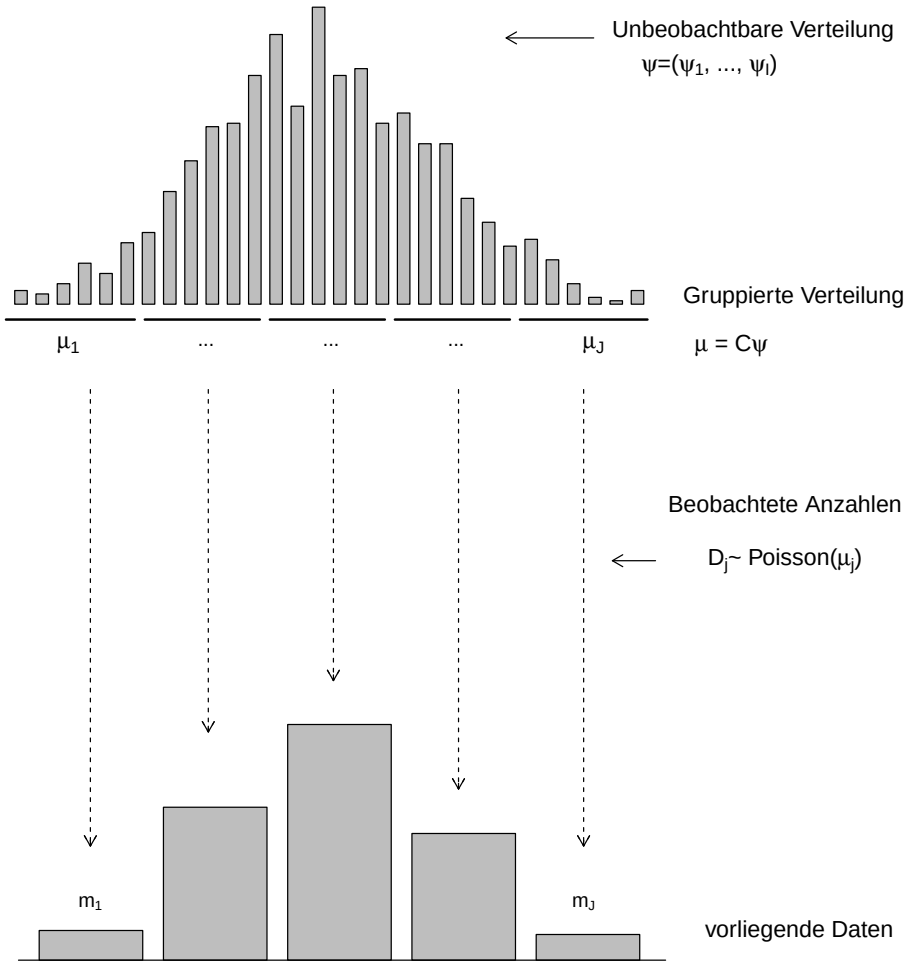


Abbildung 2.7: Methode des Composite Link Model.[34]

Jedes μ_j ergibt sich aus der Summe aller ψ_i, welche in der j-ten Klasse des Histogrammes liegen, wobei die beobachteten Anzahlen die Wahrscheinlichkeit

$$\mathbb{P}(\mathcal{D}_j = m_j) = \frac{\mu_j^{m_j}}{m_j!} \mathrm{e}^{-\mu_j}$$

haben.
Die Werte für μ ergeben sich also aus $\mu_j = \sum_{i=1}^{I} c_{ji}\psi_i$ bzw. in Matrixschreibweise

$$\boldsymbol{\mu} = C\boldsymbol{\psi}$$

mit der Kompositionsmatrix C

[34] eigene Darstellung in Anlehnung an Rizzi, S. et al. (2015). S. 139.

$$C = \begin{pmatrix} 1 & \dots & 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & 0 & 0 & \vdots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & 1 & \dots & 1 \end{pmatrix}.$$

Dabei entspricht die Anzahl der Zeilen der Anzahl der Histogrammklassen J und die Anzahl der Spalten ist die Größe I der unbeobachtbaren Verteilung ψ. Es gilt $c_{ji} = 1$ für die zu Klasse j aggregierten Werte.
Um nicht-negative Werte von ψ zu garantieren, wird die kanonische Link-Funktion mit $\psi = \exp(B\phi)$ verwendet und der Koeffizientenvektor $\phi = (\phi_1, \dots, \phi_I)$ geschätzt. Das CLM erweitert nun das GLM insofern, als dass die kanonische Link-Funktion zusätzlich mit der Kompositionsmatrix C multipliziert wird und so ein Datenpunkt für I/J Klassen steht.
Ist die Zahl I der Elemente in der zugrundeliegenden Verteilung klein, kann als Designmatrix B die Einheitsmatrix verwendet werden, um so I kategoriale Variablen zu erhalten. Ist jedoch der Wert I groß, so sollte eine B-Spline-Basis mit Dimension $k \times I$ gewählt werden, da in diesem Fall nur $k < I$ Parameter geschätzt werden müssen.[35]

Parameterschätzung im Composite Link Model

Die I Elemente der zugrundeliegenden Verteilung ψ sollen aus den $J < I$ beobachteten Anzahlen geschätzt werden. Da es sich dabei um ein inverses Problem handelt, müssen zusätzliche Annahmen getroffen werden. Es wird deshalb angenommen, dass die Verteilung glatt ist, also dass sich benachbarte Elemente von ψ nicht drastisch unterscheiden. Diese Annahme der Glattheit wird mithilfe eines Strafterms bezüglich der Koeffizienten ϕ implementiert, da Glattheit von ϕ Glattheit von $\psi = \exp(B\phi)$ induziert. Der Strafterm wird im Folgenden auch *Roughness-Penalty* genannt. Die Rauheit (roughness) wird durch die benachbarten Elemente von ϕ gesteuert. Dafür bezeichne $\Delta\phi_i$ die Differenz $\phi_i - \phi_{i-1}$ und analog Differenzen höherer Ordnung $\Delta^2\phi_i = \Delta(\Delta\phi_i) = \phi_i - 2\phi_{i-1} + \phi_{i-2}$. Setze D_k als Matrix mit Elementen $(d_{ij;r})$, welche die Differenzen der Ordnung r berechnet, also $D_r\phi = \Delta^r\phi$. Beispielhaft D_1 und D_2 mit $n = 5$:[36]

$$D_1 = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}, \qquad D_2 = \begin{pmatrix} 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \end{pmatrix}.$$

[35] vgl. Rizzi, S. et al. (2015), S. 139 ff.
[36] vgl. Eilers, P. H. C. (2007), S. 246.

Dabei gilt $\sum_{i=1}^{n}(\Delta^r\phi_i)^2 = \sum_{i=1}^{n}(D_{i;r}\phi_i)^2 = ||D_r\phi||^2 = (D_r\phi)^T(D_r\phi) = \phi^T \underbrace{D_r^T D_r}_{=:P} \phi = \phi^T P\phi$ mit Penalty $P = D_r^T D_r$.

Die Likelihood

$$L := L(\mu_j; m) = \prod_{j=1}^{J} \frac{\mu_j^{m_j}}{m_j!} \mathrm{e}^{-\mu_j}$$

führt zur Loglikelihood

$$l := l(\mu_j; m) = \sum_{j=1}^{J} \left(m_j \ln(\mu_j) - \mu_j\right).$$

Die penalisierte Loglikelihood wird nun als

$$l^* = l - \frac{\lambda}{2}P = \sum_{j=1}^{J} \left(m_j \ln(\mu_j) - \mu_j\right) - \frac{\lambda}{2}P$$

definiert. Diese Funktion kann mithilfe einer modifizierten Version des *iteratively reweighted least-squares Algorithmus* maximiert werden, wobei der optimale Wert für den Penalty-Parameter λ mithilfe des AIC bestimmt wird. Eine alternative Lösung des CLM kann mit Bayes-Methoden vorgenommen werden, die im weiteren Verlauf der vorliegenden Arbeit angewendet werden.

3 Bayes-Schätzung für eine MIC-Verteilung

Lambert, P., Eilers, P. H. C. (2009)[37] stellt eine Möglichkeit vor, mit einem bayesianischen Ansatz die Verteilung aus gruppierten Daten zu schätzen.[38] Dazu wird eine Kombination aus dem Composite Link Model und einer Roughness-Penalty verwendet, um eine glatte Verteilung zu erhalten. Dieser Ansatz wird im Folgenden erläutert. Anders als beim CLM werden hier jedoch nicht die Anzahlen, sondern direkt die Dichte geschätzt, weshalb sich die Bezeichnungen zwischen dem nachfolgenden Kapitel und dem Grundlagenkapitel unterscheiden.
Der besseren Lesbarkeit halber werden im Folgenden Zufallsvariable und Realisierung in der Notation nicht unterschieden, wenn eindeutig ist, was gemeint ist.

3.1 Bayesian Composite Link Model

Es sei X eine stetige Zufallsvariable auf einem feinen Raster (a_0, a_J), deren zugehörige Dichtefunktion f_X anhand einer Diskretisierung π auf einem groben Raster geschätzt werden soll. Das feine Raster besteht aus einer hohen Anzahl (≥ 100) von Zwischenpunkten, welche das Intervall (a_0, a_J) in I Intervalle $l_i = (\chi_{i-1}, \chi_i)$ mit Mittelpunkten $u_i, i = 1, ..., I$ und gleicher Intervallbreite Δ teilen. Die zu schätzende Größe für jedes Intervall ist

$$\pi_i = \int_{l_i} f_X(t)\ \mathrm{d}t \approx f_X(u_i)\Delta\,,$$

wobei $\pi_i \in [0, 1)$ und $\sum_{i=1}^{I} \pi_i = 1$ gelten.

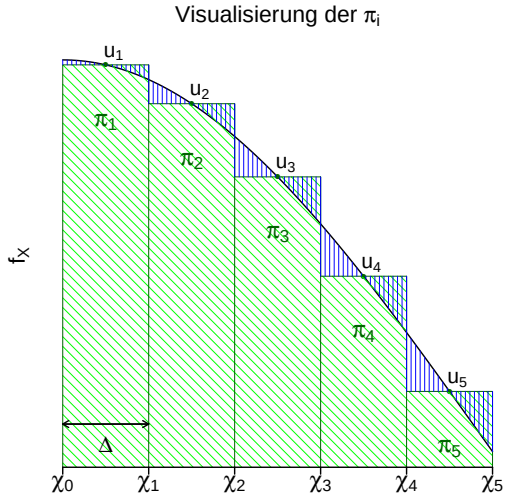


Abbildung 3.1: Visualisierung der zu schätzenden Größe π.[39]

[37] vgl. Lambert, P., Eilers, P. H. C. (2009).
[38] vgl. Jaspers, S. et al. (2016).
[39] eigene Darstellung.

Außerdem sei $m = (m_1, ..., m_J) \in \mathbb{R}^J$ die Anzahl der Beobachtungen der Zufallsvariable $\mathcal{D} = (\mathcal{D}_1, ..., \mathcal{D}_J)$ in jeder der Klassen $\mathcal{J}_j = (a_{j-1}, a_j)$, welche das Intervall (a_0, a_J) unterteilen. Es wird der Einfachheit halber angenommen, dass die Klassengrenzen aus der Menge $\{\chi_0, \ldots, \chi_I\}$ bestehen, sodass $J < I$ und $I/J \in \mathbb{N}$ gelten. Die Wahrscheinlichkeit der Beobachtungen sei multinomialverteilt und es folgt mit $n = \sum_{j=1}^{J} m_j$ die zugehörige Dichte

$$\begin{aligned}\mathbb{P}(\mathcal{D}_1 = m_1, \ldots, \mathcal{D}_J = m_J|\pi) &= \frac{n!}{\prod_{j=1}^{J}(m_j!)} \underbrace{\left(\sum_{i=1}^{I/J} \pi_i\right)^{m_1}}_{\kappa_1} \cdots \underbrace{\left(\sum_{i=I-I/J+1}^{I} \pi_i\right)^{m_J}}_{\kappa_J} \\ &\propto \prod_{j=1}^{J} \kappa_j^{m_j}.\end{aligned}$$

In der folgenden Tabelle ist die Notation der gegebenen Daten zusammengefasst.

Beobachtete MIC	Intervall	Anzahl	Anzahl (kumuliert)
x_1	$[a_0, a_1]$	m_1	y_1
x_2	$[a_1, a_2]$	m_2	y_2
$\ldots$	$\ldots$	$\ldots$	$\ldots$
x_J	$[a_{J-1}, a_J]$	m_J	y_J

Tabelle 3.1: Struktur der Ergebnisse eines Dilution-Experiments.

Der Zusammenhang zwischen den Klassen $\mathcal{J}_j$ und dem feinen Raster I wird in der Kompositionsmatrix $C \in \mathbb{R}^{J \times I}$ kodiert. Es gilt $C = (c_{ji})$ mit $c_{ji} = 1$, falls $l_i \subset \mathcal{J}_j$.

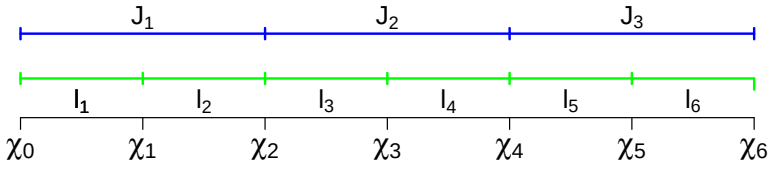


Abbildung 3.2: Visualisierung zur Kompositionsmatrix C.[40]

Die zu Abbildung 3.2 gehörende Kompositionsmatrix C ist

$$C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Die Wahrscheinlichkeit, dass eine Beobachtung in der j-ten Klasse liegt, ist

[40] eigene Darstellung.

$\kappa_j = \sum_{i=1}^{I} c_{ji}\pi_i$ bzw.

$$\boldsymbol{\kappa} = C\boldsymbol{\pi}$$

in Matrixschreibweise. Die Schätzung der π_i ist ein schlecht-konditioniertes Problem, weshalb als zusätzliche Restriktion gefordert wird, dass $\boldsymbol{\pi}$ glatt ist.
Dazu sei B eine B-Spline-Basis auf äquidistanten Knoten auf dem Raster (a_1, a_J) mit $(B)_{ik} = b_{ik} = b_k(u_i)$, welche die k Funktionswerte der Basis-Funktionen auf den Zwischenpunkten u_i $(i = 1, \ldots, I)$ enthält. Die Diskretisierung π der Verteilung wird durch

$$\pi_i = \pi_i(\phi) = \frac{\mathrm{e}^{\eta_i}}{\mathrm{e}^{\eta_1} + \cdots + \mathrm{e}^{\eta_I}}$$

mit $\eta = B\phi$ und Spline-Koeffizienten $\phi = (\phi_1, \ldots, \phi_k)$ modelliert. Auch hier gilt die Identifizierbarkeitsbedingung $\sum_{i=1}^{k} \phi_i = 1$ (siehe Formel (2.2)). Es ergibt sich also, dass das Modell für π ein GLM und das für $\boldsymbol{\kappa}$ ein CLM ist.
Die Roughness-Penalty ergibt sich wie in Kapitel 2.3 erläutert aus der Differenz der Ordnung r der Spline-Koeffizienten $\Delta^r\phi$, wobei in der vorliegenden Arbeit $r = 2$ gewählt wurde. Als a-priori-Verteilung der Spline-Koeffizienten-Differenz folgt

$$\Delta^r\phi \sim \mathcal{N}(0, \lambda^{-1}). \tag{3.1}$$

Daraus resultiert eine Annahme zur a-priori-Verteilung der Spline-Koeffizienten mit zugehöriger Dichtefunktion

$$\begin{aligned}
\mathbb{P}(\Delta^r\phi|\lambda) &= \frac{1}{\sqrt{(2\pi)^{k-r}\det(\mathrm{diag}(\lambda^{-1}))}} \exp\left(-\frac{1}{2}(\Delta^r\phi)^T \mathrm{diag}(\lambda^{-1})^{-1}\Delta^r\phi\right) \\
&= \frac{1}{\sqrt{(2\pi)^{k-r}\lambda^{-k+r}}} \exp\left(-\frac{1}{2}(\Delta^r\phi)^T \mathrm{diag}(\lambda^{-1})^{-1}\Delta^r\phi\right) \\
&\propto \lambda^{\mathcal{R}(P)/2} \exp\left(-\frac{\lambda}{2}\phi^T P\phi\right)
\end{aligned}$$

mit $\mathcal{R}(P) = k - r$. Damit ergibt sich

$$\mathbb{P}(\phi|\lambda) \propto \lambda^{\mathcal{R}(P)/2} \exp\left(-\frac{\lambda}{2}\phi^T P\phi\right).$$

Jaspers, S. et al. (2016) wählt als a-priori-Verteilung von λ eine Gammaverteilung $\mathcal{G}(a, b)$ mit $\mathbb{P}(\lambda) = \frac{b^a}{\Gamma(a)}\lambda^{a-1}\exp(-b\lambda)$, da es sich dabei um die *konjugierte Verteilung* zu (3.1) handelt.

Eine konjugierte Verteilung ist formal wie folgt definiert:[41]
Sei $\mathcal{F}$ die Klasse der Verteilungen $\mathbb{P}(y|\theta)$ und $\mathcal{G}$ die der a-priori-Verteilungen von θ, dann ist die Klasse $\mathcal{G}$ *konjugiert* zu $\mathcal{F}$, falls

$$\mathbb{P}(\theta|y) \in \mathcal{G} \text{ für alle } \mathbb{P}(\cdot|\theta) \in \mathcal{F} \text{ und } \mathbb{P}(\cdot) \in \mathcal{G}.$$

Für die *natürliche* konjugierte Verteilung wählt man $\mathcal{G}$ als Klasse der Verteilungen, die aus derselben Familie wie die Likelihood stammt. Das bedeutet, dass $\mathbb{P}(\theta)$ die natürliche konjugierte Verteilung von $\mathbb{P}(y|\theta)$ ist, wenn $\mathbb{P}(\theta|y)$ und $\mathbb{P}(\theta)$ dieselbe Form haben.
Der Vorteil konjugierter a-priori-Verteilungen ist, dass die a-posteriori-Verteilung[42]

$$\mathbb{P}(\theta|x) = \frac{\mathbb{P}(x|\theta)\,\mathbb{P}(\theta)}{\int \mathbb{P}(x|\theta')\,\mathbb{P}(\theta')\,d\theta'},$$

welche sich aus Likelihood $\theta \mapsto \mathbb{P}(x|\theta)$ und a-priori-Verteilung $\mathbb{P}(\theta)$ zusammensetzt, eine geschlossene Form hat.[43] Ansonsten muss zur Berechnung eine numerische Integration angewendet werden.

Mithilfe des Bayes-Theorems ergibt sich die gemeinsame a-posteriori-Verteilung von (ϕ, λ):

$$\begin{aligned}
\mathbb{P}(\phi,\lambda|\mathcal{D}) &= \frac{\mathbb{P}(\mathcal{D}|\phi,\lambda)\mathbb{P}(\phi,\lambda)}{\mathbb{P}(\mathcal{D})} \propto \mathbb{P}(\mathcal{D}|\phi,\lambda)\mathbb{P}(\phi,\lambda) = \mathbb{P}(\mathcal{D}|\phi,\lambda)\mathbb{P}(\phi|\lambda)\mathbb{P}(\lambda) \\
&= \left(\prod_{j=1}^{J} \kappa_j^{m_j}\right) \lambda^{\mathcal{R}(P)/2} \exp\left(-\frac{\lambda}{2}\phi^T P \phi\right) \frac{b^a}{\Gamma(a)} \lambda^{a-1} \exp(-b\lambda) \\
&\propto \left(\prod_{j=1}^{J} \kappa_j^{m_j}\right) \lambda^{a+0.5\mathcal{R}(P)-1} \exp\left(-\lambda(b + 0.5\phi^T P \phi)\right).
\end{aligned}$$

Die bedingten a-posteriori-Verteilungen für λ und ϕ sind damit

$$\begin{aligned}
\mathbb{P}(\lambda|\phi,\mathcal{D}) &= \frac{\mathbb{P}(\lambda,\phi,\mathcal{D})}{\mathbb{P}(\phi,\mathcal{D})} \propto \mathbb{P}(\lambda,\phi,\mathcal{D}) = \mathbb{P}(\phi,\lambda|\mathcal{D})\mathbb{P}(\mathcal{D}) \propto \mathbb{P}(\phi,\lambda|\mathcal{D}) \\
&\propto \left(\prod_{j=1}^{J} \kappa_j^{m_j}\right) \lambda^{a+0.5\mathcal{R}(P)-1} \exp\left(-\lambda(b + 0.5\phi^T P \phi)\right) \qquad (3.2)
\end{aligned}$$

[41] vgl. Gelman, A. et al. (2014), S. 35 f.
[42] vgl. ebenda, S. 6.
[43] vgl. ebenda, S. 322.

und

$$\begin{aligned}\mathbb{P}(\phi|\lambda,\mathcal{D}) &= \frac{\mathbb{P}(\lambda,\phi,\mathcal{D})}{\mathbb{P}(\lambda,\mathcal{D})} \propto \mathbb{P}(\lambda,\phi,\mathcal{D}) = \mathbb{P}(\phi,\lambda|\mathcal{D})\mathbb{P}(\mathcal{D}) \propto \mathbb{P}(\phi,\lambda|\mathcal{D}) \\ &\propto \left(\prod_{j=1}^{J}\kappa_j^{m_j}\right)\lambda^{a+0.5\mathcal{R}(P)-1}\exp\left(-\lambda b\right)\exp\left(-\frac{\lambda}{2}\phi^T P\phi\right) \\ &\propto \left(\prod_{j=1}^{J}\kappa_j^{m_j}\right)\exp\left(-\frac{\lambda}{2}\phi^T P\phi\right).\end{aligned}$$

Aus Formel (3.2) folgt $(\lambda|\phi,\mathcal{D}) \sim \mathcal{G}\left(a+0.5\mathcal{R}(P), b+0.5\phi^T P\phi\right)$.

3.2 Erweiterung des Ansatzes

Mithilfe des beschriebenen Ansatzes kann eine stetige Wahrscheinlichkeitsverteilung aus gruppierten Daten geschätzt werden. In der vorliegenden Arbeit soll die MIC-Verteilung derart geschätzt werden, dass Wild- und Non-Wild-Type-Anteil separat berechnet werden, sodass der Ansatz erweitert werden muss. Dabei wird nur die Verteilung des Non-Wild-Type-Anteils des Bakterium mit dem Bayesian Composite Link Model geschätzt, da der Wild-Type-Teil als unimodal-parametrisch verteilt angenommen werden kann. Das Modell muss deshalb wie folgt erweitert werden:

$$\pi_i = \mathbb{P}(X \in l_i) = \gamma \underbrace{\left(f_1(u_i;\theta_1)\Delta\right)}_{\pi_i^{(1)}} + (1-\gamma)\underbrace{\left(f_2(u_i)\Delta\right)}_{\pi_i^{(2)}}.$$

Die Wild-Type-Komponente $\pi_i^{(1)}$ wird dabei als lognormalverteilt mit Parameter $\theta_1 = (\mu_1, \sigma_1)$ angesehen. Nach der Logarithmierung der Daten wird der erste Anteil des Mischmodells mit der Verteilungsfunktion der Normalverteilung $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \mathrm{e}^{-\frac{1}{2}t^2}\mathrm{d}t$ wie folgt modelliert:

$$\pi_i^{(1)} = \Phi_{\mu_1,\sigma_1}(\chi_i) - \Phi_{\mu_1,\sigma_1}(\chi_{i-1}).$$

Dabei sind χ_i und χ_{i-1} wie oben beschrieben die Grenzen der Klassen l_i. Die Parameter μ_1, σ_1 bezeichnen Mittelwert und Varianz der Lognormalverteilung, wobei μ_1 als normalverteilt und σ_1 als invers gammaverteilt angenommen werden mit

$$\mu_1 \sim \mathcal{N}(\mu_{11}, \mu_{12}) \qquad \text{und} \qquad \sigma_1 \sim I\mathcal{G}(\sigma_{11}, \sigma_{12}).$$

Die Berechnung des Non-Wild-Type-Anteils wird wie oben beschrieben vorgenommen. Auch hier lautet das Modell für $\pi_i^{(2)}$

$$\pi_i^{(2)}(\phi) = \frac{\mathrm{e}^{\eta_i}}{\mathrm{e}^{\eta_1} + \cdots + \mathrm{e}^{\eta_I}}$$

mit $\eta = B\phi$ und $\sum_{i=1}^{k} \phi_i = 1$.
Im weiteren Verfahren müssen deshalb nicht nur die B-Spline-Koeffizienten ϕ und der Penalty-Parameter λ geschätzt werden, sondern zusätzlich auch das Gewicht γ der Mischung sowie die Parameter μ_1 und σ_1 der Lognormalverteilung. Die jeweiligen a-priori-Verteilungen sind auch hier entsprechend den konjugierten Verteilungen gewählt und in der nachstehenden Tabelle zusammengefasst.

Parameter	Verteilung	Dichtefunktion
μ_1	$\mathcal{N}(\mu_{11}, \mu_{12})$	$\mathbb{P}(\mu_1) = \frac{1}{\mu_{12}\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}\left(\frac{\mu-\mu_{11}}{\mu_{12}}\right)^2}$
σ_1	$I\mathcal{G}(\sigma_{11}, \sigma_{12})$	$\mathbb{P}(\sigma_1) = \frac{\sigma_{12}^{\sigma_{11}}}{\Gamma(\sigma_{11})} \sigma_1^{-\sigma_{11}-1} \mathrm{e}^{-\frac{\sigma_{12}}{\sigma_1}}$
γ	$Beta(\alpha, \beta)$	$\mathbb{P}(\gamma) = \frac{1}{B(\alpha,\beta)} \gamma^{\alpha-1}(1-\gamma)^{\beta-1}$

Tabelle 3.2: Zusammenfassung der gewählten a-priori-Verteilungen.

Als gemeinsame a-posteriori-Verteilung ergibt sich insgesamt:

$$\begin{aligned}
\mathbb{P}(\gamma, \mu_1, \sigma_1, \phi, \lambda|\mathcal{D}) &= \frac{\mathbb{P}(\gamma, \mu_1, \sigma_1, \phi, \lambda, \mathcal{D})}{\mathbb{P}(\mathcal{D})} \\
&= \mathbb{P}(\gamma, \mu_1, \sigma_1)\ \mathbb{P}(\phi, \lambda|\mathcal{D}) \\
&= \mathbb{P}(\gamma)\ \mathbb{P}(\mu_1)\ \mathbb{P}(\sigma_1)\ \mathbb{P}(\phi, \lambda|\mathcal{D}) \\
&\propto \left(\prod_{j=1}^{J} \kappa_j^{m_j}\right) \gamma^{\alpha-1}(1-\gamma)^{\beta-1} \exp\Big(-\frac{1}{2}\Big(\frac{\mu-\mu_{11}}{\mu_{12}}\Big)^2\Big) \sigma_1^{-\sigma_{11}-1} \\
&\quad \exp\left(-\frac{\sigma_{12}}{\sigma_1}\right) \lambda^{a+0.5\mathcal{R}(P)-1} \exp\Big(-\lambda(b + 0.5\phi^T P\phi)\Big) \qquad (3.3)
\end{aligned}$$

Zur Stichprobenerzeugung dieser Verteilung verwendet Jaspers, S. et al. (2016) Markov-Chain-Monte-Carlo-Methoden (MCMC-Methoden), wobei eine Metropolis-within-Gibbs-Strategie vorgeschlagen wird. Dabei werden Stichproben für jeden einzelnen Parameter aus allen bedingten Verteilungen, den sogenannten *Full-Conditionals*, gezogen. Diese erhält man aus Formel (3.3). Beispielsweise gilt

$$\begin{aligned}
\mathbb{P}(\lambda|\gamma, \mu_1, \sigma_1, \phi, \mathcal{D}) &= \frac{\mathbb{P}(\lambda, \gamma, \mu_1, \sigma_1, \phi, \mathcal{D})}{\mathbb{P}(\gamma, \mu_1, \sigma_1, \phi, \mathcal{D})} \ \propto\ \mathbb{P}(\lambda, \gamma, \mu_1, \sigma_1, \phi, m) \\
&= \mathbb{P}(\lambda, \gamma, \mu_1, \sigma_1, \phi|\mathcal{D})\ \mathbb{P}(\mathcal{D}) \ \propto\ \mathbb{P}(\lambda, \gamma, \mu_1, \sigma_1, \phi|\mathcal{D})
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{P}(\lambda, \gamma, \mu_1, \sigma_1, \phi|\mathcal{D})\ \mathbb{P}(\mathcal{D}) \ \propto\ \mathbb{P}(\lambda, \gamma, \mu_1, \sigma_1, \phi|\mathcal{D}) \\
&\propto \left(\prod_{j=1}^{J} \kappa_j^{m_j}\right) \gamma^{\alpha-1}(1-\gamma)^{\beta-1} \exp\Big(-\frac{1}{2}\Big(\frac{\mu-\mu_{11}}{\mu_{12}}\Big)^2\Big)\sigma_1^{-\sigma_{11}-1} \\
&\quad \exp\left(-\frac{\sigma_{12}}{\sigma_1}\right) \lambda^{a+0.5\mathcal{R}(P)-1} \exp\left(-\lambda(b+0.5\phi^T P\phi)\right) \\
&\propto \lambda^{a+0.5\mathcal{R}(P)-1} \exp(-\lambda b) \exp\left(-\frac{\lambda}{2}\phi^T P\phi\right). \qquad (3.4)
\end{aligned}
$$

Daraus folgt $(\lambda|\gamma, \mu_1, \sigma_1, \phi, \mathcal{D}) \sim \mathcal{G}(a+0.5\mathcal{R}(P); b+0.5\phi^T P\phi)$. Durch ähnliche Umformungen folgen bis auf Normierung

$$\mathbb{P}(\phi|\gamma, \mu_1, \sigma_1, \lambda, \mathcal{D}) \propto \prod_{j=1}^{J} \kappa_j^{m_j} \exp\left(-0.5\lambda\phi^T P\phi\right), \qquad (3.5)$$

$$\mathbb{P}(\gamma|\mu_1, \sigma_1, \lambda, \mathcal{D}) \propto \prod_{j=1}^{J} \kappa_j^{m_j} \gamma^{\alpha-1}(1-\gamma)^{\beta-1}, \qquad (3.6)$$

$$\mathbb{P}(\mu_1|\sigma_1, \gamma, \phi, \lambda, \mathcal{D}) \propto \prod_{j=1}^{J} \kappa_j^{m_j} \exp\left(-\frac{(\mu_1-\mu_{11})^2}{2\mu_{12}^2}\right) \qquad (3.7)$$

und

$$\mathbb{P}(\sigma_1|\mu_1, \gamma, \phi, \lambda, \mathcal{D}) \propto \prod_{j=1}^{J} \kappa_j^{m_j} \sigma_1^{-\sigma_{11}-1} \exp\left(-\frac{\sigma_{12}}{\sigma_1}\right). \qquad (3.8)$$

Im ersten Schritt werden Startwerte für alle Parameter beim Algorithmus zur Stichprobenerzeugung verwendet und sobald die ersten erzeugte Werte vorliegen, werden in den nächsten Schritten immer die aktuellsten Werte der Parameter benutzt. Für λ, μ_1 und σ_1 wird jeweils ein Gibbs-Step im Metropolis-Hastings-Algorithmus angewendet und für ϕ eine modifizierte Version des Langevin-Hastings-Algorithmus. Die Details dieser Algorithmen werden im nächsten Kapitel beschrieben.

4 Markov-Chain-Monte-Carlo-Methoden

MCMC-Methoden werden verwendet, um Stichproben aus Wahrscheinlichkeitsverteilungen zu ziehen. Dafür wird eine Markov-Kette erzeugt, deren stationäre Verteilung die gesuchte Verteilung ist. Als einfaches einführendes Beispiel bietet sich die Inversionsmethode an. Dabei wird ausgenutzt, dass jede Zufallsvariable in eine gleichverteilte Zufallsvariable und insbesondere daher auch jede gleichverteilte in eine beliebige Zufallsvariable transformiert werden kann.
Dazu sei U eine $\mathcal{U}[0,1]$-verteilte Zufallsvariable und F^{-1} die Quantilsfunktion $F^{-1}(q) := \inf\{x \in \mathbb{R} \mid F(x) \geq q\}$. Dann ist $X := F^{-1}(U)$ eine reelle Zufallsvariable mit Verteilungsfunktion F.
Es gilt somit $\mathbb{P}(X \leq x) = \mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x)$ und damit $X \sim F$.[44]
Diese Methode zur Erzeugung von Zufallsvariablen kann jedoch nur für Verteilungen angewendet werden, für welche die Quantilsfunktion effizient und ohne großen Aufwand berechnet werden kann.[45] In allen anderen Fällen können sogenannte *indirekte Methoden* benutzt werden, die im Folgenden beschrieben werden. Dazu werden zunächst kurz die Eigenschaften von Markov-Ketten sowie die Schritte der *Acceptance-Rejection-Methode* erläutert. Mithilfe dieser Grundlagen werden dann der *Metropolis-Hastings-Algorithmus* und dessen Modifizierung, der *Langevin-Hastings-Algorithmus*, beschrieben.

4.1 Markov-Ketten

Eine Markov-Kette $(X_n)_{n\geq 1}$ ist ein stochastischer Prozess in diskreter Zeit, bei dem ein zukünftiger Zustand nur vom aktuellen und nicht von den Zuständen der Vergangenheit abhängt. Der Prozess ist also *gedächtnislos* und nur der aktuelle Status hat Einfluss auf den nächsten. Das Ziel bei der Anwendung von Markov-Ketten ist, Wahrscheinlichkeiten anzugeben, mit denen zukünftige Ereignisse eintreten.

Eine solche Kette besteht dabei aus einer Zustandsmenge S, einem Wahrscheinlichkeitsvektor $(p_i)_{i\in S}$, einer Matrix $\mathcal{P} = (p_{ij})_{i,j\in S}$ sowie einer Folge von Zufallsvariablen $(X_n)_{n\geq 1}$. Dabei ist die Zustandsmenge $S = \{i_0, i_1, i_2, ...\}$ endlich oder abzählbar unendlich mit Zuständen i_k. Der Wahrscheinlichkeitsvektor $(p_i)_{i\in S}$ mit $p_i \geq 0$ für alle i und $\sum_i p_i = 1$ beinhaltet die Wahrscheinlichkeiten p_i, dass der Prozess bei Zustand i startet.
Die Matrix $\mathcal{P}$ enthält die Übergangswahrscheinlichkeiten p_{ij}. Wenn die Markov-Kette

[44] vgl. Casella, G., Robert, C. P. (2010), S. 44.
[45] vgl. Kolonko, M. (2008), S. 88.

aktuell in Zustand i ist, dann befindet sie sich mit Wahrscheinlichkeit p_{ij} im nächsten Schritt in Zustand j. Dabei gilt für die Zeilensummen $\sum_i p_{ij} = 1$, also ist $\mathcal{P}$ eine *stochastische* Matrix.
Sei $(X_n)_{n\geq 1}$ ein stochastischer Prozess mit Werten in S und Übergangsmatrix $\mathcal{P}$. Der Prozess heißt Markov-Kette, wenn gilt:

$$\mathbb{P}(X_{k+1} = j | X_k = i, X_{k-1} = i_{k-1}, \ldots, X_1 = i_1, X_0 = i_0) = \mathbb{P}(X_{k+1} = j | X_k = i) = p_{ij}.$$

Genau dann ist der aktuelle Zustand nur vom vorherigen Zustand abhängig.[46] Diese bedingte Verteilung heißt *Markov-Kern* K mit

$$X_{k+1} | X_0, X_1, \ldots, X_k \sim K(X_k, X_{k+1}).$$

Es gilt beispielsweise für eine einfache *Random-Walk*-Markov-Kette

$$X_{k+1} = X_k + \epsilon_k$$

mit Fehlern $\epsilon_k \sim \mathcal{N}(0, 1)$. Daraus folgt, dass der Markov-Kern $K(X_k, X_{k+1})$ der Verteilung $\mathcal{N}(X_k, 1)$ entspricht.
Eine Markov-Kette heißt *geschlossen* oder *invariant*, wenn für eine nicht-leere Teilmenge $C \subset S$ gilt: $p_{ij} = 0$ für $i \in C$ und $j \notin C$.[47] Ist die Kette also einmal in einem der Zustände aus C angekommen, dann werden nur noch Zustände dieser Teilmenge angenommen. Eine Markov-Kette heißt *irreduzibel*, wenn S die einzige geschlossene Teilmenge ist. Das bedeutet, dass jeder Zustand des Zustandsraumes angenommen und somit jeder Zustand x von jedem Zustand y aus erreicht werden kann. Eine Markov-Kette heißt *stationär*, wenn gilt: Hat die Markov-Kette zu einem Zeitpunkt die Verteilung q, dann auch zu jedem nachfolgenden Zeitpunkt, also[48]

$$X_k \sim q, \text{ dann auch } X_{k+1} \sim q.$$

Eine stationäre Markov-Kette ist außerdem *rekurrent*, was bedeutet, dass jeder Zustand fast sicher unendlich oft angenommen wird.[49] Liegt eine rekurrente Markov-Kette mit stationärer Verteilung vor, so ist diese die *Grenzverteilung*. Die Verteilung von X_t ist dann unabhängig vom Startwert X_0 immer q. Diese Eigenschaft wird *Ergodizität* genannt und ist für das Ziehen von Zufallsvariablen von großer Bedeutung: Erzeugt nämlich ein Markov-Kern eine ergodische Markov-Kette mit stationärer Verteilung q,

[46] vgl. Behrends, E. (2000), S. 4 ff.
[47] vgl. ebenda S. 23.
[48] vgl. Casella, G., Robert, C. P. (2010), S. 168 ff.
[49] vgl. ebenda, S. 169.

dann wird mithilfe der Methode zur Stichprobenerzeugung im Endeffekt von der Verteilung q eine Stichprobe gezogen.[50]

4.2 Acceptance-Rejection-Methoden

Die zu Beginn des Kapitels erläuterte Inversionsmethode ist wegen mangelnder analytischer Lösbarkeit nicht für alle Verteilungen anwendbar. In solchen Fällen werden oft indirekte Methoden angewendet, die auf dem Konzept von *Acceptance-Rejection* basieren. Dabei wird eine erzeugte Zufallszahl nur dann als Beobachtung angenommen, wenn sie gewisse Voraussetzungen erfüllt.

Für die Acceptance-Rejection-Methode muss die Form der zu simulierenden Dichtefunktion f bis auf eine multiplikative Konstante bekannt sein (*Target-Density*). Zum Erzeugen von Zufallsvariablen wird eine Dichte g verwendet, die einfach zu simulieren ist (*Proposal-Density*). Die Dichte g muss folgende Eigenschaften erfüllen:[51]

(i) Die Dichten f und g haben denselben Träger.

(ii) Es existiert eine Konstante c mit $\frac{f(x)}{g(x)} \leq c$ für alle x.

Dann kann $X \sim f$ simuliert werden, indem zunächst $Y \sim g$ und $U \sim \mathcal{U}[0,1]$ erzeugt werden. Wenn die Ungleichung

$$U \leq \frac{f(Y)}{cg(Y)} \tag{4.1}$$

erfüllt ist, dann wird $X = Y$ gesetzt. Ansonsten werden Y und U neu erzeugt und wiederum Ungleichung (4.1) geprüft. In Pseudocode sieht der Acceptance-Rejection-Algorithmus wie folgt aus:

Algorithmus 1: Acceptance-Rejection

```
1 Erzeuge Y ~ g und U ~ U[0,1]
2 if U ≤ f(Y)/(cg(Y)) then
3 |   Akzeptiere X = Y
4 else
5 |   Gehe zu Schritt 1.
6 end
```

Wenn $X = Y$ akzeptiert wird, entspricht die Verteilung der akzeptierten Zufallsvariable

[50] vgl. Casella, G., Robert, C. P. (2010), S. 169.
[51] vgl. ebenda, S. 51 f.

der Verteilung von X, wie nachfolgende Berechnung zeigt:

$$\begin{aligned}
\mathbb{P}\left(Y \leq x | U \leq \frac{f(Y)}{cg(Y)}\right) &= \frac{\mathbb{P}(Y \leq x, U \leq \frac{f(Y)}{cg(Y)})}{\mathbb{P}(U \leq \frac{f(Y)}{cg(Y)})} = \frac{\int_{-\infty}^{x} \int_{0}^{\frac{f(y)}{cg(y)}} g(y) \, \mathrm{d}u \, \mathrm{d}y}{\int_{-\infty}^{\infty} \int_{0}^{\frac{f(y)}{cg(y)}} g(y) \, \mathrm{d}u \, \mathrm{d}y} \\
&= \frac{\int_{-\infty}^{x} \frac{f(y)}{cg(y)} g(y) \, \mathrm{d}y}{\int_{-\infty}^{\infty} \frac{f(y)}{cg(y)} g(y) \, \mathrm{d}y} = \frac{\int_{-\infty}^{x} f(y) \, \mathrm{d}y}{\int_{-\infty}^{\infty} f(y) \, \mathrm{d}y} = \int_{-\infty}^{x} f(y) \, \mathrm{d}y \\
&= \mathbb{P}(X \leq x). \qquad (4.2)
\end{aligned}$$

4.3 Metropolis-Hastings-Algorithmus

Das Funktionsprinzip von MCMC-Methoden beruht darauf, dass für eine gegebene Target-Density f ein Markov-Kern K mit stationärer Verteilung f generiert wird, mithilfe dessen eine Markov-Kette $(X_n)_{n\geq 1}$ erzeugt werden kann (Ergodizität). Die Schwierigkeit liegt nur darin, ein passendes K zu finden, das mit beliebigem f verbunden werden kann. Dafür existieren aber Methoden, die solche Markov-Kerne bilden, welche theoretisch für alle Dichten f verwendet werden können.[52] Eine dieser Methoden ist der *Metropolis-Hastings-Algorithmus*, der eine Markov-Kette $(X_n)_{n\geq 1}$ durch die nachfolgenden Schritte erzeugt:[53]

Algorithmus 2: Metropolis-Hastings

Input: $X_t = x_t$

Output: Markov-Kette (X_t)

1 Generiere $Y_t \sim g(y, x_t)$

2 Setze $X_{t+1} = \begin{cases} Y_t & \text{mit Wahrscheinlichkeit } \alpha(x_t, Y_t) \\ x_t & \text{mit Wahrscheinlichkeit } 1 - \alpha(x_t, Y_t) \end{cases}$

wobei $\alpha(x, y) = \begin{cases} \min\{1, \frac{f(y)}{f(x)} \frac{g(x|y)}{g(y|x)}\} & f(x)g(y|x) > 0 \\ 1 & f(x)g(y|x) = 0 \end{cases}$

Bei der Anwendung des Algorithmus wird dafür $U_t \sim \mathcal{U}[0,1]$ erzeugt und $X_{t+1} = Y_t$ gesetzt, falls $U_t \leq \alpha(x_t, Y_t)$ gilt, ansonsten $X_{t+1} = X_t$. Für symmetrische Proposal-Densities gilt $g(y|x) = g(x|y)$, wodurch sich die Berechnung von α zu $\alpha(x, y) = \min\{1, \frac{f(y)}{f(x)}\}$ vereinfacht. Der beschriebene Algorithmus erfüllt die sogenannte *detai-*

[52] vgl. Casella, G., Robert, C. P. (2010), S. 170.

[53] vgl. Casella, G., Robert, C. P. (2010), S. 170 f. und Dellaportas, P., Roberts, G. O. (2003), S. 5.

led balance Bedingung

$$f(x)K(y|x) = f(y)K(x|y)$$

mit Markov-Kern K. Daraus kann man folgern, dass die Markov-Kette irreduzibel und f die Dichte der stationären Verteilung ist.[54]
Als einfaches Beispiel wird der Metropolis-Hastings-Algorithmus mit $f \sim \mathcal{N}(0,1)$ und $g(y|x) \sim \mathcal{N}(x,1)$ implementiert.

```
Metropolis_Hastings <- function(n, x0)
{
  x <- vector(mode = "numeric")
  x[1] <- x0
  for(i in 1:(n-1))
  {
    u <- runif(1)
    y <- rnorm(1, x[i], 1)
    if (u <= exp(-0.5* (y^2 - x[i]^2)))
      x[i+1] <- y
    else
      x[i+1] <- x[i]
  }
  return(x)
}
```

Quellcode 4.1: Beispiel zum Metropolis-Hastings-Algorithmus.[55]

Die Funktion `Metropolis_Hastings` benötigt als Eingabeparameter die gewünschte Stichprobengröße `n` sowie einen Startwert `x0`. Anschließend wird, wie beschrieben, zunächst eine gleichverteilte Zufallsvariable erzeugt und mit einer Zufallsvariable aus der Proposal verglichen. Das Ergebnis des Random Walks ist in Abbildung 4.1 dargestellt.

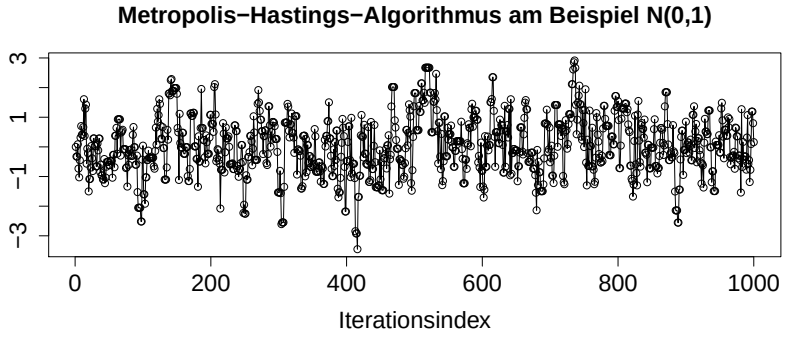


Abbildung 4.1: Random Walk des Metropolis-Hastings-Algorithmus.[56]

Bei dieser Art Zufallszahlen zu erzeugen, wird eine sogenannte *Burn-in-Phase* benötigt. Das bedeutet, dass die produzierten Zufallsvariablen erst nach einer gewissen Anzahl

[54] vgl. Casella, G., Robert, C. P. (2010), S. 172.
[55] eigene Darstellung.
[56] eigene Darstellung.

erzeugter Werte ihre stationäre Verteilung erreichen und deshalb die ersten Werte vernachlässigt werden müssen. Wie viele Werte das sind, hängt von verschiedenen Faktoren ab, zum Beispiel ist die Konvergenz für univariate Verteilungen mit endlichem Träger schneller als für andere Verteilungen.[57] Je nach Stichprobenumfang werden oft pauschal die ersten 1.000 erzeugten Werte nicht beachtet.
Ein Histogramm des obigen Beispiels mit der geschätzten Dichte zeigt folgende Abbildung. In grün ist die Dichte einer standardnormalverteilten Zufallsvariable dargestellt. Das Histogramm sowie die blaue Kurve gehören zu der mit dem Metropolis-Hastings-Algorithmus erzeugten Stichprobe.

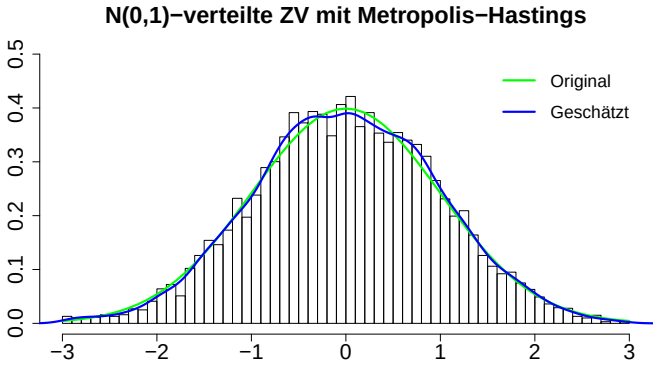


Abbildung 4.2: Verteilung erzeugt mit Metropolis-Hastings-Algorithmus.[58]

4.4 Metropolis-within-Gibbs-Strategie

Das Ziel der Metropolis-within-Gibbs-Strategie ist, eine Folge von Stichproben einer unbekannten gemeinsamen Wahrscheinlichkeitsverteilung mehrerer Zufallsvariablen zu erzeugen. Sei dazu $X = (X_1, \ldots, X_p)$ ein Zufallsvektor. Es wird angenommen, dass von den dazugehörigen bedingten Dichten $f_1, \ldots, f_p$ Stichproben generiert werden können, also dass

$$X_i|x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p \sim f_i(x_i|x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p) \text{ für } i = 1, 2, \ldots, p$$

simuliert werden kann. Das Problem, von der gemeinsamen Verteilung zu samplen, kann auf diese Art in kleinere Blöcke geteilt werden, die einzeln simuliert werden und immer von den anderen aktuell erzeugten Werten abhängen (Full-Conditionals). Das Ergebnis ist wiederum eine Markov-Kette, da Gibbs-Sampling ein Spezialfall des Metropolis-Hastings-Algorithmus ist.[59]

[57] vgl. Gentle, J. (1996), S. 140 f.
[58] eigene Darstellung.
[59] vgl. Casella, G., Robert, C. P. (2010), S. 206.

Algorithmus 3: Metropolis-within-Gibbs

Input: $X^{(t)} = (X_1^{(t)}, \ldots, X_p^{(t)}) = (x_1^{(t)}, \ldots, x_p^{(t)})$

Output: Markov-Kette (X_t)

1 Generiere $X_1^{(t+1)} \sim f_1(x_1 | x_2^{(t)}, \ldots, x_p^{(t)})$

2 Generiere $X_2^{(t+1)} \sim f_2(x_2 | x_1^{(t+1)}, x_3^{(t)}, \ldots, x_p^{(t)})$

$\vdots$

p Generiere $X_p^{(t+1)} \sim f_p(x_p | x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_{p-1}^{(t)})$

Der Metropolis-within-Gibbs-Algorithmus ist also eine Zusammensetzung von p Metropolis-Hastings-Algorithmen mit Acceptance-Wahrscheinlichkeit 1.[60] Die erzeugte Stichprobe $(X_1^{(t+1)}, X_2^{(t+1)}, \ldots, X_p^{(t+1)})$ des Zufallsvektors ist eine Realisierung der gemeinsamen Verteilung $f(x_1, x_2, \ldots, x_p)$.

4.5 Langevin-Hastings-Algorithmus

Der Metropolis-Hastings-Algorithmus wird mit einem beliebigen Startwert angewendet, der häufig ohne genauere Informationen über die Zielverteilung gewählt wird. Man verlässt sich so nur auf den Acceptance-Rejection-Schritt, welcher das Ergebnis des Algorithmus in die richtige Richtung leiten soll. Liegen jedoch mehr Informationen über die Zielverteilung vor, beispielsweise der *Gradient*, dann können effizientere Methoden zur Stichprobenerzeugung angewendet werden. Der Gradient ist der Vektor, welcher die ersten partiellen Ableitungen aller Komponenten enthält, also

$$\nabla f = \frac{\partial f}{\partial x_1} e_1 + \cdots + \frac{\partial f}{\partial x_d} e_d = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{pmatrix}$$

mit den Einheitsvektoren $e_1, \ldots, e_d$. Mithilfe der partiellen Ableitungen können Aussagen über die Änderungen des Funktionswertes von einem Punkt aus in Richtung der Koordinatenachsen gemacht werden. Dabei zeigt der Gradient immer in die Richtung des größten Zuwachses von f.[61]

Eine effizientere Methode zur Stichprobenerzeugung als der Metropolis-Hastings-Algorithmus ist der sogenannte *Langevin-Hastings-Algorithmus*. Bei diesem wird im Allgemeinen eine Proposal-Density der Form

[60] vgl. Gilks, W. R., S. 12.

[61] vgl. Papula, L. (2011), S. 61 f.

$$q(x,y) = \frac{1}{\sqrt{(2\pi\sigma)^d}} \exp\left(\frac{-||y - x - \frac{\sigma^2}{2}\nabla \log(f(x))||^2}{2\sigma^2}\right)$$

verwendet.[62] Damit wird angenommen, dass $X \sim \mathcal{N}(x + \frac{\sigma^2}{2}\nabla \log(f(x)), \sigma^2 I_d)$, also dass die Markov-Kette nicht entsprechend des vorherigen Zustandes zentriert ist, sondern der Information angepasst ist, an welcher Stelle die Target-Density eine höhere Wahrscheinlichkeit hat.
Jaspers, S. et al. (2016) verwendet zur Dichteschätzung eine modifizierte Version des *Metropolis-adjusted Langevin-Algorithmus*[63] (MALA). Der Basis-Algorithmus benutzt dabei als Proposal-Density eine k-variate Normalverteilung

$$\mathcal{N}\Big(\phi^{(t-1)} + 0.5\ \delta\nabla \log \mathbb{P}(\phi^{(t-1)}|\mathcal{D}), \delta I_k\Big)$$

mit Parameter $\delta > 0$ und der Kurzschreibweise $\mathbb{P}(\phi|\mathcal{D}) = \mathbb{P}(\phi|\lambda^{(t-1)}, \mathcal{D})$. Mithilfe von δ wird die Akzeptanzrate zum optimalen Wert 57,4 % hinbewegt, wobei Werte von 40 - 80 % ebenfalls akzeptabel sind.[64] Die Proposal wird mit Wahrscheinlichkeit

$$\alpha(\phi^{(t-1)}, \phi) = \min\left(1, \frac{\mathbb{P}(\phi|\mathcal{D})}{\mathbb{P}(\phi^{(t-1)}|\mathcal{D})}\frac{q(\phi, \phi^{(t-1)})}{q(\phi^{(t-1)}\phi)}\right)$$

angenommen, wobei jetzt

$$q(x,y) = \frac{1}{\sqrt{(2\pi\delta)^k}} \exp\left(-\frac{1}{2\delta}\Big|\Big|y - x - 0.5\delta\nabla \log \mathbb{P}(x|\mathcal{D})\Big|\Big|^2\right)$$

gilt. In der modifizierten Variante des Algorithmus wird die Proposal aus

$$\mathcal{N}\Big(\phi^{(t-1)} + 0.5\ \delta\Sigma\nabla \log \mathbb{P}(\phi^{(t-1)}|\mathcal{D}), \delta\Sigma\Big)$$

gezogen und so die Mischung der Markov-Kette verbessert. Die Matrix Σ spiegelt dabei die Abhängigkeitsstruktur der a-posteriori-Verteilung wider und wird im Folgenden gleich der bereits eingeführten Matrix $P = D_r^T D_r$ gesetzt. Mit diesen Anpassungen und

$$G := \nabla \log\ \mathbb{P}(\phi|\mathcal{D}) \qquad \text{sowie} \qquad G^{(t-1)} := \nabla \log\ \mathbb{P}(\phi^{(t-1)}|\mathcal{D}) \tag{4.3}$$

folgt

$$\frac{q(\phi, \phi^{(t-1)})}{q(\phi^{(t-1)}, \phi)} = \exp\Big(-\frac{1}{2}(G + G^{(t-1)})^T\Big((\phi - \phi^{(t-1)}) + \frac{\delta\Sigma}{4}(G - G^{(t-1)})\Big)\Big).$$

[62] vgl. Dellaportas, P., Roberts, G. O. (2003), S. 7 ff.
[63] vgl. Roberts, G. O., Tweedie, R. L. (1996), S. 345 f.
[64] vgl. Roberts, G. O., Rosenthal, J. S. (1998), S. 256.

Ein automatisches Tuning von δ in Richtung der optimalen Akzeptanzrate von 57,4 % kann mit dem Verfahren aus Atchadé, Y. F., Rosenthal, J. S. (2005)[65] erreicht werden. Dazu wird am Ende jedes Iterationsschritts der Wert δ auf $\delta^{(t+1)}$ gesetzt, wobei

$$\sqrt{\delta^{(t+1)}} = h\Big(\sqrt{\delta^{(t)}} + \xi_t\big(\alpha(\phi^{(t-1)}, \phi) - 0,574\big)\Big)$$

gelte mit

$$h(x) = \begin{cases} \epsilon & \text{, falls} \quad x < \epsilon \\ x & \text{, falls} \quad x \in (\epsilon, A) \\ A & \text{, falls} \quad x > A \end{cases}$$

sowie $\epsilon = 10^{-4}$ und $A = 10^4$. Außerdem muss ξ_t eine fallende Folge von positiven reellen Zahlen sein, sodass $|\xi_t - \xi_{t-1}| \leq t^{-1}$ gilt. Eine mögliche Wahl ist $\xi_t = t^{-1}$. Als Startwert für δ hat sich in der Praxis $\delta^{(1)} = 1,65^2/k^{1/3}$ etabliert.[66]

[65] vgl. Atchadé, Y. F., Rosenthal, J. S. (2005), S. 822 f.
[66] vgl. Roberts, G. O., Rosenthal, J. S. (1998), S. 258 ff.

5 Anwendung der Methoden zur Dichteschätzung

In diesem Kapitel werden die beschriebenen Methoden an den realen Daten von EUCAST angewendet, die zu Beginn nochmals genauer beschrieben werden. Bei der Dichteschätzung werden Wild-Type- und Non-Wild-Type-Anteil der Verteilung nacheinander bestimmt. Für den linken Teil der Verteilung werden zunächst Startwerte mit dem in Kapitel 2.2 beschriebenen Verfahren berechnet, mithilfe derer man anschließend Stichproben und damit Schätzer für die a-posteriori-Verteilungen erhält.
Der rechte Teil der Verteilung wird mit dem Bayesian Composite Link Model geschätzt. Bei der Verknüpfung beider Ergebnisse wird zusätzlich zu der von Jaspers, S. et al. (2016) vorgeschlagenen Modellierung von π eine alternative Definition vorgestellt. Alle Berechnungen wurden mit der Statistiksoftware `R` durchgeführt.[67] Die wichtigsten Implementierungsschritte sind hier vorgestellt, der komplette Quellcode ist im Anhang zu finden.

5.1 MIC-Verteilung von EUCAST

Die vorliegenden Daten umfassen 39.270 Datenpunkte im Bereich zwischen $^1/_8$ und 512 mg/l Antibiotikum, die in die beschriebenen 13 Kategorien eingeteilt sind. Der Wild-Type der Verteilung liegt augenscheinlich im Bereich bis 16 mg/l und der Non-Wild-Type, der Resistenzen gegen das Antibiotikum entwickelt hat, im Bereich von 16 mg/l bis 512 mg/l. Die gegebenen Datenpunkte stammen direkt von der EUCAST-Homepage und entsprechen dem Stand vom 29.03.2017. In der Implementierung werden die MIC-Kategorien als $\log_2$ betrachtet, sodass der Bereich das Intervall $[-3; 9]$ umfasst. Abbildung 1.1 zeigt nochmals die bereits in der Einleitung visualisierte Datengrundlage.

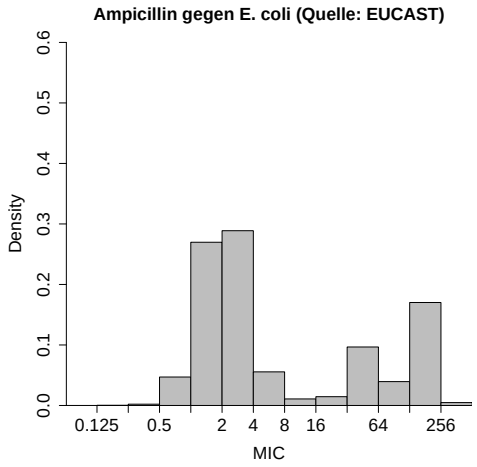


Abbildung 1.1: Ampicillin gegen E. coli.[68]

[67] vgl. R Core Team (2015).
[68] eigene Darstellung in Anlehnung an MIC-EUCAST (2017).

5.2 Wild-Type-Anteil

Das Verfahren von Turnidge, J. et al. (2006) wird verwendet, um Startwerte für den Bayes-Ansatz zu bestimmen. Es werden Werte für Mittelwert μ_1 und Standardabweichung σ_1 der Lognormalverteilung sowie das Mischungsverhältnis γ zwischen Wild-Type und Non-Wild-Type benötigt.
Der erste Modalwert der vorliegenden Daten liegt bei der Kategorie 2 mg/l Antibiotikum, deshalb wird dort mit der Regression gestartet. Die nachfolgende Abbildung zeigt die Ergebnisse der Regression für die MIC-Kategorien 4 bis 256, die in Tabelle 5.1 zusammengefasst sind.

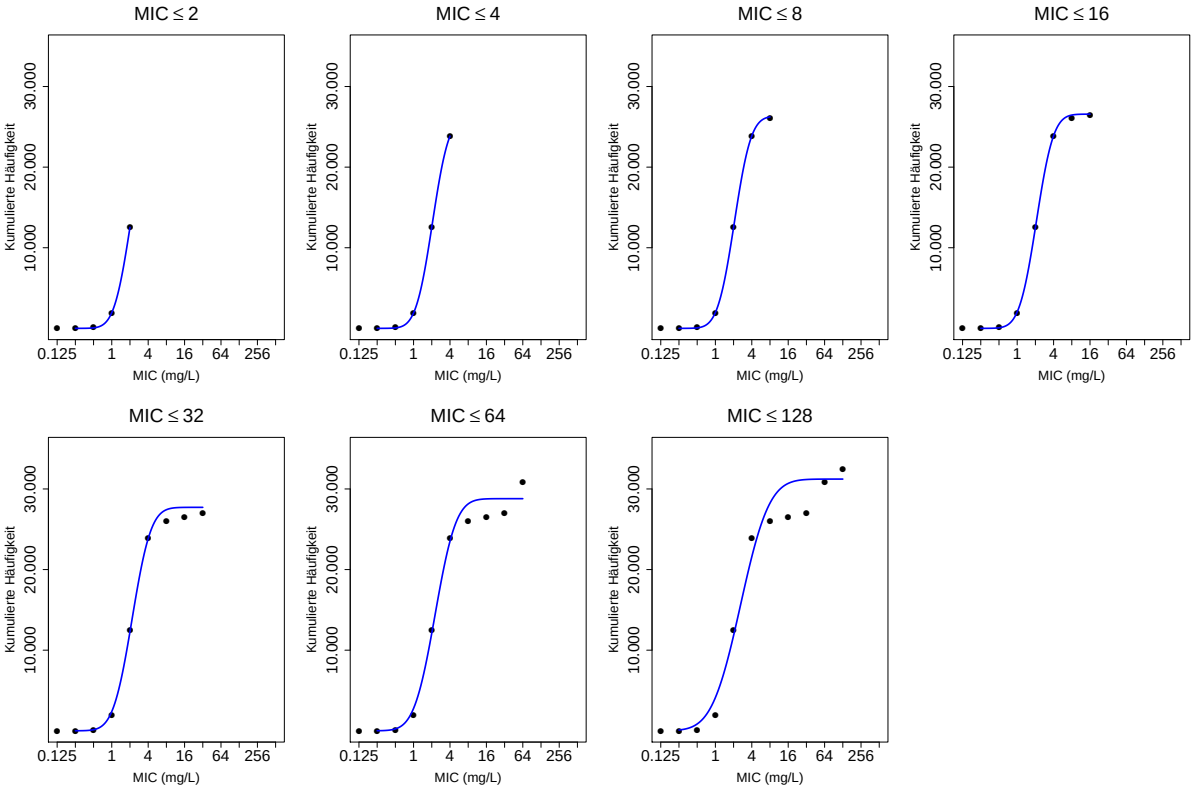


Abbildung 5.1: Sukzessive nicht-lineare Regressionsanalyse.[69]

Die Ergebnisse in der zweiten Zeile in Tabelle 5.1 weisen die geringste Differenz zwischen beobachteter und geschätzter Anzahl der Isolate auf und werden deshalb als Startwerte für den Bayes-Ansatz genutzt. Dabei werden die geschätzten Werte $\hat{\mu} = 1,04$, $\hat{\sigma} = 0,71$ und $\gamma = \frac{26.061}{\sum_{j=1}^{J} y_j} = 0,66$ so verwendet, dass sie jeweils der Mittelwert der entsprechenden a-priori-Verteilung sind mit Varianzen $\sqrt{0,05}$ für μ_1, 0,05 für σ_1 und 0,0005 für γ. Damit gilt $\mu_1 \sim \mathcal{N}(\mu_{11} = 1,04\ , \mu_{12} = 0,47)$, $\sigma_1 \sim IG(\sigma_{11} = 12,18\ , \sigma_{12} = 7,97)$ und $\gamma \sim Beta(\alpha = 295,61\ , \beta = 149,83)$.

[69] eigene Darstellung.

	Anzahl Beobachtungen			μ		σ	
Endpunkt	n	N	Diff.	$\hat{\mu}$	Std.Err.	$\hat{\sigma}$	Std.Err.
4	23.880	26.297	2.417	1,04	0,003	0,72	0,004
8	**26.061**	**26.157**	**96**	**1,04**	**0,003**	**0,71**	**0,004**
16	26.485	26.323	162	1,04	0,008	0,72	0,011
32	27.055	26.574	481	1,06	0,018	0,74	0,025
64	30.851	27.720	3.131	1,12	0,095	0,82	0,132
128	32.400	28.795	3.605	1,18	0,133	0,91	0,183
256	39.083	31.225	7.858	1,38	0,265	1,24	0,362

Tabelle 5.1: Ergebnisse der nicht-linearen Regressionen.

Die Werte für die Hyperparameter der a-priori-Verteilungen ergeben sich aus den folgenden Zusammenhängen:

$$\mathrm{E}(\sigma_1) = \frac{\sigma_{12}}{\sigma_{11} - 1} = 0,71 \ , \qquad \mathrm{Var}(\sigma_1) = \frac{\sigma_{12}^2}{(\sigma_{11} - 1)^2(\sigma_{11} - 2)} = 0,05$$

und

$$\mathrm{E}(\gamma) = \frac{\alpha}{\alpha + \beta} = 0,66 \ , \qquad \mathrm{Var}(\gamma) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} = 0,0005.$$

Anhand dieser Startwerte können nun Stichproben für die a-posteriori-Verteilungen (3.6), (3.7) und (3.8) und erzeugt werden. Dazu wurde der Metropolis-Hastings-Algorithmus mit den oben beschriebenen a-priori-Verteilungen und einer Iterationslänge von 20.000 verwendet und 20-mal wiederholt. Quellcode 5.1 zeigt die wichtigsten Schritte der Implementierung.

```
MIC_wild_type <- function(...)
{
  [...]
  for(i in 1:it.length)
  {
    # ------ Mu ------
    mu.hyper     <- mu_hyperparameters(mu1[i], mu_variance)
    proposalmu1 <- rnorm(1, mu.hyper[1], mu.hyper[2])
    u <- runif(1)

    if(f_mu1(mu1[i], mu11, mu12) == 0) accept <- T
    else accept <- u < f_mu1(proposalmu1, mu11, mu12)/f_mu1(mu1[i], mu11, mu12)

    if(accept){
      mu1[i+1] <- proposalmu1
    }
    else{
      mu1[i+1] <- mu1[i]
    }

    # ------ Sigma ------
    sigma.hyper     <- sigma_hyperparameters(sigma1[i], sigma_variance)
    proposalsigma1 <- rinvgamma(1, shape = sigma.hyper[1], scale = sigma.hyper[2])
    u <- runif(1)
```

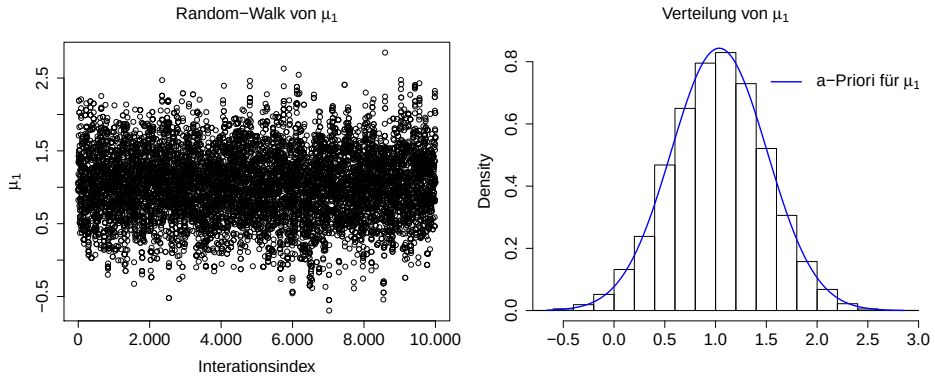
```
25
26      if(f_sigma1(sigma1[i], sigma11, sigma12) == 0) accept <- TRUE
27      else accept <- u < f_sigma1(proposalsigma1, sigma11, sigma12)/
28                          f_sigma1(sigma1[i], sigma11, sigma12)
29
30      if(accept){
31        sigma1[i+1] <- proposalsigma1
32      }
33      else{
34        sigma1[i+1] <- sigma1[i]
35      }
36
37      # ------ Gamma ------
38      gamma.hyper    <- gamma_hyperparameters(gamma[1], gamma_variance)
39      proposalgamma <- rbeta(1, gamma.hyper[1], gamma.hyper[2])
40      u <- runif(1)
41
42      if(f_gamma(gamma[i], alpha, beta) == 0) accept <- TRUE
43      else accept <- u < f_gamma(proposalgamma, alpha, beta)/
44                  f_gamma(gamma[i], alpha, beta)
45
46      if(accept){
47        gamma[i+1] <- proposalgamma
48      }
49      else{
50        gamma[i+1] <- gamma[i]
51      }
52    }
53    [...]
54  }
```

Quellcode 5.1: Metropolis-Hastings-Algorithmus.[70]

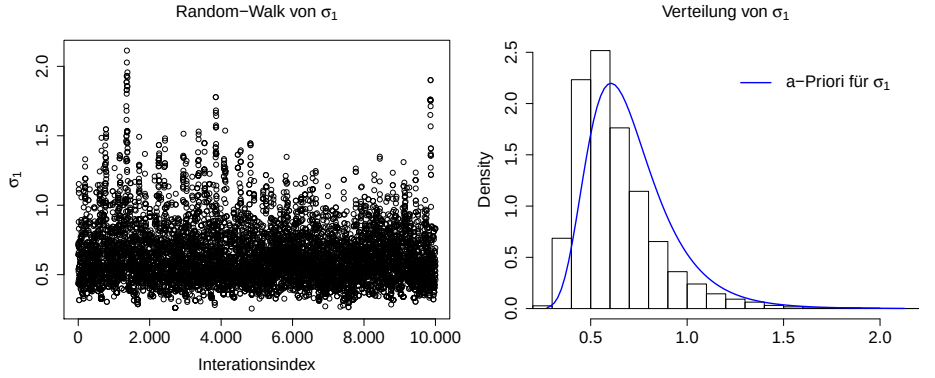
In den Zeilen 8-20 findet die Berechnung für μ_1 statt, zwischen den Zeilen 23 und 36 wird eine Stichprobe für σ_1 erzeugt und anschließend folgt ab Zeile 39 der Teil für γ. In jedem der drei Abschnitte werden zunächst die Hyperparameter der a-priori-Verteilung der einzelnen Parameter mit den Funktionen `mu1_hyperparameters`, `sigma1_hyperparameters` bzw. `gamma_hyperparameters` bestimmt. Als Proposal wird jeweils eine Zufallsvariable aus der gewählten a-priori-Verteilung mit den aktuellsten Hyperparametern verwendet. Nach diesen Vorbereitungen wird dann der Acceptance-Rejection-Schritt mit den entsprechenden Targets `f_mu1`, `f_sigma1` bzw. `f_gamma` durchgeführt. Die endgültigen Ergebnisse der Schätzung der einzelnen Parameter werden durch den Mittelwert der Stichproben bestimmt.

Die Ergebnisse des Random Walks sowie Histogramme der Parameter mit der zugehörigen a-priori-Verteilung sind in Abbildung 5.2 dargestellt. Aus diesen Stichproben ergeben sich die Bayes-Schätzer $\hat{\mu}_1 = 1,04$, $\hat{\sigma}_1 = 0,63$ und $\hat{\gamma} = 0,66$ durch Berechnung des Mittelwertes ohne Berücksichtigung der ersten 10.000 Werte (burn-in). Als ECOFF erhält man die MIC-Kategorie 8 mg/l Antibiotikum.

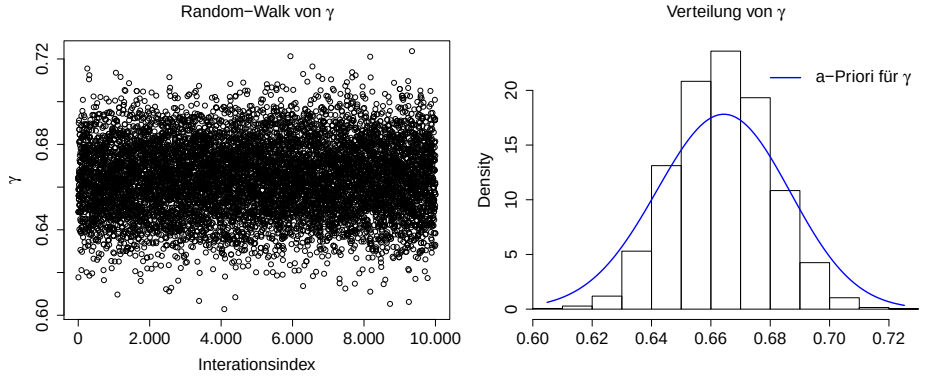
[70] eigene Darstellung.



(a) Ergebnis der Schätzung für μ_1



(b) Ergebnis der Schätzung für σ_1



(c) Ergebnis der Schätzung für γ

Abbildung 5.2: Ergebnisse des Bayesian Composite Link Model für den Wild-Type.[71]

5.3 Non-Wild-Type-Anteil

Zur Schätzung des Non-Wild-Type-Anteils der vorliegenden Daten wird eine modifizierte Version des Langevin-Hastings-Algorithmus verwendet. Die Datenpunkte liegen in 13 Klassen vor, also gilt $J = 13$. Das feine Raster, auf dem die Dichte geschätzt wird, sollte laut Jaspers, S. et al. (2016) über 100 Zwischenpunkte enthalten, sodass $\frac{I}{J} = 15$ und damit $I = 195$ gewählt wurde. Damit ergibt sich die Kompositionsmatrix

[71] eigene Darstellungen.

$C \in \mathbb{R}^{13\times 195}$, die mithilfe des folgenden Codes erzeugt wird.

```
CompositionMatrix <- function(J, I)
{
  C <- matrix(0, nrow = J, ncol = I)
  count <- 0
  for(j in 1:J)
  {
    C[j, (I/J * count + 1) : (I/J * (count + 1))] <- rep(1, I/J)
    count <- count + 1
  }
  return(C)
}
```

Quellcode 5.2: Berechnung der Kompositionsmatrix C.[72]

Die Berechnung der Intervallgrenzen χ_i sowie der Mittelpunkte u_i zeigt Quellcode 5.3. Der Vektor χ soll das Intervall $[-3, 9]$ in I Intervalle unterteilen und hat damit die Länge $I+1$. Der Vektor u_i der Mittelpunkte folgt aus Vektor χ und die Intervalllänge Δ ergibt sich aus $\Delta = \dfrac{x_J - x_1}{I}$.

```
set.chi <- function(x, I, Delta)
{
  chi <- vector(mode = "numeric")
  # divide interval (min(x), max(x)) in I intervals => vector has length I+1
  chi[1] <- min(x)
  chi[I+1] <- max(x)
  for(i in 2:I)
  {
    chi[i] <- chi[i-1] + Delta
  }
  return(chi)
}

set.u <- function(x, I, chi)
{
  # midpoints:
  u <- vector(mode= "numeric")
  for(i in 2:(I+1))
  {
    u[i-1] <- chi[i-1]+(chi[i]-chi[i-1])/2
  }
  return(u)
}
```

Quellcode 5.3: Hilfsfunktionen für die Vektoren χ und u.[73]

Anschließend kann der in Kapitel 4.5 beschriebene Langevin-Hastings-Algorithmus angewendet werden. Dazu wird zuerst die Berechnung des Gradienten vereinfacht.

[72] eigene Darstellung.
[73] eigene Darstellung.

Zunächst gilt

$$\begin{aligned}
\phi^T P \phi &= (\phi_1 \dots \phi_k) \begin{pmatrix} p_{11} & \dots & p_{1k} \\ \vdots & \ddots & \vdots \\ p_{k1} & \dots & p_{kk} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_k \end{pmatrix} \\
&= (\phi_1 \dots \phi_k) \begin{pmatrix} \phi_1 p_{11} + \dots + \phi_k p_{1k} \\ \vdots \\ \phi_1 p_{k1} + \dots + \phi_k p_{kk} \end{pmatrix} \\
&= \phi_1(\phi_1 p_{11} + \dots + \phi_k p_{1k}) + \dots + \phi_k(\phi_1 p_{k1} + \dots + \phi_k p_{kk}). \qquad (5.1)
\end{aligned}$$

Mit Formel (5.1) und G aus Formel (4.3) ergibt sich dann

$$\begin{aligned}
G &= \nabla \log\ \mathbb{P}(\phi|\mathcal{D}) = \nabla \log\ \mathbb{P}(\phi|\lambda^{(t-1)}, \mathcal{D}) = \nabla \log \left(c \cdot \exp\left(\frac{\lambda^{(t-1)}}{2} \phi^T P \phi \right) \right) \\
&= \nabla \left(\log(c) - \frac{\lambda^{(t-1)}}{2} \phi^T P \phi \right) = \frac{\lambda^{(t-1)}}{2} \nabla \phi^T P \phi \\
&= \frac{\lambda^{(t-1)}}{2} \nabla \Big(\phi_1 \big(\phi_1 p_{11} + \dots + \phi_k p_{1k} \big) + \dots + \phi_k \big(\phi_1 p_{k1} + \dots + \phi_k p_{kk} \big) \Big) \\
&= \frac{\lambda^{(t-1)}}{2} \begin{pmatrix} 2\phi_1 p_{11} + \sum_{i=2}^{k} \phi_i (p_{1i} + p_{i1}) \\ \vdots \\ 2\phi_k p_{kk} + \sum_{i=1}^{k-1} \phi_i (p_{ki} + p_{ik}) \end{pmatrix} \\
&= \frac{\lambda^{(t-1)}}{2} \begin{pmatrix} \sum_{i=1}^{k} \phi_i (p_{1i} + p_{i1}) \\ \vdots \\ \sum_{i=1}^{k} \phi_i (p_{ki} + p_{ik}) \end{pmatrix}
\end{aligned}$$

und damit die Ableitung an Stelle j:

$$\nabla_j = \frac{\lambda^{(t-1)}}{2} \sum_{i=1}^{k} \phi_i (p_{ji} + p_{ij}).$$

Für $G^{(t-1)}$ wird analog der vorherige Zustand des Koeffizientenvektors ϕ verwendet. Als Länge von ϕ wurde $k = J = 13$ gewählt. Quellcode 5.4 zeigt die Berechnung des Gradienten in `R`. Die Eingabewerte `chain` und `lambda` sind dabei die Werte der Markov-Kette bzw. λ im benötigten Zustand.

```
1 Gradient <- function(length = k, chain, lambda)
2 {
3   gradient <- vector(mode = "numeric")
4   for(i in 1:length)
5   {
6     tmp <- 0
7
```

```
    for(j in 1:length)
    {
      a <- chain[j]*(P[i, j] + P[j, i])
      tmp <- tmp + a
    }
    gradient[i] <- -tmp * lambda / 2
  }
  return(gradient)
}
```

Quellcode 5.4: Berechnung des Gradienten.[74]

Nun kann der Langevin-Hastings-Algorithmus angewendet werden. Dieser wurde ebenfalls 20-mal mit einer Iterationslänge von 20.000 und Burn-in-Phase von 10.000 durchgeführt. Die wichtigsten Schritte der Implementierung in R zeigt nachstehender Code.

```
MIC_non_wild_type_LH <- function(...)
{
  [...]
  data <- rep(normalize(m),rep(factorI, length(m)))  # percentage
  phi  <- ginv(basis) %*% log(data)
  phi  <- normalize(phi) # identifiability constraint

  chain <- matrix(data = NA, ncol = k, nrow = it.length + 1)
  chain[1, ] <- phi

  for(i in 1:it.length)
  {
    G.prev <- Gradient(k, chain[i,], lambda[i], P)
    proposal <- mvrnorm(1, chain[i,] + 0.5 * delta[i] * P %*% G.prev ,
                        delta[i] * P)
    G.cur <- Gradient(k, proposal, lambda[i], P)

    rand <- runif(1)

    alpha <- min(1, (exp(f_phi(lambda[i], proposal, P, basis, C)-
                           f_phi(lambda[i], chain[i,], P, basis, C))) *
                    exp(-1/2 * t(G.cur + G.prev) %*%
                          ((proposal - chain[i,]) +
                             delta[i]/4 * P %*% (G.cur - G.prev))))
    accept <- rand < alpha

    if(accept){
      chain[i+1,] <- proposal
    [...]

    }
    else{
      chain[i+1,] <- chain[i,]
      [...]

    }
    lambda[i+1] <- rgamma(1, shape = a + 0.5 * Rp,
                          rate = b + 0.5 * t(chain[i+1,]) %*% P %*% chain[i+1,])

```

[74] eigene Darstellung.

```
40      delta.next.root <- h(sqrt(delta[i]) + gamma.tuning(i)*(alpha - 0.574))
41      delta[i+1] <- delta.next.root^2
42    }
43  [...]
44  }
```

Quellcode 5.5: Metropolis-Hastings-Algorithmus.[75]

Zu Beginn müssen Startwerte für den Koeffizientenvektor ϕ erzeugt werden (Zeilen 4-6). Dazu werden die relativen Häufigkeiten der Daten in den Klassen jeweils I-mal wiederholt und anschließend wird ein Startvektor $\phi^{(1)}$mit

$$\eta = B\phi^{(1)} \Leftrightarrow \phi^{(1)} = B^{+}\eta$$

erzeugt, wobei als B^{+} die Moore-Penrose-Inverse verwendet wird, da die Matrix B nicht quadratisch und somit nicht invertierbar ist. Nach einigen weiteren Vorbereitungen wie der Initialisierung von λ und δ kann der Algorithmus wie beschrieben durchgeführt werden. Die Ergebnisse von `chain` sind die Spaltenmittelwerte der Markov-Kette. Nach wiederholter Durchführung des Langevin-Hastings-Algorithmus ergab sich eine durchschnittliche Akzeptanzrate von 57,3849 %, die nahe an der optimalen Akzeptanzrate von 57,4 % liegt, sowie eine Schätzung für die Dichte unter der Verwendung von B-Splines.

5.4 MIC-Density Gesamtergebnis

Im Folgenden werden Wild- und Non-Wild-Type der Verteilung gemeinsam betrachtet. Jaspers, S. et al. (2016) definiert π wie in Kapitel 3.2 beschrieben als

$$\pi_i = \mathbb{P}(X \in l_i) = \gamma \underbrace{\left(f_1(u_i;\theta_1)\Delta\right)}_{\pi_i^{(1)}} + (1-\gamma) \underbrace{\left(f_2(u_i)\Delta\right)}_{\pi_i^{(2)}}.$$

Das Ergebnis dieser Definition von π ist in Abbildung 5.3 dargestellt. Mit der in dieser Arbeit verwendeten Umsetzung der vorgestellten Methode liegt die Kurve im Bereich des Wild-Types sehr weit über den gegebenen Daten und im Bereich des Non-Wild-Types darunter. Der Dichtewerte im linken Bereich liegt deshalb so hoch, weil die beiden einzelnen Schätzungen jeweils ungleich null zwischen den MIC-Kategorien ½ und 8 mg/l Antibiotikum-Konzentration sind. Die Ergebnisse beider Schätzungen werden addiert, sodass sich zu hohe Werte ergeben. Im rechten Bereich wird die Non-Wild-Type Schätzung durch den Faktor $(1-\gamma)$ außerdem zu stark geglättet.

[75] eigene Darstellung.

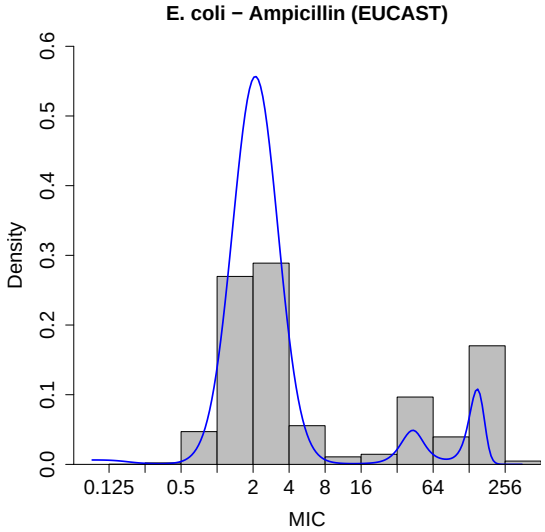


Abbildung 5.3: Dichteschätzung nach ursprünglicher Definition von π.[76]

Aus diesen Gründen wurde in der vorliegenden Arbeit eine alternative Definition von π verwendet: Bis zum berechneten ECOFF wird die für den Wild-Type geschätzte Dichte verwendet und ab dort die für den Non-Wild-Type, wodurch eine strikte Klassifizierung der beiden Teile stattfindet. Das Ergebnis zeigt folgende Abbildung.

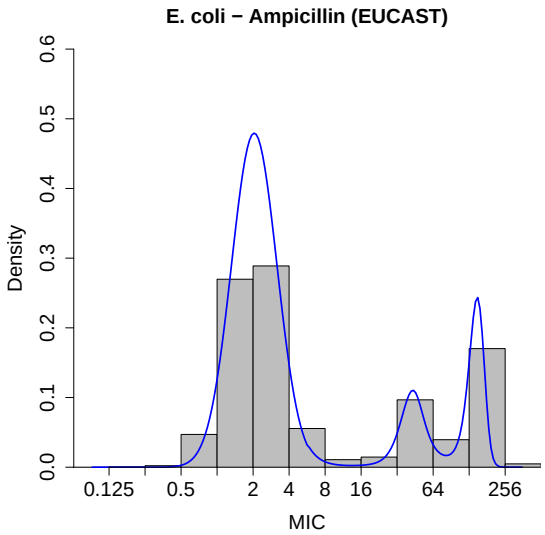


Abbildung 5.4: Dichteschätzung nach alternativer Definition von π.[77]

Bei der Simulationsstudie im weiteren Verlauf wird ebenfalls die alternative Definition von π verwendet.

[76] eigene Darstellung.
[77] eigene Darstellung.

6 Simulationsstudie

Zum Testen der vorgestellten Methode wird eine Simulationsstudie durchgeführt. Dazu wird eine Mischverteilung aus drei Lognormalverteilungen verwendet, die nach Turnidge, J. et al. (2006) die MIC-Verteilung gut beschreibt.[78] Anschließend wird das verwendete Verfahren anhand des mittleren quadratischen Fehlers beurteilt.

6.1 Dichteschätzung

Als Mischverteilung für die Simulationsstudie wurde

$$X \sim 0,6 \log\mathcal{N}(2\ ;0,8) + 0,2 \log\mathcal{N}(5,5\ ;0,7) + 0,2 \log\mathcal{N}(7,5\ ;0,6) \tag{6.1}$$

verwendet, wobei $\log\mathcal{N}$ eine Lognormalverteilung mit Parametern $\theta = (\mu_1, \sigma_1)$ bezeichne. Der Wild-Type-Anteil wird durch die erste Komponente repräsentiert und der Non-Wild-Type durch die Mischung zweier Lognormalverteilungen.
Die Berechnungen wurden analog zu denen der echten Daten durchgeführt. Das Ergebnis ist in der nachfolgenden Abbildung dargestellt, wobei die einzelnen Ergebnisse der Dichteschätzung in grau gezeichnet sind, das Gesamtergebnis der Schätzung in schwarz und die tatsächlich simulierte Dichte in grün.

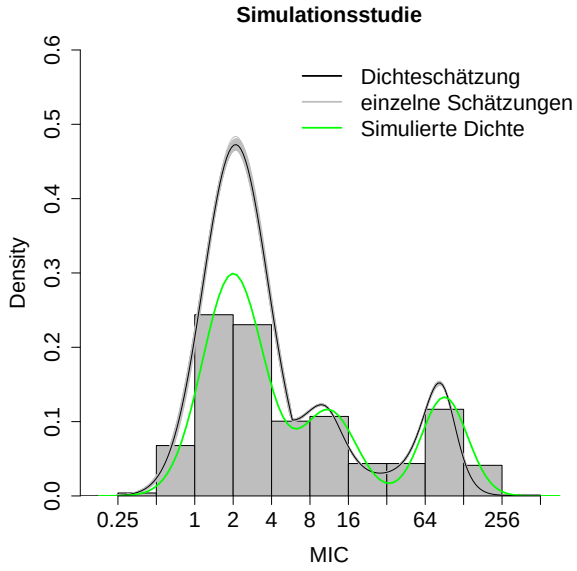


Abbildung 6.1: Ergebnis der Simulationsstudie.[79]

Zur Erzeugung der simulierten zensierten Daten wurde aus Formel (6.1) eine Stichprobe der Größe 5.000 gezogen und diese in Klassen zwischen $[-2, 9]$ der Länge 1 eingeteilt.

[78] vgl. Turnidge, J. et al. (2006), S. 420.
[79] eigene Darstellung.

Anschließend wurden Wild- und Non-Wild-Type mit den beschriebenen Verfahren mit einer Iterationslänge von 20.000, Burn-in-Phase von 10.000 und 100-maliger Wiederholung geschätzt.

6.2 Analyse des verwendeten Verfahrens

Die Güte des im Zuge dieser Arbeit implementierten Verfahrens kann anhand des *Mean Squared Error* (MSE) bewertet werden. Dabei wird für jede Monte-Carlo-Iteration m für jeden Zwischenpunkt u_i die quadratische Abweichung zwischen wahrer Dichte f und geschätzter Dichte $\hat{f}$ ermittelt und anschließend über alle Iterationen punktweise der Mittelwert berechnet. In Tabelle 6.2 ist das Vorgehen kurz skizziert.

Iteration	$i = 1$	$\dots$	$i = I$
$m = 1$	$\left(f(u_1) - \hat{f}^{(1)}(u_1)\right)^2$	$\dots$	$\left(f(u_I) - \hat{f}^{(1)}(u_I)\right)^2$
$\vdots$	$\vdots$		$\vdots$
$m = 100$	$\left(f(u_1) - \hat{f}^{(100)}(u_1)\right)^2$	$\dots$	$\left(f(u_I) - \hat{f}^{(100)}(u_I)\right)^2$
Monte-Carlo-MSE:	$\frac{1}{m}\sum_{j=1}^{m}\left(f(u_1) - \hat{f}^{(m)}(u_1)\right)^2$	$\dots$	$\frac{1}{m}\sum_{j=1}^{m}\left(f(u_I) - \hat{f}^{(m)}(u_I)\right)^2$

Tabelle 6.1: Berechnung des Monte-Carlo-MSE.

Der Monte-Carlo-MSE kann nun punktweise gezeichnet werden.

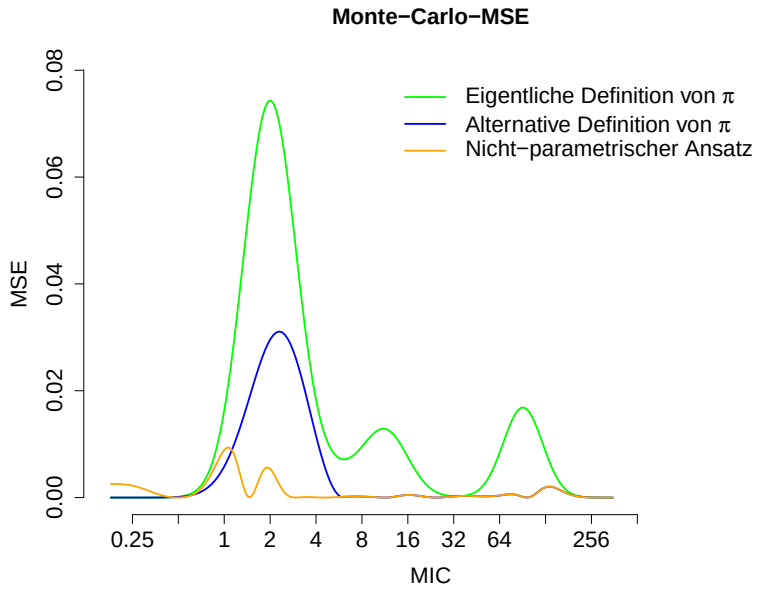


Abbildung 6.2: Monte-Carlo-MSE der verschiedenen Ergebnisse.[80]

[80] eigene Darstellung.

Die Grafik zeigt in grün den punktweisen Monte-Carlo-MSE für die eigentliche Definition von π, welche in Jaspers, S. et al. (2016) angewendet wird. In blauer Farbe ist der MSE für die in dieser Arbeit verwendete Definition von π dargestellt und die orangefarbene Kurve zeigt den mittleren quadratischen Fehler, wenn nur die nicht-parametrische Schätzmethode mit B-Splines verwendet wird. Es ist erkennbar, dass die alternative π-Definition überall einen kleineren MSE aufweist als die der Jaspers, S. et al. (2016) nachempfundene. Die zum nicht-parametrischen Ansatz gehörende Kurve verläuft jedoch hauptsächlich unter den anderen beiden Kurven und sollte somit vorzuziehen sein. Trotzdem ist es für die Dichteschätzung wahrer MIC-Daten wichtig, beide Teile der Verteilung separat zu schätzen, damit der ECOFF sowie das Mischungsverhältnis zwischen resistenten und nicht-resistenten Keimen bestimmt werden können.[81]

[81] vgl. Jaspers, S. et al. (2014b), S. 40.

7 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde das von Jaspers, S. et al. (2016) vorgestellte Paper „*A Bayesian approach to the semi-parametric estimation of a MIC distribution*“ thematisch behandelt. Es wurden zunächst mathematische Grundlagen erläutert und anschließend die verwendete Methodik beschrieben, nachvollzogen sowie angewendet. Dazu wurde die Dichte von realen MICs von EUCAST in zwei Schritten geschätzt: Erst wurden Startwerte für die Hyperparameter des parametrischen Anteil der Verteilung geschätzt und damit die Bayes-Schätzung mit dem Metropolis-Hastings-Algorithmus vorgenommen. Im zweiten Schritt wurde mit B-Splines der nicht-parametrische Bestandteil der Verteilung in einer modifizierten Version des Langevin-Hastings-Algorithmus ermittelt. Für die Gesamtbetrachtung der Dichteschätzung wurde eine alternative Möglichkeit zu der aus Jaspers, S. et al. (2016) vorgestellt.
Die entwickelte Methode wurde anschließend anhand einer Simulationsstudie getestet. Dazu wurde eine Mischverteilung aus drei Lognormalverteilungen gewählt und das Verfahren angewendet. Ausgewertet wurde es anschließend mithilfe des MSE, wobei die Methode mit der in Jaspers, S. et al. (2016) beschriebenen Definition der Dichte mit der in dieser Arbeit verwendeten alternativen Definition sowie der komplett nicht-parametrischen Schätzung verglichen wurde. Dabei wurde der Vorteil der semi-parametrischen Schätzung deutlich: Es kann der ECOFF als 99,9 %-Quantil der parametrischen Verteilung bestimmt werden, mithilfe dessen die Bakterienpopulation in Wild- und Non-Wild-Type eingeteilt werden kann.

Die Möglichkeiten zur Forschung und Weiterentwicklung der bisher verwendeten Verfahren sind vielfältig und im Hinblick auf die Relevanz für die menschliche Gesundheit wichtig. Ein wesentlicher Punkt ist, dass zwar die Wild-Type-Komponente der Verteilung über die Zeit stabil ist, beim Non-Wild-Type-Anteil jedoch sowohl das Verhältnis zwischen resistenten und nicht-resistenten Keimen als auch die Höhe der MIC über einen längeren Zeitraum schwanken können. Es wäre deshalb zu überlegen, ein zeitabhängiges Mischungsverhältnis $\gamma(t)$ im Modell einzuführen sowie die Koeffizienten der B-Splines ebenfalls in Abhängigkeit der Zeit darzustellen. So wäre es möglich, Trends in einem Zeitraum mehrerer Jahre zu erkennen und ein Überwachungstool zu entwickeln.

8 Anhang

Dieses Kapitel enthält den verwendeten Code mit einzelnen kurzen Beschreibungen. Der erste Teil beinhaltet Funktionen, die einerseits Berechnungen vereinfachen sowie auslagern und deren Resultate andererseits in weiteren Rechnungen benötigt werden. Weiter folgen die Implementierung der Dichteschätzung des Wild-Types mit Regressionsanalyse sowie Bayes-Schätzung und außerdem der Code für die Dichteschätzung des Non-Wild-Type. Jeweils abschließend werden alle Schritte für die Dichteschätzung der EUCAST-Daten gezeigt, die Schätzung der simulierten Daten funktioniert analog. Für die Berechnungen werden die `R`-Pakete `MASS`, unter anderem für die Funktionen `mvrnorm` (multivariate Normalverteilung) und `ginv` (Moore-Penrose-Inverse) sowie das Paket `MCMCpack` für die Funktion `rinvgamma` benötigt.

8.1 Allgemeine Funktionen, Hilfsfunktionen

Im nachfolgenden Code sind ausgelagerte Hilfsfunktionen zu finden, die zur Vorbereitung der Dichteschätzung dienen.

```
normalize <- function(x)
{
  # returns vector with sum = 1
  return(x/sum(x))
}

#-------------------------------------------------------------------------------

set.Delta <- function(x, I)
{
  return( (max(x)-min(x))/I)
}

#-------------------------------------------------------------------------------

set.chi <- function(x, I, Delta)
{
  chi <- vector(mode = "numeric")
  # divide interval (min(x), max(x)) in I intervals => vector has length I+1
  chi[1] <- min(x)
  chi[I+1] <- max(x)
  for(i in 2:I)
  {
    chi[i] <- chi[i-1] + Delta
  }

  return(chi)
}

#-------------------------------------------------------------------------------


```

```
set.u <- function(x, I, chi)
{
  # midpoints:
  u <- vector(mode= "numeric")
  for(i in 2:(I+1))
  {
    u[i-1] <- chi[i-1]+(chi[i]-chi[i-1])/2
  }
  return(u)
}
```

Quellcode 8.1: Hilfsfunktionen.[82]

Der nächste Codeteil enthält spezielle Funktionen wie die Berechnung der Kompositionsmatrix und B-Spline-Basis.

```
CompositionMatrix <- function(J, I)
{
  # C is JxI matrix
  C <- matrix(0, nrow = J, ncol = I)
  count <- 0   # counter for columns
  for(j in 1:J)
  {
    C[j, (I/J * count + 1) : (I/J * (count + 1))] <- rep(1, I/J)
    count <- count + 1
  }
  return(C)
}

# ------------------------------------------------------------------------

B <- function(x, i, q, t)
{
  delta <- t[2]-t[1]
  t <- c(t, max(t)+delta*1:(q+1))
  if(q == 0)
    (1) * (x >= t[i] & x < t[i+1]) + (0) * (x >= t[i+1] & x < t[i])
  else
    (x-t[i])/(t[i+q] - t[i])*B(x, i, q-1, t) +
      (t[i+q+1] - x)/(t[i+q+1]-t[i+1])*B(x, i+1, q-1, t)
}

# ------------------------------------------------------------------------

BSplineBasis <- function(k, I, u, x)
{
  tmp <- vector(mode = "numeric")
  for(i in 1:(k))
  {
    for(j in 1:(I))
    {
      tmp <- c(tmp, B(u[j], i, 3, x))
    }
  }
  basis <- matrix(tmp, nrow = I, byrow = FALSE )
  return(basis)
}
```

[82] eigene Darstellung.

```
42 # ------------------------------------------------------------------------------
43
44 Gradient <- function(length = k, chain, lambda, P)
45 {
46   gradient <- vector(mode = "numeric")
47   for(i in 1:length)
48   {
49     tmp <- 0
50     for(j in 1:length)
51     {
52       a <- chain[j]*(P[i, j] + P[j, i])
53       tmp <- tmp + a
54     }
55     gradient[i] <- -tmp * lambda / 2
56   }
57   return(gradient)
58 }
```

Quellcode 8.2: Spezielle Funktionen für die Dichteschätzung.[83]

Nun folgen Funktionen, die zur Dichteschätzung des Wild-Type benötigt werden. Das sind neben der Berechnung des ersten Modalwertes die a-posteriori-Verteilungen der Parameter μ_1, σ_1 und γ sowie die Funktionen `*_hyperparameters`, die anhand des jeweiligen Mittelwertes und der gewählten Varianzen die Hyperparameter zurückgibt. Die Funktionen `create_list_wildtype` und `print.wildtype` kommen in ähnlicher Form noch häufiger vor: Mit der ersten Funktion wird eine Liste erzeugt, welche alle relevanten Ergebnisse enthält und die zweite Funktion definiert, welche Teile dieser Liste als sichtbarer Output ausgegeben werden.

```
1  firstMode <- function(x, m)
2  {
3    for(i in 2:(length(m)-1))
4    {
5      # checks if current value is higher as previous and lower as next
6      if(m[i-1] <= m[i] && m[i] >= m[i+1])
7
8        return(x[i])
9    }
10 }
11
12 # ------------------------------------------------------------------------------
13
14 # target density of mu1
15 f_mu1 <- function(mu1, mu11, mu12)
16 {
17   out <- exp(-(mu1-mu11)^2/(2*mu12^2))
18   return(out)
19 }
20
21 # ------------------------------------------------------------------------------
22
23 # target density of sigma1
24 f_sigma1 <- function(sigma1, sigma11, sigma12)
```

[83] eigene Darstellung.

```
{
  out <- sigma1^(-sigma11 - 1) * exp(-sigma12/sigma1)
  return(out)
}

# ------------------------------------------------------------------------------

# target density of gamma
f_gamma <- function(gamma, alpha, beta)
{
  out <- gamma^(alpha-1)*(1-gamma)^(beta-1)
  return(out)
}

# ------------------------------------------------------------------------------

mu1_hyperparameters <- function(mean, variance)
{
  return(c(mean, sqrt(variance)))
}

# ------------------------------------------------------------------------------

sigma1_hyperparameters <- function(mean, variance)
{
  sigma11 <- (mean^2+2*variance)/variance
  sigma12 <- mean*((mean^2+2*variance)/variance-1)
  return(c(sigma11, sigma12))
}

# ------------------------------------------------------------------------------

gamma_hyperparameters <- function(mean, variance)
{
  c <- mean/(1-mean)
  beta <- (c-(c+1)^2*variance)/((c+1)^3*variance)
  alpha <- c*beta
  return(c(alpha,beta))
}

# ------------------------------------------------------------------------------

# functions for visible and invisible output to ensure
  # that the plots can be created afterwards
create_list_wildtype = function(mu1, mu11, mu12, sigma1, sigma11, sigma12,
                                gamma, alpha, beta,
                                mu1_vec, sigma1_vec, gamma_vec)
{
  l = list(mu1 = mu1, mu11 = mu11, mu12 = mu12,
           sigma1 = sigma1, sigma11 = sigma11, sigma12 = sigma12,
           gamma = gamma, alpha = alpha, beta = beta,
           mu1_vec = mu1_vec, sigma1_vec = sigma1_vec,
           gamma_vec = gamma_vec)
  class(l) = "wildtype"
  return(l)
}

```

```
82 # ------------------------------------------------------------------------------
83
84 print.wildtype = function(x, ...){
85   x = x[c("mu1", "sigma1", "gamma", "mu11", "mu12", "sigma11", "sigma12", "alpha",
86           "beta")]
87   NextMethod()
88 }
```

Quellcode 8.3: Funktionen für die Dichteschätzung des Wild-Type.[84]

Der folgende Code ist für die Dichteschätzung des Non-Wild-Type relevant. Er enthält die a-posteriori-Verteilung des Koeffizientenvektors ϕ sowie die Funktionen `h` und `gamma.tuning`, die für den Langevin-Hastings-Algorithmus benötigt werden.

```
1  # target density of phi
2  f_phi <- function(lambda, x, P, B, C)
3  {
4    # calculate log of target density
5    eta <- B %*% x
6    pi <- normalize(exp(eta))
7    kappa <- C %*% pi
8    out <- sum(m*log(kappa))-lambda/2 * t(x) %*% P %*% x
9    return(out)
10 }
11
12 # ------------------------------------------------------------------------------
13
14 gamma.tuning <- function(iteration)
15 {
16   return(1/iteration)
17 }
18
19 # ------------------------------------------------------------------------------
20
21 h <- function(x)
22 {
23   if(x < epsilon) return(epsilon)
24   else if(x > A) return(A)
25   else return(x)
26 }
27 epsilon <- 10^(-4)
28 A <- 10^4
29
30 # ------------------------------------------------------------------------------
31
32 # functions for visible and invisible output to ensure
33   # that the plots can be created afterwards
34 create_list_nonwildtype = function(chain_vec, lambda_vec, chain_result,
35                                    lambda_result, BSplineBasis, acceptance_rate)
36 {
37   l = list(chain_vec = chain_vec, lambda_vec = lambda_vec,
38            chain_result = chain_result, lambda_result = lambda_result,
39            BSplineBasis = BSplineBasis, acceptance_rate = acceptance_rate)
40   class(l) = "nonwild"
41   return(l)
42 }
43
```

[84] eigene Darstellung.

```
44 # ------------------------------------------------------------------------------
45
46 print.nonwild = function(x, ...){
47   x = x[c("chain_result", "lambda_result", "acceptance_rate")]
48   NextMethod()
49 }
```

Quellcode 8.4: Funktionen für die Dichteschätzung des Non-Wild-Type.[85]

Schließlich folgen die Funktionen, die für die Simulationsstudie verwendet werden. Die Funktion `censored_data` erzeugt mit gegebenem Stichprobenumfang `n` zunächst eine Stichprobe der gewählten Mischverteilung und zensiert diese mithilfe eines Histogrammes mit Klassenbreiten 1. Anschließend werden x und m bestimmt, indem nur positive Anzahlen zwischen den Nullen am Anfang und Ende gespeichert werden. Die zweite Funktion berechnet für gegebenes x den Dichtewert der gewählten Mischverteilung. Da beim Histogramm der Wert für m immer am rechten Rand des Histogrammbalkens abgelesen wird, muss eine Stetigkeitskorrektur bei x stattfinden.

```
censored_data <- function(n, range)
{
  sample <- c(rnorm(0.6 * n, 2, 0.8), rnorm(0.2 * n, 4.5, 0.7),
              rnorm(0.2 * n, 7.5, 0.6))
  break_range <- sort(c(range, min(range)-1, max(range)+1))
  toEst <- hist(sample, xlim = range(break_range),
                breaks = break_range, freq= FALSE)$counts
  x <- break_range
  x <- x[ min( which ( toEst != 0 )) : max( which( toEst != 0 ))]
  m <- toEst[ min( which ( toEst != 0 )) : max( which( toEst != 0 ))]

  result <- list(x = x, m = m, sample = sample)
  return(result)
}

# ------------------------------------------------------------------------------

sim_density  <- function(x)
{
  # x+0.5 correction because of continuity
  return(0.6 * dnorm(x + 0.5, 2, 0.8) + 0.2 * dnorm(x + 0.5, 4.5,0.7) +
       0.2 * dnorm(x + 0.5,7.5,0.6))
}
```

Quellcode 8.5: Funktionen für Simulationsstudie.[86]

Im folgenden Code werden alle benötigten Parameter berechnet und in einer Liste gespeichert.

```
x <- c(0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.125, 0.25, 0.5, 1, 2, 4, 8,
       16, 32, 64, 128, 256, 512)
m <- c(0, 0, 0, 0, 0, 0, 1, 13, 84, 1847, 10593, 11342, 2181, 424, 570, 3796,
       1549, 6683, 187)
x <- x[ min( which ( m != 0 )) : max( which( m != 0 ))]
x <- log(x,2)
```

[85] eigene Darstellung.

```
m <- m[ min( which ( m != 0 )) : max( which( m != 0 ))]


factorI <- 15
J       <- length(m)
I       <- factorI * J
k       <- length(m)
Delta   <- set.Delta(x, I)
chi     <- set.chi(x, I, Delta)
u       <- set.u(x, I, chi)
C       <- CompositionMatrix(J, I)
basis   <- BSplineBasis(k, I, u, x)

iterations <- 20
it.length  <- 20000
burn_in    <- 10000

params <- list(x = x, m = m, factorI = factorI, J = J, I = I, k = k, Delta = Delta,
               chi = chi, u = u, C = C, basis = basis,
               iterations = iterations, it.length = it.length, burn_in = burn_in)
```

Quellcode 8.6: Parameterbestimmung für Dichteschätzung.[87]

8.2 Wild-Type

In diesem Kapitel befindet sich der Code zur Dichteschätzung des Wild-Type-Anteils der Verteilung. Zunächst folgt die Implementierung der nicht-linearen Regression inklusive dem Verfahren von Turnidge, J. et al. (2006), das zur Erzeugung der Startwerte für den Bayes-Ansatz benötigt wird. Dort werden für alle MIC-Kategorien größer als die `start_dilution` lineare Regressionen mit sukzessive größer werdenden Teilmengen der Daten vorgenommen und anschließend diejenige Teilmenge gewählt, bei der die Differenz zwischen wahrem und geschätzten N am geringsten ist.

```
NonLinearRegression <- function(x, y, start_dilution, start_sigma,
                                estimated_cutoff_log)
{
  # vectors for results
  N               <- vector(mode = "numeric")
  N_stderror      <- vector(mode = "numeric")
  N_diffs         <- vector(mode = "numeric")
  mu              <- vector(mode = "numeric")
  mu_stderror     <- vector(mode = "numeric")
  sigma           <- vector(mode = "numeric")
  sigma_stderror  <- vector(mode = "numeric")
  aic             <- vector(mode = "numeric")

  true_n    <- cumsum(y)
  cutoffs   <- start_dilution:(max(x)-1)  # cutoffs for for-loop
  index     <- 1


```

[87] eigene Darstellung.

```
  start_mu <- weighted.mean(x[x<=estimated_cutoff_log],
                            y[x<=estimated_cutoff_log])

  for(cutoff in cutoffs)
  {
    z <- cumsum(y[x<=cutoff])
    w <- x[x<=cutoff]

    # non-linear-least-squares
    lmod <- nls(z ~ N*pnorm(w, mu, sigma),
                start = list(N = max(z), mu = start_mu, sigma=start_sigma))
    reg_summary <- summary(lmod)

    N[index]     <- coef(lmod)[1]
    mu[index]    <- coef(lmod)[2]
    sigma[index] <- coef(lmod)[3]

    N_stderror[index]     <- reg_summary$parameters[1,2]
    mu_stderror[index]    <- reg_summary$parameters[2,2]
    sigma_stderror[index] <- reg_summary$parameters[3,2]

    aic[index] <- AIC(lmod)

    N_diffs[index] <- N[index] - max(z)

    index <- index + 1
  }

  # choose best subset of regressions
  result_index <- which(N_diffs==min(abs(N_diffs)))

  gamma <- true_n[x==cutoffs[result_index]]/sum(y)

  result <- list(N_vec = N, N_Std_errors = N_stderror,
                 mu_vec = mu, mu_Std_errors = mu_stderror,
                 sigma_vec = sigma, sigma_Std_errors = sigma_stderror,
                 aic = aic, tested_cutoffs = cutoffs,
                 wild-type-cutoff-Estimation = 2^cutoffs[result_index],
                 N_est = N[result_index], mu_est = mu[result_index],
                 sigma_est = sigma[result_index], gamma_est = gamma)
  return(result)
}
```

Quellcode 8.7: Nicht-lineare-Regression nach Verfahren von Turnidge, J. et al. (2006).[88]

Der nächste Code zeigt die Funktion, mit der die Parameter μ_1, σ_1 und γ mit Bayes-Methoden geschätzt werden. Für jeden Parameter wird aus der a-priori-Verteilung eine Stichprobe simuliert, mit einer uniformverteilten Zufallsvariable verglichen und der nächste Zustand der Markov-Kette entsprechend der Beschreibung zum Metropolis-Hastings-Algorithmus bestimmt. Die Ergebnisse der Parameter sind jeweils der Mittelwert der erzeugten Markov-Kette.

```
MIC_wild_type <- function(mu_mean_est, sigma_mean_est, gamma_mean_est,
                          it.length, burn_in, simulation,
```

[88] eigene Darstellung.

```
                          mu_variance = sqrt(0.05), sigma_variance = 0.05,
                          gamma_variance = 0.0005,
                          mu1_start = mu_mean_est,
                          sigma1_start = sigma_mean_est,
                          gamma_start = gamma_mean_est)
{
  gamma  <- vector(mode = "numeric")
  mu1    <- vector(mode = "numeric")
  sigma1 <- vector(mode = "numeric")

  gamma[1]  <- gamma_start
  mu1[1]    <- mu1_start
  sigma1[1] <- sigma1_start

  # set hyperparameters
  mu11 <- mu1_hyperparameters(mu_mean_est, mu_variance)[1]
  mu12 <- mu1_hyperparameters(mu_mean_est, mu_variance)[2]

  sigma11 <- sigma1_hyperparameters(sigma_mean_est, sigma_variance)[1]
  sigma12 <- sigma1_hyperparameters(sigma_mean_est, sigma_variance)[2]

  alpha <- gamma_hyperparameters(gamma_mean_est, gamma_variance)[1]
  beta  <- gamma_hyperparameters(gamma_mean_est, gamma_variance)[2]

  after.burnin <- burn_in:it.length


  for(i in 1:it.length)
  {
    # ------------------ mu1 ------------------------------------------
    mu.hyper    <- mu1_hyperparameters(mu1[i], mu_variance)
    proposalmu1 <- rnorm(1, mu.hyper[1], mu.hyper[2])
    u <- runif(1)

    if(f_mu1(mu1[i], mu11, mu12) == 0) accept <- TRUE
    else accept <- u < f_mu1(proposalmu1, mu11, mu12)/f_mu1(mu1[i], mu11, mu12)
    if(accept){
      mu1[i+1] <- proposalmu1
    }
    else{
      mu1[i+1] <- mu1[i]
    }

    # ----------------- sigma1 ----------------------------------------
    sigma.hyper    <- sigma1_hyperparameters(sigma1[i], sigma_variance)
    proposalsigma1 <- rinvgamma(1, shape = sigma.hyper[1], scale = sigma.hyper[2])
    u <- runif(1)

    if(f_sigma1(sigma1[i], sigma11, sigma12) == 0) accept <- TRUE
    else accept <- u < f_sigma1(proposalsigma1, sigma11, sigma12)/
                       f_sigma1(sigma1[i], sigma11, sigma12)

    if(accept){
      sigma1[i+1] <- proposalsigma1
    }
    else{
      sigma1[i+1] <- sigma1[i]
```

```
    }

    # ------------------ gamma ------------------------------------------
    gamma.hyper    <- gamma_hyperparameters(gamma[1], gamma_variance)
    proposalgamma <- rbeta(1, gamma.hyper[1], gamma.hyper[2])
    u <- runif(1)

    if(f_gamma(gamma[i], alpha, beta) == 0) accept <- TRUE
    else accept <- u < f_gamma(proposalgamma, alpha, beta)/
                       f_gamma(gamma[i], alpha, beta)

    if(accept){
      gamma[i+1] <- proposalgamma
    }
    else{
      gamma[i+1] <- gamma[i]
    }

  }

  # results are means of vectors
  mu1_res    <- mean(mu1[after.burnin])
  sigma1_res <- mean(sigma1[after.burnin])
  gamma_res  <- mean(gamma[after.burnin])

  list <- create_list_wildtype(mu1_res, mu11, mu12, sigma1_res, sigma11, sigma12,
    gamma_res, alpha, beta, mu1, sigma1, gamma)
  return(list)
}
```

Quellcode 8.8: Bayes-Schätzung des Wild-Type.[89]

Hier folgt die Funktion, welche die Bayes-Schätzung mehrmals wiederholt, um so robustere Ergebnisse zu erzielen.

```
Monte_Carlo_wild_type <- function(iterations, it.length, burn_in, simulation,
                                  mu_mean_est, sigma_mean_est, gamma_mean_est,
                                  chi)
{
  I <- length(chi) - 1

  wildtype     <- matrix(data = NA, ncol = 3, nrow = iterations)
  mu_matrix    <- matrix(data = NA, ncol = iterations, nrow = it.length + 1)
  sigma_matrix <- matrix(data = NA, ncol = iterations, nrow = it.length + 1)
  gamma_matrix <- matrix(data = NA, ncol = iterations, nrow = it.length + 1)

  individual_estimates <- matrix(data = NA, ncol = I, nrow = iterations)

  for(i in 1:iterations)
  {
    estimate <- MIC_wild_type(mu_mean_est, sigma_mean_est, gamma_mean_est,
                                        it.length, burn_in)
    wildtype[i,]     <- c(estimate$mu1, estimate$sigma1, estimate$gamma)
    mu_matrix[,i]    <- estimate$mu1_vec
    sigma_matrix[,i] <- estimate$sigma1_vec
    gamma_matrix[,i] <- estimate$gamma_vec
```

[89] eigene Darstellung.

```
22
23      pi_1 <- vector(mode = "numeric")
24      for(j in 2:(I+1))
25      {
26        # correction for continuity
27        pi_1[j-1] <- pnorm(chi[j]-0.5, estimate$mu1, estimate$sigma1) -
28                     pnorm(chi[j-1]-0.5, estimate$mu1, estimate$sigma1)
29      }
30
31      individual_estimates[i, ] <- pi_1
32    }
33    parameters <- colMeans(wildtype)
34
35    mu1    <- parameters[1]
36    sigma1 <- parameters[2]
37    gamma  <- parameters[3]
38
39    mu1_vector    <- estimate$mu1_vec
40    sigma1_vector <- estimate$sigma1_vec
41    gamma_vector  <- estimate$gamma_vec
42
43    cutoff <- if(simulation == 0) 2^round(qnorm(0.999, mu1, sigma1))
44              else 2^round(qnorm(0.999, mu1, sigma1) - 0.5)
45
46    pi_1 <- vector(mode = "numeric")
47    for(i in 2:(I+1))
48    {
49      # correction for continuity
50      pi_1[i-1] <- pnorm(chi[i]-0.5, mu1, sigma1) -
51        pnorm(chi[i-1]-0.5, mu1, sigma1)
52    }
53
54    create_list_wildtype_MonteCarlo(mu1, estimate$mu11, estimate$mu12,
55                                    sigma1, estimate$sigma11, estimate$sigma12,
56                                    gamma, estimate$alpha, estimate$beta,
57                                    mu1_vector, sigma1_vector, gamma_vector,
58                                    cutoff, individual_estimates, pi_1)
59  }
```

Quellcode 8.9: Monte-Carlo-Simulation für den Wild-Type.[90]

Abschließend folgt der verwendete Code zur Dichteschätzung des Wild-Type-Anteils. Zunächst wird der erste Modalwert für den Parameter `start_dilution` bestimmt, bei der die sukzessiven nicht-linearen Regressionen starten. Die Variable `nonLinReg` enthält nach der Berechnung die Ergebnisse der Regression, unter anderem alle einzelnen Ergebnisse, welche in Abbildung 5.1 dargestellt sind, sowie die Werte für N, μ_1 und σ_1, die anschließend in der Monte-Carlo-Simulation als Startwerte verwendet werden.

```
1 # NonLinear Regression ------------------------------------------------------
2
3 start_dilution <- firstMode(params$x,params$m)
4 nonLinReg <-NonLinearRegression(params$x, params$m, start_dilution, start_sigma = 1,
5                   estimated_cutoff_log = 3)
6
```

[90] eigene Darstellung.

```
# MCMC ------------------------------------------------------------------

wildtype_result <- Monte_Carlo_wild_type(params$iterations, params$it.length,
                                         params$burn_in,simulation = 0,
                                         mu_mean_est = nonLinReg$mu_est,
                                         sigma_mean_est = nonLinReg$sigma_est,
                                         gamma_mean_est = nonLinReg$gamma_est,
                                         params$chi)
pi_1 <- wildtype_result$pi_1
```

Quellcode 8.10: Endgültige Dichteschätzung vom Wild-Type.[91]

8.3 Non-Wild-Type

Dieser Teil widmet sich der nicht-parametrischen Dichteschätzung. Bei dem nachfolgenden Code handelt es sich um die Funktion, die den Langevin-Hastings-Algorithmus durchführt, welcher in Kapitel 5.3 bereits beschrieben wurde.

```
MIC_non_wild_type_LH <- function(x, m, factorI, u, C, basis, lambda.start,
                                 it.length = 20000, k = length(m), a = 0.0001,
                                 b = 0.0001, diff_order = 2, burn_in = 10000
                                 )
{
  after.burnin <- burn_in:it.length
  J <- length(m)   # number of given data points
  I <- factorI * J

  D <- diff(diag(k), diff = diff_order)
  P <- t(D) %*% D
  Rp <- qr(P)$rank

  data <- rep(normalize(m),rep(factorI, length(m)))  # percentage
  phi  <- ginv(basis) %*% log(data)
  phi  <- normalize(phi)   # identifiability constraint

  lambda    <- vector(mode = "numeric")
  lambda[1] <- lambda.start

  delta    <- vector(mode = "numeric")
  delta[1] <- 1.65^2/k^(1/3)

  acceptance_rate_vec <- vector(mode = "numeric")

  chain <- matrix(data = NA, ncol = k, nrow = it.length + 1)
  chain[1, ] <- phi

  for(i in 1:it.length)
  {
    G.prev <- Gradient(k, chain[i,], lambda[i], P)
    proposal <- mvrnorm(1, chain[i,] + 0.5 * delta[i] * P %*% G.prev ,
                        delta[i] * P)
    G.cur <- Gradient(k, proposal, lambda[i], P)

```

[91] eigene Darstellung.

```
    rand <- runif(1)

    alpha <- min(1, (exp(f_phi(lambda[i], proposal, P, basis, C)-
                           f_phi(lambda[i], chain[i,], P, basis, C))) *
                    exp(-1/2 * t(G.cur + G.prev) %*%
                          ((proposal - chain[i,]) +
                             delta[i]/4 * P %*% (G.cur - G.prev))))
    accept <- rand < alpha

    if(accept){
      chain[i+1,] <- proposal
      acceptance_rate_vec[i] <- 1

    }
    else{
      chain[i+1,] <- chain[i,]
      acceptance_rate_vec[i] <- 0

    }
    lambda[i+1] <- rgamma(1, shape = a + 0.5 * Rp,
                           rate = b + 0.5 * t(chain[i+1,]) %*% P %*% chain[i+1,])

    delta.next.root <- h(sqrt(delta[i]) + gamma.tuning(i)*(alpha - 0.574))
    delta[i+1] <- delta.next.root^2

  }

  chain_result <- colMeans(chain[after.burnin,])
  lambda_result <- mean(lambda[after.burnin])
  acceptance_rate <- sum(acceptance_rate_vec)/length(acceptance_rate_vec)

  list <- create_list_nonwildtype(chain, lambda, chain_result,
                                  lambda_result, basis, acceptance_rate)
  return(list)

}
```

Quellcode 8.11: Bayes-Schätzung des Non-Wild-Type.[92]

Auch für diesen Teil der Verteilung wurde eine Funktion implementiert, die für robustere Ergebnisse die Bayes-Schätzung mehrmals wiederholt.

```
Monte_Carlo_non_wild_type <- function(iterations, it.length, burn_in,
                                      x, m, factorI, u, C, basis,
                                      lambda.start = 0.0001, a = 0.0001, b = 0.0001,
                                      diff_order = 2, k = length(m)
                                      )
{
  D <- diff(diag(k), diff = diff_order)
  P <- t(D) %*% D
  Rp <- qr(P)$rank

  nonwildtype      <- matrix(data = NA, ncol = k, nrow = iterations)
  individual_estimates <- matrix(data = NA, ncol = I, nrow = iterations)
  acceptance_rate <- vector(mode = "numeric")
  lambda_res <- vector(mode = "numeric")
```

[92] eigene Darstellung.

```
  Delta <- u[2]-u[1]
  for(i in 1:iterations)
  {
    estimate <- MIC_non_wild_type_LH(x, m, factorI, u, C, basis, lambda.start,
                                     it.length = it.length, burn_in = burn_in)
    nonwildtype[i,]          <- estimate$chain_result
    individual_estimates[i,] <- normalize(exp(basis%*%estimate$chain_result))
    acceptance_rate[i]       <- estimate$acceptance_rate
    lambda_res[i]            <- estimate$lambda_result
  }
  phi <- colMeans(nonwildtype)
  pi_2 <- normalize(exp(basis%*%phi))
  f_2 <- pi_2/Delta

  lambda <- mean(lambda_res)

  result <- list(individual_estimates = individual_estimates, phi = phi,
                 pi_2 = pi_2, f_2 = f_2, acceptance = acceptance_rate,
                 lambda = lambda)
  return(invisible(result))
}
```

Quellcode 8.12: Monte-Carlo-Simulation für den Non-Wild-Type.[93]

Die Dichteschätzung des Non-Wild-type kann mithilfe des folgenden Codes realisiert werden.

```
nonwildtype_result <- Monte_Carlo_non_wild_type(iterations, it.length, burn_in,
                                                params$x, params$m, params$factorI,
                                                params$u, params$C, params$basis
                                                )
pi_2 <- nonwildtype_result$pi_2
```

Quellcode 8.13: Endgültige Dichteschätzung vom Non-Wild-Type.[94]

8.4 Gesamtschätzung der Dichte

Für die Gesamtschätzung der Dichte wird wie in Kapitel 5.4 beschrieben eine alternative Definition von π verwendet. Es werden von der Schätzung des Wild-Type alle Werte bis zum berechneten ECOFF und ab dieser Grenze alle Werte der nicht-parametrischen Schätzung genommen. Anschließend muss dieser Vektor normalisiert werden, um eine Dichte zu repräsentieren.

```
cutoff_log <- log(wildtype_result$wild_type_cutoff, 2)
pi <- c(pi_1[params$u <= cutoff_log],pi_2[params$u > cutoff_log])
pi <- normalize(pi)
```

Quellcode 8.14: Gesamtbetrachtung der Dichte.[95]

[93] eigene Darstellung.
[94] eigene Darstellung.
[95] eigene Darstellung.

Literaturverzeichnis

Andrews, J. M. (2001): *Determination of minimum inhibitory concentrations*. Journal of Antimicrobial Chemotherapy **48**, Suppl. S1, 5-16.

Atchadé, Y. F., Rosenthal, J. S. (2005): *On adaptive Markov chain Monte Carlo algorithms*. Bernoulli **11** 815-828.

Behrends, E. (2000): *Introducing to Markov Chains: with Special Emphasis on Rapid Mixing*. Wiesbaden, Springer.

BfR - Bundesinstitut für Risikobewertung (2016): `http://www.bfr.bund.de/de/escherichia_coli-54352.html`. [Eingesehen am 07.11.2016].

Boor, C. de (1993): *B(asic)-spline basics*. In: Piegl, L. (Hg.): *Fundamental Developments of Computer-Aided Geometric Modelling*. San Diego, Academic Press, 27-49.

Burnham, K., Anderson, D. (2002): *Model Selection and Multimodal Inference*. 2. Ed. New York, Springer.

Casella, G., Robert, C. P. (2010): *Introducing Monte Carlo Methods with R*. Berlin, Springer.

Chemie.de (2017): `http://www.chemie.de/lexikon/Penicillin.html`. [Eingesehen am 21.03.2017].

Dellaportas, P., Roberts, G. O. (2003): *Spatial Statistics and Computational Methods*. New York, Springer.

Efromovich, S. (1999): *Nonparametric Curve Estimation: Methods, Theory and Applications*. 2. Aufl. New York, Springer.

Eilers, P. H. C. (2007): *Ill-posed problems with counts, the composite link model, and penalized likelihood*. Statistical Modelling **7** 239-254.

Eilers, P. H. C., Marx, B. D. (1996): *Flexible smoothing with B-splines and Penalties*. Statistical Science **11** 89-121.

EUCAST (2016): `http://www.eucast.org`. [Eingesehen am 21.03.2017].

Fahrmeir, L., Kneib, T., Lang, S. (2009): *Regression: Modelle, Methoden und Anwendungen*. 2. Aufl. Berlin/Heidelberg/New York, Springer.

Fortmann-Roe, S. (2012): *Understanding the Bias-Variance Tradeoff*. `http://scott.fortmann-roe.com/docs/BiasVariance.html`. [Eingesehen am 24.10.2016].

Gelman, A., Carlin, J. B., Stern, H. S. et al. (2014) *Bayesian Data Analysis*. 3. Ed. Florida, CRC Press.

Gentle, J. E. (2003): *Random Number Generation and Monte Carlo Methods*. 2. Ed. New York, Springer.

Gilks, W. R., Richardson, S., Spiegelhalter, D. J. (1996): *Markov Chain Monte Carlo in Practice*. Dordrecht, Springer Science+Business Media.

Jaspers, S., Aerts, M., Verbeke, G. and Beloeil, P. A. (2014a): *Estimation of the wild-type minimum inhibitory concentration value distribution*. Statistics in Medicine **33** 289-303.

Jaspers, S., Aerts, M., Verbeke, G. and Beloeil, P. A. (2014b): *A new semi-parametric mixture model for interval censored data, with applications in the field of antimicrobial resistance*. Computational Statistics and Data Analysis **71** 30-42.

Jaspers, S., Lambert, P., Aerts, M. (2016): *A Bayesian approach to the semi-parametric estimation of a MIC distribution*. Submitted to the Annals of Applied Statistics.

Kolonko, M. (2008): *Stochastische Simulation: Grundlagen, Algorithmen und Anwendungen*. Wiesbaden, Vieweg+Teubner.

Lambert, P., Eilers, P. H. C. (2009): *Bayesian density estimation from grouped continuous data*. Computational Statistics and Data Analysis **53** 1388-1399.

McCullagh P., Nelder J. A. (1989): *Generalized Linear Models*. 2. Ed. London, Chapman & Hall.

MIC-EUCAST (2017): `https://mic.eucast.org/Eucast2/regShow.jsp?Id=1529`. [Eingesehen am 03.02.2017].

Palumbi, S. R. (2001): *Humans as the world's greatest evolutionary force*. Science **293(5536)** 1786-1790.

Papula, L. (2011): *Mathematik für Ingenieure und Naturwissenschaftler: ein Lehr- und Arbeitsbuch für das Grundstudium. Band 3*. 6., überarb. und erw. Aufl. Wiesbaden, Vieweg+Teubner.

R Core Team (2015): *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria. `http://www.R-project.org/`.

Rizzi, S., Gampe, G., Eiler, P. H. C. (2015): *Efficient Estimation of Smooth Distributions From Coarsely Grouped Data.* American Journal of Epidomology **182(2)** 138-147.

Roberts, G. O., Rosenthal, J. S. (1998): *Optimal scaling of discrete approximations to Langevin diffusions.* Journal of the Royal Statistical Society, Series **B60** 225-268.

Roberts, G. O., Tweedie, R. L. (1996). *Exponential convergence of Langevin distributions and their discrete approximations.* Bernoulli **24** 341-363.

Runge, C. (1901): *Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten.* Zeitschrift für Mathematik und Physik **46** 224-243.

Snipes, M., Taylor, D. C. (2014): *Model selection and Akaike Information Criteria: An example from wine ratings and prices.* Wine Economics and Policy **3(2014)** 3–9.

Thompson, R., Baker, R. J. (1981): *Composite link functions in generalized linear models.* The American Statistician **30** 125-131.

Turnidge, J., Kahlmeter, G., Kronvall, G. (2006): *Statistical characterisation of bacterial wild-type MIC value distributions and the determination of epidemiological cut-off values.* Clinical Microbiology and Infection **12** 418-425.

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit eigenständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Marburg, den 3. April 2017