

EBERHARD-KARLS-UNIVERSITÄT TÜBINGEN

WISSENSCHAFTLICHE ARBEIT

Modellierung und Analyse von Räuber-Beute-Populationsdynamiken

Autor:
Sanja STEGERER

Betreut durch:
Prof. Rainer NAGEL

Arbeitsbereich:
Funktionalanalyse



4. Januar 2016²

¹ Quelle: <https://500px.com/photo/741583/cheetah-chasing-gazelle-by-austin-thomas>
Datum: 10.09.2015, 12.37 Uhr PST

² Titelseite entwickelt anhand Vorlage von: https://de.wikibooks.org/wiki/TeX/_Eine_Titelseite_erstellen Datum: 10.09.2015, 12.59 Uhr PST

Danksagung

Auf dieser Seite möchte ich den Menschen danken, die mich durch den Prozess des Schreibens meiner wissenschaftlichen Arbeit begleitet und unterstützt haben.

In erster Linie möchte ich meinem Betreuer Prof. Rainer Nagel danken. Er hat mir die Möglichkeit gegeben, meine Zulassungsarbeit meinen Lebensumständen entsprechend in Los Angeles zu schreiben und war trotzdem immer für Fragen erreichbar. Er hat mir alle Freiheiten gelassen, das Thema meiner Zulassungsarbeit zu bestimmen und sie dann auch nach meinen Vorstellungen zu entwerfen, aber gleichzeitig Schranken gesetzt, wenn ich vor lauter Ideen über Bord zu gehen drohte. Deswegen an dieser Stelle ein großes Dankeschön an die tolle Betreuung durch Prof. Nagel.

Weiter möchte ich meinem Ehemann und meinen Eltern danken, die mich durch die ganze Zeit des Schreibens unterstützt und mit Rat und Tat bei wichtigen Entscheidungen geholfen haben. Besonders mein Ehemann hat mir mit seiner moralischen Unterstützung immer den Rücken gestärkt und aus den gelegentlichen Motivationstiefs heraus geholfen.

Inhaltsverzeichnis

I Einleitung	1
1 Modelle in der Biologie	1
2 Überblick	2
II Die Räuber-Beute-Beziehung - Biologischer Exkurs	3
3 Einführung in die Populationsökologie	3
4 Die Räuber-Beute-Beziehung (Allgemeines)	3
4.1 Räuber und der Einfluss von Koevolution auf Eigenschaften beider Populationen	3
III Die Räuber-Beute-Beziehung - Modellierung und Analyse	6
5 Das Modell von Lotka und Volterra	7
5.1 Modellierung	7
5.1.1 Modellannahmen	7
5.1.2 Das Differenzialgleichungssystem	8
5.2 Grafische Analyse des Lotka-Volterra-Differenzialgleichungssystems	10
5.2.1 Gleichgewichtslösungen	10
5.2.2 Richtungsvektoren von Lösungen mit Anfangswerten außerhalb der Gleichgewichtslösungen	12
5.3 Qualitative Analyse des Lotka-Volterra Differenzialgleichungssystems	15
5.3.1 Existenz und Eindeutigkeit von Systemen von Differenzialgleichungen	15
5.3.2 Stabilität der Gleichgewichtspunkte	17
5.3.3 Linearisierung	18
5.3.4 Die Lösung des linearisierten Lotka-Volterra Modells	19
5.3.5 Eigenwertanalyse	24
Phasenportrait von G_1 :	24
Phasenportrait von G_2 :	25
5.3.6 Stabil, asymptotisch stabil, instabil	28
5.3.7 Skizzieren der Lösung des nichtlinearen Lotka-Volterra Modells .	29
5.3.8 Schlussfolgerung	32
5.4 Schwachpunkte des Lotka-Volterra Modells	33
Verbesserungsmöglichkeiten:	34

6 Modellerweiterung – Logistisches Wachstum der Beutepopulation	36
6.1 Modellierung	36
6.1.1 Modellannahmen	36
6.1.2 Das Differenzialgleichungssystem	38
6.2 Grafische Analyse des logistischen Räuber-Beute Modells	39
6.2.1 Gleichgewichtslösungen	39
6.2.2 Richtungvektoren von Lösungen mit Anfangswerten außerhalb der Gleichgewichtslösungen	41
Abschätzung:	41
6.3 Qualitative Analyse des logistischen Modells	45
6.3.1 Stabilität der Gleichgewichtspunkte	45
6.3.2 Linearisierung im Gleichgewichtspunkt	45
6.3.3 Die Lösung der Linearisierung um G₂ und G₃	47
6.3.4 Eigenwertanalyse von G₂ und G₃	53
Einordnen in die 3 Oberkategorien	53
Zusammenfassung:	56
Kurvenverlauf in der Phasenebene	57
6.3.5 Skizzieren der nichtlinearen Lösung des logistischen Räuber-Beute Modells	60
6.4 Diskussion des logistischen Lotka-Volterra Modell	64
6.4.1 Vergleich zu Lotka-Volterra Modell	64
Vergleich der Modelle für die Beutepopulation:	65
Vergleich der Modelle für die Räuberpopulation:	66
Zusammenfassung und Verbesserungsvorschläge:	66
7 Abschluss	68
Appendix	70
A Differenzialgleichungen	70
A.1 Gewöhnliche Differenzialgleichung erster Ordnung	70
A.2 Lösungen von Differenzialgleichungen	72
B Matrixexponentialfunktion	73
B.1 Die Matrixexponentialfunktion	73
B.2 Sätze und Definitionen für die Matrixexponentialfunktion und ihre Asymptotik	74
C Die Lösung eines zweidimensionalen linearen Systems von Differenzialgleichungen	78
D Programme zur Erstellung der Graphiken	81
D.1 Lotka-Volterra Modell	81
D.1.1 2D-Grafiken	81
D.1.2 3D-Grafiken und Trajektorie	93

D.2 Logistisches Lotka-Volterra Modell	98
D.3 Grafiken für Finale Diskussion	119
D.4 Grafiken für Appendix	124

Teil I

Einleitung

1 Modelle in der Biologie

Mathematische Modellierung in den Naturwissenschaften ist heutzutage eines der wichtigsten Handwerkzeuge zum Abstrahieren der komplexen Vorgänge und Interaktionen in der Natur.

Gerade in der Biologie werden sie mit wachsendem Wissen immer wichtiger und unabdingbarer. Zwei herauszuhebende Beispiele sind die Molekularbiologie und die Naturschutzwissenschaft.

Seit dem Start des Humangenomprojekts (1990) ([Hum]) hat das Feld der Molekularbiologie immense Fortschritte gemacht (siehe [PEDP99]). Diese versprechen unter dem Schlagwort "personalisierte Medizin" baldige Möglichkeiten zur Erkennung der einer Krankheit zugrunde liegenden molekularen Zusammenhänge und schafft Hoffnung auf eine Heilung einer Vielzahl an Krankheiten ([PEDP99]).

Die Zusammenhänge und Interaktionen von Molekülen und Genen, sowie deren Einwirkung auf die Zelle, das Organ und den gesamten Körper, in dem diese Zelle lebt, wurden erst in den letzten Jahren entdeckt. Dabei wird ganz nach den Worten von Hans Künig, "Wer viel weiß, weiß auch, dass er nichts weiß, - zumindest, wenn er weise ist"³ klar, dass mit mehr Wissen über diese Zusammenhänge auch die Komplexität der Interaktionen und Abhängigkeiten zunimmt und neue bessere Modelle, sowie computergestützte Methoden benötigt werden, um sie für das menschliche Verständnis zugänglich zu machen (siehe [Mat], Importance). Eine Zusammenarbeit von Mathematikern, Molekularbiologen und Computerwissenschaftlern wird in diesem Feld immer wichtiger.

Die Populationsökologie (für die Definition siehe Kapitel 3) ist eine der ersten biologischen Felder, in denen überhaupt mathematische Modelle entwickelt wurden. Das Vorhandensein von Pflanzen und Wildtieren war für die Menschen schon immer von lebensnotwendiger Bedeutung, sei es als Nahrungs-, sowie Fell- und Lederlieferanten, aber auch als Bedrohung durch Raubtiere. Somit ist es nachvollziehbar, dass das erste mathematische Modell in der Biologie versucht, grundlegende Populationsdynamiken zu beschreiben.

Diese Publikation wurde von Thomas Robert Malthus 1798 mit dem Titel "An Essay on the Principle of Population" (Wiederveröffentlichung in 2012 siehe [Mal12]) veröffentlicht. Diese inspirierte unter anderem Pierre Francois Verhulst, welcher 1838 das logistische Wachstumsmodell für Populationen beschrieb (siehe [Log]), aber auch die Begründer der bis heute anerkannten Evolutionstheorie Charles R. Darwin und Alfred

³Der Anfang aller Dinge: Naturwissenschaft und Religion, Piper 2005, S. 91
gefunden in https://de.wikiquote.org/wiki/Hans_K%C3%BCng

Russel Wallace⁴, wobei Wallace zusätzlich Begründer der Inselbiogeographie-Theorie ist (siehe [Thi12], 5-6 Artenschutz im Räumlichen Kontext, Inselbiogeographie, S.12).

Seither wurden von Mathematikern und Biologen viele weitere Modelle entwickelt und verbessert. Aufgrund der in den letzten Jahren drastisch erhöhten Fähigkeit, Daten zu sammeln und zu verarbeiten, erhielt dieses Feld der Schnittstelle von Mathematik und Biologie einen weiteren Aufschwung (siehe [Mat], Importance). Nicht nur haben sich die Fähigkeit, diese Modelle zu nutzen, verbessert, sondern auch die Möglichkeiten präzisere Modelle für spezifische Populationsinteraktionen zu entwickeln und für den Naturschutz nutzbar zu machen.

Das Erstellen und Nutzen solcher Modelle ist mit steigender Wahrnehmung des drohenden sechsten großen Artensterbens (siehe [Thi12], Einführung S.3 und [Jac]) besonders essentiell im Natur- und Artenschutz geworden. Eine gute und möglichst realistische Vorhersage der Entwicklung wichtiger Populationen innerhalb eines Ökosystems kann Naturschützern die Möglichkeit geben, passend in die Dynamiken einzutreten und somit über das Fortbestehen oder Aussterben des gesamten Ökosystems zu bestimmen.

2 Überblick

In dieser wissenschaftlichen Arbeit werden zwei der ersten und grundlegenden mathematische Modelle zur Modellierung der Beziehung zwischen Räubern und ihrer Beute beschrieben und analysiert. Hierfür wird zu Beginn der biologische Hintergrund im Rahmen der Populationsökologie angeführt um die Modellierung in biologischen Kontext zu bringen. Historische Hintergründe zu den einzelnen Modellen werden in den jeweiligen Kapiteln ebenfalls weiter ausgeführt.

Im mathematischen Teil dieser Arbeit wird mit dem bekanntesten und einfachsten Modell begonnen: Das Lotka-Volterra Modell. Zuerst werden die Modellannahmen angeführt und das Modell beschrieben. Es folgt eine grafische, sowie qualitative Analyse des Modells. Als zweites Modell wird das logistische Lotka-Volterra Modell behandelt und analog zum Lotka-Volterra Modell grafisch und qualitativ analysiert. Im Hintergrund wurden numerische Methoden für die grafische Darstellung der Modelle benutzt. Der Code hierfür wird im Appendix D angeführt.

⁴siehe [DW58] für den Brief von Wallace an Darwin, noch vor der Veröffentlichung von Darwins berühmten Werk "The Origin of species" (siehe [Dar09]), geschrieben und später von "the Linnean Society of London" als Journal-Artikel publiziert. Siehe außerdem <http://www.100welshheroes.com/en/biography/alfredrussellwallace> Datum: 14.09.2015 15.29 Uhr PST, sowie [Thi12], 5-6 Artenschutz im Räumlichen Kontext, Inselbiogeographie, S.12

Teil II

Die Räuber-Beute-Beziehung - Biologischer Exkurs

3 Einführung in die Populationsökologie

Die Räuber-Beute-Beziehung ist ein Teilgebiet der Populationsökologie (Teilbereich der Ökologie), der sich im Speziellen mit der Interaktion von Populationen verschiedener oder gleicher Arten untereinander und mit ihrer Umwelt beschäftigt ([Pop]). Hierbei untersucht sie im Speziellen die Größen von Populationen, zu geringeren Teilen auch ihre Verteilungen und die Prozesse, welche zu beidem führen ([BMT96], Preface, S. vii). Diese werden mit mathematischen Modellen beschrieben ([BMT96], Preface, S. vii). Sie ermöglichen ein besseres Verständnis von den Verhältnissen und Zusammenhängen von Populationsdynamiken in der realen Welt, zu Teilen besser als Beobachtungen der realen Welt selbst (übersetzt aus [BMT96], Preface, S. vii). Eine allgemeine Populationsdynamik kann durch 4 grundlegende Prozesse beschrieben werden: Geburt (G), Tod (T), Einwanderung (E) und Auswanderung (A), die einer einfachen Gleichung

$$N_{t+1} = N_t + G - T + E - A$$

genügen, wobei N_t die Populationsgröße zum Zeitpunkt t , und N_{t+1} die Populationsgröße derselben Population eine Zeitperiode später bei $t+1$ darstellt ([BMT96], Chapter 1: Describing Populations, S.4). Ein Anstieg in der Populationsgröße kann beobachtet werden, falls die Zahl der Geburten und Einwanderungen größer ist als die der Todesfälle und Auswanderungen.

4 Die Räuber-Beute-Beziehung (Allgemeines)

Die Räuber-Beute-Beziehung ist eine besondere Beziehung zwischen mindestens zwei Populationen. Hierbei fressen Individuen der einen Population, die so genannten Räuber, Teile oder Individuen einer anderen Population, die so genannte Beute.

4.1 Räuber und der Einfluss von Koevolution auf Eigenschaften beider Populationen

Räuber können in 4 unterschiedliche Gebiete unterteilt werden. ([BMT96], Chapter 5: Predation, Section 1: Introduction S. 117)⁵

- Echte Räuber:
Räuber, welche andere Tiere töten und fressen, um das eigene Überleben zu sichern.

⁵Quelle für die Aufzählung siehe ebenfalls [BMT96]

- Parasitoide:

Dies sind Organismen, meist Insekten der Ordnung Hymenoptera, welche im Adult-Stadium unabhängig von einer bestimmten anderen Population leben, aber ihre Eier in unmittelbarer Nähe oder direkt an Individuen der Beutepopulation legen. Die aus den Eiern schlüpfenden Larven ernähren sich im Larvenstadium ausschließlich von der Beute, was gegen Ende des Larvenstadiums zum Tod des Beutindividuums führt.

- Parasiten:

Organismen, welche den Großteil ihres Lebens in unmittelbarer Nähe oder direkt an oder in der Beute, in diesem Fall "Wirtspopulation" genannt, leben, ihre Nahrung von dem Wirt beziehen und dabei dessen Fitness reduzieren. In den meisten Fällen führt eine Besiedelung eines Wirts durch einen Parasit nicht zum Tod des Wirts.

- Herbivoren:

Dies sind pflanzenfressende Tiere, welche ihr Überleben durch den teilweisen oder ganzen Fraß von Pflanzen sichern. Dabei gibt es Herbivoren, welche den Tod einer Pflanze direkt durch komplettes Auffressen hervorrufen und andere, welche die Fitness der Pflanze durch teilweises Fressen verringern, aber nicht direkt töten. In vielen Fällen jedoch führt der Konsum durch Herbivoren direkt oder indirekt zum Tod der Pflanze.

Populationen, welche in einer Räuber-Beute-Beziehung miteinander leben, beeinflussen sich gegenseitig stark in ihrer Evolution und ihren Eigenschaften. Diese Art von Evolution wird als Koevolution bezeichnet.

Koevolution ist ein Entwicklungsprozess, welcher, abhängig von Umgebungs- und Populationseigenschaften, schnell oder langsam laufen kann. Die Populationen erzeugen hierbei Teile des der Evolution zu Grunde liegenden Selektionsdruck ([BMT96], Chapter 5: Predation, Section 3: Coevolution S. 120), welcher sich aus vielen kleinen Selektionsdrücken zusammensetzt:

- Räuberpopulation:

Ein Selektionsdruck, welcher Individuen mit effizienterer Beute-Wahl und Beute-Beschaffung einen Vorteil gegenüber anderen Individuen mit anderen Methoden gibt. Diese Individuen haben deswegen eine größere Fitness und eine höhere Wahrscheinlichkeit zur Fortpflanzung und somit Weitergabe ihrer Gene. Durch die Weitergabe der Gene ist allerdings nicht gesichert, dass genau dieses Vorteil bringende Verhalten an die nächste Generation weitergegeben wird. Es besteht jedoch die Chance auf Anpassung der ganzen Population, da die Nachkommen dieses Individuums wieder eine höhere Fitness als ihre Artgenossen und somit eine höhere Wahrscheinlichkeit zur Fortpflanzung haben.

- Beutepopulation:

Ein Selektionsdruck, welcher Individuen mit besseren Flucht-, Tarn- oder sonstigen Abwehrmechanismen eine höhere Überlebenswahrscheinlichkeit und somit

eine höhere Wahrscheinlichkeit zur Fortpflanzung und somit zur Weitergabe ihrer Gene an die nächste Generation gibt ([BMT96], Chapter 5: Predation, Section 3: Coevolution, S. 120). Auch hier ist wieder nicht gesichert, dass die Vorteil bringenden Eigenschaften durch die Gene an die nächste Generation weiter gegeben werden. Wie bei der Räuber-Population besteht jedoch die Chance auf Anpassung der ganzen Population.

Hierbei können Individuen der Beutepopulation durch eine Anpassung in eine bestimmte Richtung ein breiteres Spektrum an Räubern abwehren als sich umgekehrt die Räuber an mehrere Beutepopulationen anpassen können. Räuberpopulationen verlieren häufig durch die Anpassung an eine bestimmte Beutepopulation die Fähigkeit weitere Beutepopulationen effizient zu bejagen ([BMT96], Chapter 5: Predation, Section 3: Coevolution, S. 120).

Dies führt dazu, dass Räuberpopulationen in den meisten Fällen eine "Lieblings"-Beutepopulation haben. Diese zeichnet sich dadurch aus, dass die Räuber durch die Bejagung dieser Beute am effizientesten ihre Nahrungsbedürfnisse abdecken können und eine Bejagung dieser Beute besonders häufig beobachtet werden kann ([BMT96], Chapter 5: Predation, Section 3: Coevolution, S. 120). Als Beispiel für eine solche Räuber-Beute-Beziehung kann der kanadische Luchs und der Schneeschuhhase genannt werden ([Kan], Hunting and Diet)

Ausgehend von der Variation und Größe der Population an Beutetieren haben sich deswegen 2 Grobkatagorien von Räubern entwickelt: Generalisten und Spezialisten. In einer Umwelt von geringem Selektionsdruck auf die Räuberpopulation durch viele Beutepopulationen ist der Räuber nicht gezwungen, sich auf seine Lieblings-Beuteart zu konzentrieren und kann diese bei Bedarf wechseln. Diese Räuber nennt man Generalisten.

In Umgebungen, in denen nur wenige Beutearten vorkommen, ist es für die Räuberpopulationen essenziell, möglichst effizient eine bestimmte Beuteart bejagen zu können, da sonst die Extinktion droht. Solche Räuber nennt man Spezialisten. In solchen Gebieten gilt dasselbe in umgekehrter Form auch für die Beute, weswegen gerade dort Koevolution besonders schnell stattfinden kann.

Teil III

Die Räuber-Beute-Beziehung - Modellierung und Analyse

Es stellt sich nun die Frage, welches mathematische Modell eine reale Räuber-Beute-Populationsdynamik reproduzieren kann und wie gut dieses Modell die Realität wieder gibt. Des Weiteren ist interessant ob mit diesem Modell die weitere Entwicklung beider Populationen vorhergesagt werden kann. Dies ist in der heutigen Zeit besonders wertvoll, da in den letzten Jahren mehr und mehr Arten aussterben ([Thi12], Einführung S.3 und [Jac]) und die noch Lebenden den Schutz der Menschen benötigen. Gerade die Beziehung zwischen Räubern und ihrer Beute kann ein ganzes Ökosystem beeinflussen (Siehe [Thi12], Einführung, S.37), man nennt solche Räuberarten dann "keystone species" ([Thi12], Einführung, S.39), aufgrund ihrer Schlüsselrolle in diesem Ökosystem, und deswegen ist die Vorhersage dieser Entwicklung ein wichtiger Bestandteil des aktuellen Artenschutzes (Siehe Folien zu "Keystone Species" in [Thi12], Einführung, S. 37-42).

Das erste Modell zur Beschreibung einer Räuber-Beute Beziehung stammt von Alfred J. Lotka (1925) und Vito Volterra (1926). Wobei das Modell von beiden je unabhängig entwickelt wurde ([Lot]). Es wird deswegen auch das Lotka-Volterra Modell für die Räuber-Beute-Beziehung genannt.

Das zweite in dieser Arbeit betrachtete Modell integriert logistisches Wachstum für die Beute-Population in das Lotka-Volterra Modell. Das Modell für logistisches Wachstum für Populationen und die logistische Gleichung wurde 1838 von Pierre-Francois Verhulst erstmals entwickelt (Siehe [Log], Applications), welche auch von Alfred J. Lotka 1925 nochmals unter dem Namen "law of population growth" (siehe [Log], Applications) neu entwickelt und niedergeschrieben wurde. Heute wird die logistische Gleichung auch außerhalb der Modellierung von Populationswachstum verwendet (z.B. Logistische Regression und zur Modellierung Neuronaler Netze im Feld machine learning in den Computerwissenschaften oder zur Modellierung von Tumor-Wachstum in den Medizinwissenschaften (siehe [Log]), Applications).

5 Das Modell von Lotka und Volterra

5.1 Modellierung

5.1.1 Modellannahmen

Sei P die Anzahl der Individuen einer Population von Räubern (bezeichne die Räuberpopulation im Folgenden mit P) und N die Anzahl der Individuen einer Population von Beutetieren der Räuberpopulation P (bezeichne die Beutepopulation im Folgenden mit N).

Ziel ist es, $P(t)$ und $N(t)$ in Abhängigkeit der Zeit t zu finden.

Das Lotka-Volterra Modell hat 4 grundlegende Annahmen.⁶:

1. Die Beutepopulation wächst in Abwesenheit des Räubers exponentiell.
2. In Anwesenheit der Räuberpopulation sinkt die Beutepopulation proportional zur Interaktionshäufigkeit von Räuber- und Beutepopulation.
3. Die Räuberpopulation stirbt in Abwesenheit der Beutepopulation mit exponentieller Sterberate aus.
4. In Anwesenheit der Beutepopulation wächst die Räuberpopulation proportional zur Interaktionshäufigkeit von Räuber- und Beutepopulation.

Hieraus können die folgenden Differenzialgleichungen formuliert werden:

1. Sei $a > 0$ die Wachstumsrate der Beutepopulation. Dann wird das Wachstum der Beutepopulation beschrieben durch

$$\frac{dN}{dt}(t) = a \cdot N(t), \quad t \geq 0.$$

2. Die Häufigkeit von Interaktionen der beiden Populationen ist abhängig von der Größe beider Populationen. Je größer eine Population, desto größer die Wahrscheinlichkeit, dass Interaktion zwischen den Populationen stattfinden, desto größer die Interaktionshäufigkeit. Die Interaktionshäufigkeit kann also modelliert werden durch das Produkt der Populationen $P(t) \cdot N(t)$. Sei $b > 0$ außerdem der Sterbeparameter der Beute bei Interaktionen von Räuber- und Beutepopulationen, dann ergibt sich für die Beutepopulation

$$\frac{dN}{dt}(t) = aN(t) - b \cdot N(t)P(t), \quad t \geq 0.$$

3. Sei $d > 0$ die Sterberate der Räuberpopulation in Abwesenheit der Beutepopulation. Dann kann die Abnahme der Räuberpopulation beschrieben werden als

$$\frac{dP}{dt}(t) = -d \cdot P(t), \quad t \geq 0.$$

⁶Die 4 grundlegenden Annahmen wurden annehmend an den online-Kurs von Prof. Paul Blanchard, Boston University und Patrick Cummings auf der website www.edX.org mit dem Namen BUx Math226.1x (die Übersichtsseite zum Kurs siehe [[EdX](#)]) entwickelt. Das Video hierzu befindet sich in Module 10, Systems of Differential Equations, Video 2.

4. Die Häufigkeit der Interaktionen zwischen beiden Populationen kann wieder durch $P(t) \cdot N(t)$ beschrieben werden mit $c > 0$ als Wachstumsparameter für die Räuberpopulation. Daraus ergibt sich für die Räuberpopulation

$$\frac{dP}{dt}(t) = c \cdot P(t)N(t) - dP(t), \quad t \geq 0.$$

5.1.2 Das Differenzialgleichungssystem

Folgerung 5.1.2.1 (Lotka-Volterra-Gleichungen).

Aus den Modellannahmen ergibt sich das folgende System von Differenzialgleichungen

$$\begin{aligned} \frac{dN}{dt}(t) &= N(t)(a - bP(t)), \\ \frac{dP}{dt}(t) &= P(t)(cN(t) - d). \end{aligned} \quad (1)$$

Ein solches System kann auch in Vektorschreibweise umformuliert werden. Sei hierbei

$$x(t) := \begin{pmatrix} N(t) \\ P(t) \end{pmatrix}, \quad \frac{dx}{dt}(t) := \begin{pmatrix} \frac{dN}{dt}(t) \\ \frac{dP}{dt}(t) \end{pmatrix}.$$

Somit ergibt sich für das Differenzialgleichungssystem von Lotka-Volterra

$$\frac{dx}{dt}(t) = \begin{pmatrix} \frac{dN}{dt}(t) \\ \frac{dP}{dt}(t) \end{pmatrix} = \begin{pmatrix} N(t)(a - bP(t)) \\ P(t)(cN(t) - d) \end{pmatrix} =: \begin{pmatrix} f_N(N(t), P(t)) \\ f_P(N(t), P(t)) \end{pmatrix} = \begin{pmatrix} f_N(x(t)) \\ f_P(x(t)) \end{pmatrix}. \quad (1^*)$$

Ziel ist nun, die Lösungen der Differenzialgleichungen zu finden, um die tatsächlichen Populationsgrößen zu erhalten.

Bemerkung 5.1.2.2.

Das Differenzialgleichungssystem (1) aus Folgerung 5.1.2.1

$$\begin{aligned} \frac{dN}{dt}(t) &= N(t)(a - bP(t)), \\ \frac{dP}{dt}(t) &= P(t)(cN(t) - d) \end{aligned} \quad (1)$$

ist ein System von autonomen gewöhnlichen Differenzialgleichungen erster Ordnung mit den Lösungen $P(t)$ und $N(t)$.

Für eine Einführung in Differenzialgleichungen siehe Appendix A und insbesondere Definitionen A.1.1 für gewöhnliche Differenzialgleichungen erster Ordnung, A.1.2 für autonome Differenzialgleichungen und A.2.3 für Systeme von Differenzialgleichungen. Zum Vertieften Lesen, sowie eine detaillierte Einführung in Differenzialgleichungen siehe [TP63].

Im folgenden Kapitel wird das Lotka-Volterra Modell hauptsächlich grafisch analysiert. Hierbei wird zuerst nach der einfachsten Lösung eines Systems von Differenzialgleichungen gesucht, die sogenannten Gleichgewichtslösungen (siehe Definition 5.2.1.1). Diese geben dem bis jetzt noch völlig unbekannten Verlauf der Lösungen Struktur, welche als Grundlage für die weitere Analyse dient.

5.2 Grafische Analyse des Lotka-Volterra-Differenzialgleichungssystems

5.2.1 Gleichgewichtslösungen

Definition 5.2.1.1 (Gleichgewichtslösungen).

Sei g eine Differenzialgleichung wie in Definition A.1.1. Sei $x(t)$ eine Lösung der Differenzialgleichung g . Man nennt $x(t)$ eine Gleichgewichtslösung von g , falls

$$\frac{dx}{dt}(t) = 0.$$

Theorem 5.2.1.2.

Die Gleichgewichtslösungen zum Lotka-Volterra-Differenzialgleichungssystem (1) sind

$$G_1(t) = \begin{cases} N(t) = 0 \\ P(t) = 0 \end{cases} \quad G_2(t) = \begin{cases} N(t) = \frac{d}{c} \\ P(t) = \frac{a}{b} \end{cases}$$

Beweis.

- Lösung $G_1(t)$ ist klar.

- Für $P(t) = 0$ oder $N(t) = 0$ gilt: Aus $P(t) = 0$ folgt $N(t) = 0$ und umgekehrt und führt somit zu $G_1(t)$.

Zeige nur den Fall $P(t) = 0$, da Fall $N(t) = 0$ analog geht.

Es ist

$$0 = \frac{dN}{dt}(t) = N(t)(c \cdot 0 - d) = -N(t) \cdot d \iff N(t) = 0.$$

- Für $G_2(t)$ betrachte den Fall $P(t) \neq 0$ und $N(t) \neq 0$. Es gilt also

$$\begin{aligned} 0 &= N(t)(a - bP(t)), \\ 0 &= P(t)(cN(t) - d). \end{aligned}$$

Somit

$$\begin{aligned} 0 &= cN(t) - d, & \Leftrightarrow & d = cN(t) & \Leftrightarrow & \frac{d}{c} =: N(t), \\ 0 &= a - bP(t), & \Leftrightarrow & a = bP(t) & \Leftrightarrow & \frac{a}{b} =: P(t). \end{aligned}$$

□

Es gibt also zwei Zustände, in denen sich die Populationsgrößen beider Populationen nicht verändern. Dabei ist die Gleichgewichtslösung $G_1(t)$ (Es existieren keine Populationen) nicht weiter interessant.

Bei Gleichgewichtslösung $G_2(t)$ hingegen existieren Individuen in beiden Populationen im Gleichgewichtszustand.

Es stellt sich die Frage, wie sich die Dynamik entwickelt, wenn die Start-Populationsgrößen nicht die Gleichgewichts-Populationsgrößen sind.

Da die Gleichungen im Lotka-Volterra Modell voneinander abhängig sind, ist es in Grafiken sinnvoll, die Funktionen $P(t)$ und $N(t)$ gegeneinander aufzuzeigen, und die t -Achse in der Grafik wegfallen zu lassen. Diese Ebene nennt man **N-P-Ebene** oder auch **Phasenebene**.

Falls eine spezifische Lösung des Differenzialgleichungssystems existiert, nennt man die Lösungskurve in der Phasenebene auch **Trajektorie oder Phasenkurve** ([MG87], S 42).

Die Veränderungen, welche durch t hervorgerufen werden, werden in der Phasenebene indirekt dargestellt.

Die folgende Abbildung zeigt die Gleichgewichtslösungen der Gleichungen des Lotka-Volterra Modells mit gegebenen Parametern.

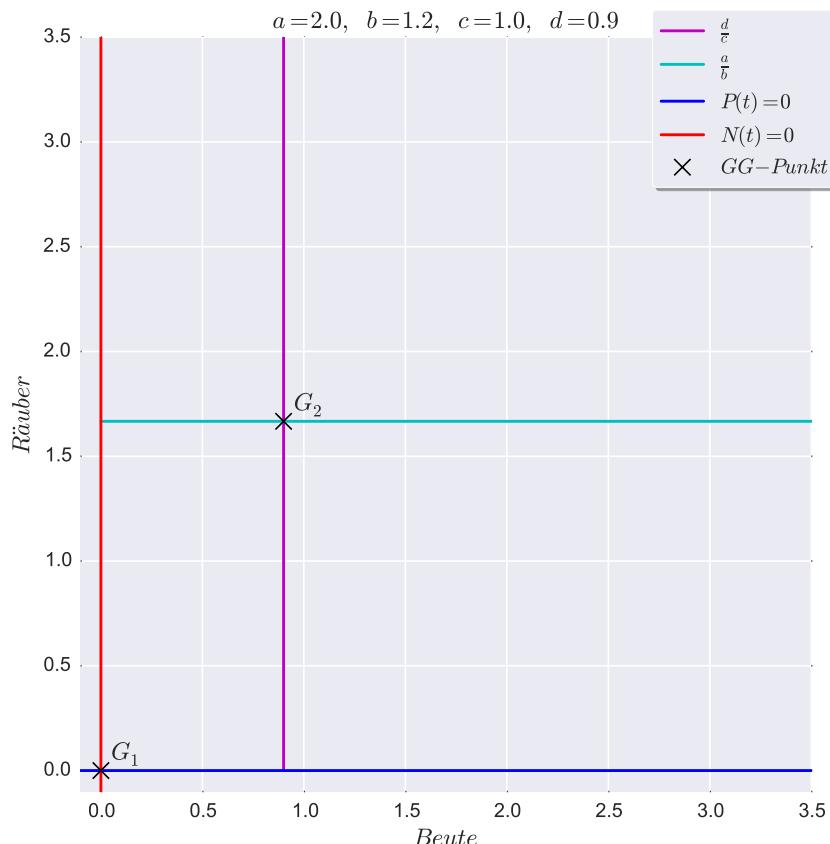


Abbildung 1: Die 4 Gleichgewichtsgeraden mit den zugehörigen Gleichgewichtspunkten

5.2.2 Richtungsvektoren von Lösungen mit Anfangswerten außerhalb der Gleichgewichtslösungen

Die Gleichgewichtslösung $G_2(t)$ ist in der Phasenebene als Punkt repräsentiert, weswegen Gleichgewichtslösungen auch häufig **Gleichgewichtspunkte** in der Phasenebene genannt werden. Der Gleichgewichtspunkt $G_2 := (\frac{d}{c}, \frac{a}{b})$ ist genau der Schnittpunkt der beiden Geraden $N(t) = \frac{d}{c}$ und $P(t) = \frac{a}{b}$.

Als nächsten Schritt gilt es abzuschätzen, wie sich die Populationen verändern, falls eine oder beide Populationen nicht ihre Gleichgewichtsgröße haben.

Betrachte hierfür die Differenzialgleichungen einzeln. Für die Abschätzung von $\frac{dP}{dt}$ wird $N(t)$ jeweils oberhalb und unterhalb seiner Gleichgewichtslösung betrachtet und umgekehrt.

$$1. \frac{dN}{dt}(t) = N(t)(a - bP(t))$$

- Sei $P(t) > \frac{a}{b}$, dann gilt

$$\frac{dN}{dt}(t) = N(t)(a - bP(t)) < N(t)(a - b \cdot \frac{a}{b}) = 0, \quad \text{also} \quad \frac{dN}{dt}(t) < 0.$$

- Sei $P(t) < \frac{a}{b}$, dann gilt

$$\frac{dN}{dt}(t) = N(t)(a - bP(t)) > N(t)(a - b \cdot \frac{a}{b}) = 0, \quad \text{also} \quad \frac{dN}{dt}(t) > 0.$$

- $\frac{dP}{dt}(t) = P(t)(cN(t) - d)$

- Sei $N(t) > \frac{d}{c}$, dann gilt

$$\frac{dP}{dt}(t) = P(t)(cN(t) - d) > P(t)(c \cdot \frac{d}{c} - d) = 0, \quad \text{also} \quad \frac{dP}{dt}(t) > 0.$$

- Sei $N(t) < \frac{d}{c}$, dann gilt

$$\frac{dP}{dt}(t) = P(t)(cN(t) - d) < P(t)(c \cdot \frac{d}{c} - d) = 0, \quad \text{also} \quad \frac{dP}{dt}(t) < 0.$$

Die folgende Abbildung zeigt diese Abschätzung. Hierbei hat nur die Pfeilrichtung Bedeutung, die Längen der Pfeile haben keine Bedeutung.

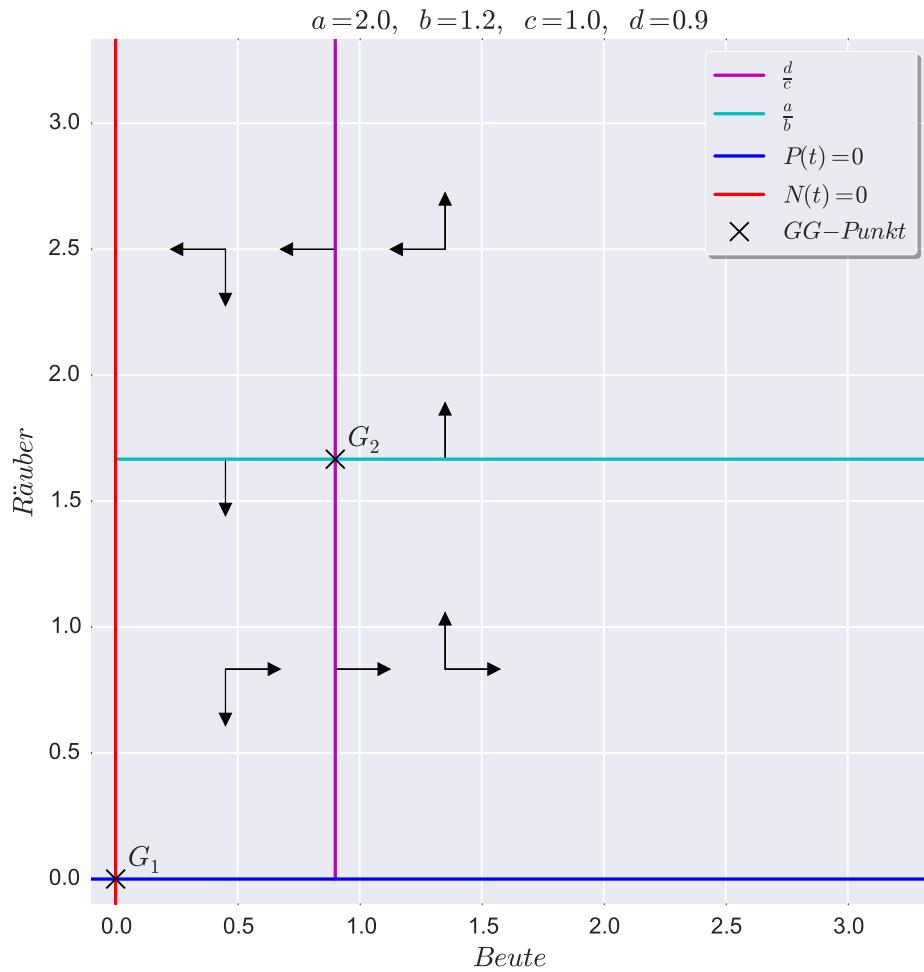


Abbildung 2: Richtungspfeile aufgrund obiger Abschätzung

Die Pfeile deuten darauf hin, dass sich die Populationen kreisförmig um G_2 bewegen.

Definition 5.2.2.1 (Richtungsfeld).

Mithilfe einer computergestützten Auswertung der Differenzialgleichung in regelmäßigen Abständen für verschiedene Werte von $P(t)$, sowie $N(t)$ kann man eine Grafik erstellen, welche die Werte der Differenzialgleichung in Form von Pfeilrichtung und Pfeillänge darstellt. Eine solche Abbildung nennt man **Richtungsfeld** in der Phasenebene.

Bemerkung 5.2.2.2.

Die folgende Grafik zeigt das Richtungsfeld aus Theorem 5.1.2.1 für die Parameter $a = 2$, $b = 1.2$, $c = 1$, $d = 0.9$ des Lotka Volterra Modells. Sie wurde erstellt mit der `quiver()`-Funktion in python (Benutzt wurde die Library Matplotlib für Grafiken in python, für die Dokumentation der Funktion `quiver`, siehe [Pytb]). Zur Betrachtung des Codes zum Erstellen dieser Grafik siehe D.1.1 Zeilen 126-182 und den Aufruf in Zeile 423.

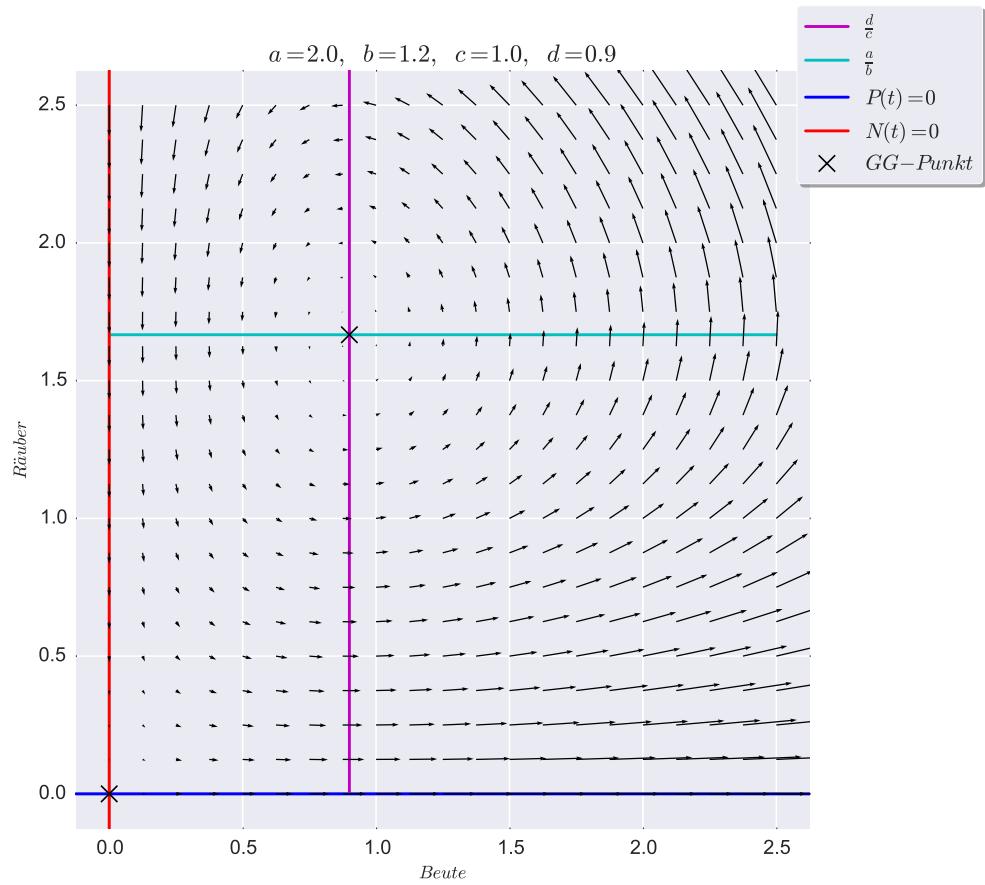


Abbildung 3: Richtungsfeld für das Lotka-Volterra Modell mit 0,05-facher Skalierung der Richtungsvektoren

Die Richtungspfeile deuten eine kreisförmige Bewegung der beiden Populationen um den Gleichgewichtspunkt G_2 , mit langsamer Änderungsrate in der Nähe des Gleichgewichtspunktes und schnellerer bei weiter entfernten Punkten, an. Im Folgenden soll mehr über den genauen Verlauf der Funktionen $P(t)$ und $N(t)$ in der Nähe von G_2 herausgefunden werden.

Dabei wird untersucht, ob sich die Populationen auf den Gleichgewichtspunkt zu oder davon weg bewegen, oder ob keines von beidem zutrifft und sie in gleichem Abstand zum Gleichgewichtspunkt bleiben und periodisch sind.

5.3 Qualitative Analyse des Lotka-Volterra Differenzialgleichungssystems

In diesem Abschnitt wird das Lotka-Volterra Modell qualitativ analysiert. Dies ist die Analyse der Eigenschaften der Lösungskurven, ohne die Lösung selbst zu berechnen ([Enc]). Dadurch erhält man den Überblick über das Verhalten der Lösungskurven mit unterschiedlichen Anfangswerten außerhalb der Gleichgewichtslösungen.

Die folgenden Fragen sollen in diesem Kapitel behandelt werden.

1. Sind die Gleichgewichtslösungen anziehend oder abstoßend für Lösungen mit Anfangswerten in der Nähe der Gleichgewichtslösungen?
2. Wie sieht das Verhalten für $t \rightarrow \infty$ aus?
3. Können Lösungskurven skizziert werden?

Das Kapitel wird mit Grundlagen und der Existenz und Eindeutigkeit von Lösungen von Differenzialgleichungen und Systemen von Differenzialgleichungen begonnen und im Anschluss auf die 3 Leitfragen eingegangen.

5.3.1 Existenz und Eindeutigkeit von Systemen von Differenzialgleichungen

Definition 5.3.1.1 (Anfangswertproblem eines Systems von gewöhnlichen Differenzialgleichungen).

Sei ein n -dimensionales System an Differenzialgleichungen wie in Definition A.2.3 gegeben und ein Anfangswert

$$b = (b_1, b_2, \dots, b_n)^T,$$

sodass

$$\begin{aligned} y_1(t_0) &= b_1 \\ y_2(t_0) &= b_2 \\ &\vdots \\ y_n(t_0) &= b_n. \end{aligned}$$

Dann nennt man das System von Differenzialgleichungen zusammen mit dem Anfangswert ein **Anfangswertproblem**.

Folgerung 5.3.1.2 (Anfangswert für Lotka-Volterra Modell).

Das Differenzialgleichungssystem (1) aus Theorem 5.1.2.2 für das Lotka-Volterra Modell in Vektorschreibweise (1) mit Anfangswert b kann zum Anfangswertproblem*

$$\frac{dx}{dt}(t) = \begin{pmatrix} f_N(x(t)) \\ f_P(x(t)) \end{pmatrix}, \quad b = x_0 \tag{2*}$$

zusammengefasst werden.

Bemerkung 5.3.1.3.

In der Modellierung gibt es in den meisten Fällen einen Ist-Zustand als Anfangswert. Um Klarheit über die Existenz und Eindeutigkeit einer Lösung zu erhalten, werden im Folgenden die grundlegenden Sätze hierfür zitiert.

Theorem 5.3.1.4 (Existenz und Eindeutigkeit für die Lösung von Differenzialgleichungen).

Sei $\frac{dx}{dt}(t) = f(t, x)$ eine beschränkte, stetige Funktion von t und $x(t)$ in $S \in \mathbb{R}^2$ ((t, x)-Ebene). Sei außerdem ein Punkt $(t_0, x_0) \in S$ gegeben.

Die Funktion $f(t, x(t))$ genüge der Bedingung für die Lipschitz-Stetigkeit in S . Es existiert also für je 2 Punkte $(t, x), (t, \tilde{x}) \in S$ eine Konstante N , sodass

$$|f(t, x) - f(t, \tilde{x})| \leq N \cdot |x - \tilde{x}| \quad \forall (t, x), (t, \tilde{x}) \in S.$$

Dann existiert ein Intervall

$$I_0 : |x - x_0| < h, \quad h > 0,$$

auf dem genau eine einzige stetige Funktion $x(t)$ mit stetiger Ableitung auf I_0 existiert, für die gilt

$$\frac{dx}{dt}(t) = f(t, x(t)),$$

$$x(t_0) = x_0.$$

Beweis. Siehe [TP63], S. 734 ff. □

Theorem 5.3.1.5 (Existenz und Eindeutigkeit für ein System von Differenzialgleichungen erster Ordnung).

Sei das Differenzialgleichungssystem

$$\frac{dy_1}{dt}(t) = f_1(t, y_1(t), y_2(t), \dots, y_n(t)),$$

$$\frac{dy_2}{dt}(t) = f_2(t, y_1(t), y_2(t), \dots, y_n(t)),$$

⋮

$$\frac{dy_n}{dt}(t) = f_n(t, y_1(t), y_2(t), \dots, y_n(t)),$$

und der Anfangswert

$$y_1(t_0) = b_1,$$

$$y_2(t_0) = b_2,$$

⋮

$$y_n(t_0) = b_n,$$

mit $b \in \mathbb{R}^n$, $f_1, \dots, f_n \in S$ stetig, gegeben. Hierbei sei S definiert durch

$$|t - t_0| \leq k_0, \\ |y_1(t) - b_1| \leq k_1, \dots, |y_n(t) - b_n| \leq k_n.$$

Jede der Funktionen f_1, \dots, f_n sei Lipschitzstetig, d.h. für je 2 Punkte $(t, y_1, y_2, \dots, y_n), (t, \bar{y}_1, \bar{y}_2, \dots, \bar{y}_n)$ für alle $i = 1, \dots, n$ $\exists N$ Konstante, sodass gilt

$$|f_i(t, y_1, \dots, y_n) - f_i(t, \bar{y}_1, \dots, \bar{y}_n)| \leq N \cdot (|y_1 - \bar{y}_1| + |y_2 - \bar{y}_2| + \dots + |y_n - \bar{y}_n|).$$

Dann existiert ein Intervall

$$I_0 : |t - t_0| < h \quad h > 0$$

in dem genau eine Familie von stetigen Funktionen $y_1(t), \dots, y_n(t)$ existiert, deren stetige Ableitungen das gegebene Differenzialgleichungssystem mit gegebenem Anfangswert erfüllen.

Beweis. Siehe [TP63], S. 764 ff. □

Bemerkung 5.3.1.6.

Für eine etwas andere Formulierung mit vollständigem Beweis des Existenz- und Eindeutigkeitssatzes siehe [HSD13] Kapitel 17, S. 385 ff.

Für ein vertieftes Studium von Eindeutigkeit und Existenz der Lösung von Differenzialgleichungen erster Ordnung, sowie Systemen von Differenzialgleichungen erster Ordnung siehe [CL55] Chapter 1 und [TP63].

5.3.2 Stabilität der Gleichgewichtspunkte

Um die Leitfragen 1 (Eigenschaften des Gleichgewichtspunkts) und 2 (Verhalten der Lösung für $t \rightarrow \infty$) zu beantworten ist die Linearisierung des Modells in seinen Gleichgewichtspunkten typischerweise der erste Schritt zur Analyse der lokalen Stabilität des nichtlinearen Differenzialgleichungssystems (siehe [Lin]).

Lineare Systeme sind meistens einfacher zu analysieren und es sind bereits Lösungsmethoden zur Betrachtung des Verhaltens für $t \rightarrow \infty$ bekannt. Außerdem zeigen linearisierte Systeme nahe der Gleichgewichtslösung in den meisten Fällen ähnliches Verhalten wie ihre nichtlinearen Originale. Dies wird am Ende des Kapitels im Satz 5.3.6.3 gezeigt.

Wir werden also für Lösungen nahe des Gleichgewichtspunkts das Verhalten des linearisierten Lotka-Volterra Modells betrachten und, wenn möglich, Rückschlüsse auf das tatsächliche System ziehen.

5.3.3 Linearisierung

Linearisierung eines Systems kann erreicht werden durch Linearisierung beider Gleichungen mithilfe der Taylorreihe (siehe [Tay]). Die daraus resultierenden linearen Gleichungen bilden dann ein System von Linearen Differenzialgleichungen (zur Definition von Systemen linearer Differenzialgleichungen siehe C.0.1). Da im Gleichgewichtspunkt linearisiert werden soll, wird für die Herleitung ein allgemeiner Gleichgewichtspunkt $G = (g_1, g_2)$ angenommen. Es gilt also

$$\frac{dN}{dt}(t) \approx \underbrace{f_N(g_1, g_2)}_{=0} + \frac{\partial f_N}{\partial N(t)}(g_1, g_2) \cdot (N(t) - g_1) + \frac{\partial f_N}{\partial P(t)}(g_1, g_2) \cdot (P(t) - g_2),$$

$$\frac{dP}{dt}(t) \approx \underbrace{f_P(g_1, g_2)}_{=0} + \frac{\partial f_P}{\partial N(t)}(g_1, g_2) \cdot (N(t) - g_1) + \frac{\partial f_P}{\partial P(t)}(g_1, g_2) \cdot (P(t) - g_2).$$

Die Partiellen Ableitungen für das Lotka Volterra Modell sind

$$\frac{\partial f_N}{\partial N(t)} = \frac{\partial(N(t) \cdot (a - P(t)b)}{\partial N(t)} = a - P(t) \cdot b$$

$$\frac{\partial f_N}{\partial P(t)} = \frac{\partial(N(t)a - N(t)P(t)b)}{\partial P(t)} = -N(t)b$$

$$\frac{\partial f_P}{\partial N(t)} = \frac{\partial(P(t) \cdot cN(t) - P(t)d)}{\partial N(t)} = c \cdot P(t)$$

$$\frac{\partial f_P}{\partial P(t)} = \frac{\partial(P(t) \cdot (cN(t) - d)}{\partial P(t)} = c \cdot N(t) - d$$

Mit der Jacobi-Matrix

$$J_{P,N} = \begin{pmatrix} \frac{\partial f_N}{\partial N(t)}(g_1, g_2) & \frac{\partial f_N}{\partial P(t)}(g_1, g_2) \\ \frac{\partial f_P}{\partial N(t)}(g_1, g_2) & \frac{\partial f_P}{\partial P(t)}(g_1, g_2) \end{pmatrix} = \begin{pmatrix} a - g_2b & -g_1b \\ g_2c & cg_1 - d \end{pmatrix}$$

kann das linearisierte Gleichungssystem umgeschrieben werden in

$$\begin{pmatrix} \frac{dN}{dt}(t) \\ \frac{dP}{dt}(t) \end{pmatrix} \approx J_{P,N} \cdot \begin{pmatrix} N(t) - g_1 \\ P(t) - g_2 \end{pmatrix}. \quad (\text{L1})$$

Mit der Linearisierung durch die Taylor Reihe wird ein System von Gleichungen berechnet, welches den Punkt, in dem linearisiert wurde, ebenfalls annimmt.

Wir benötigen für die weitere Analyse des linearisierten Systems den genauen Gleichgewichtspunkt jedoch nicht und es ist generell einfacher, wenn man ein System mit dem Ursprung als Gleichgewichtspunkt betrachtet. Betrachte deswegen hier O.B.d.A.

das in den Ursprung verschobene System

$$\begin{aligned}
\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} &= \begin{pmatrix} \frac{dN}{dt}(t) \\ \frac{dP}{dt}(t) \end{pmatrix} + J_{P,N} \cdot \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \\
&\approx J_{P,N} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} - J_{P,N} \cdot \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} + J_{P,N} \cdot \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \\
&= J_{P,N} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} \\
&= \begin{pmatrix} a - g_2 b & -g_1 b \\ g_2 c & c g_1 - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix}. \tag{L2}
\end{aligned}$$

Setzen wir nun die tatsächlichen Werte von $G_2(t)$ und $G_1(t)$ in L2 ein, erhalten wir die zwei Gleichungssysteme

- $G_1 = (0, 0)$

$$\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} \approx \begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix}.$$

- $G_2 = (\frac{d}{c}, \frac{a}{b})$

$$\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} \approx \begin{pmatrix} a - \frac{a}{b}b & -\frac{d}{c}b \\ \frac{a}{b}c & c\frac{d}{c} - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = \begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix}.$$

5.3.4 Die Lösung des linearisierten Lotka-Volterra Modells

Aus der Vorlesung "Lineare Dynamische Systeme" von Prof. Nagel ([Nag12]) ist bekannt, dass die Lösung eines linearen Systems durch die Matrix-Exponentialfunktion dargestellt werden kann. Genaueres dazu ist in Appendix B zu finden. Die Berechnung der Matrix-Exponentialfunktion für diagonalisierbare Matrizen ist besonders einfach. Es wird deswegen mit der Berechnung der Eigenwerte in den beiden Gleichgewichtspunkten des linearisierten Lotka-Volterra Modells gestartet, um die Diagonalisierbarkeit der Matrix zu bestimmen.

1. Für G_1 : Dazu ist

$$A = \begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix}$$

bereits in Diagonalform mit den Eigenwerten

$$\lambda_1 = a, \quad \lambda_2 = -d.$$

2. Für G_2 : Für

$$A = \begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix}$$

ist das charakteristische Polynom

$$p_A(\lambda) = \lambda^2 - \left(-\frac{d}{c}b \cdot \frac{a}{b}c\right) = \lambda^2 - da.$$

Dann ergibt sich für

$$\lambda^2 - \left(-\frac{d}{c}b \cdot \frac{a}{b}c\right) \stackrel{!}{=} 0 \iff \lambda = \pm i\sqrt{ad}.$$

Es gilt also

$$\lambda_1 = i\sqrt{ad}, \quad \lambda_2 = -i\sqrt{ad}.$$

Beide Matrizen sind somit diagonalisierbar.

Der folgende Satz vereinfacht die Lösung eines linearen Systems mit diagonalisierbarer Matrix⁷. In Appendix C wird ein Überblick über weitere Lösungsmöglichkeiten zweidimensionaler Systeme gegeben und außerdem generell mögliche Eigenwerte eines solchen Systems betrachtet.

Satz 5.3.4.1 (A diagonalisierbar).

Sei ein zweidimensionales lineares Differenzialgleichungssystem wie in Definition C.0.1 gegeben und A sei diagonalisierbar. Dann ist die Lösung dieses Differenzialgleichungssystems gegeben als

$$z(t) = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t}.$$

Dabei sind v_1, v_2 Eigenvektoren zu Eigenwerten λ_1, λ_2 und c_1, c_2 allgemeine Konstanten, welche durch die Anfangswerte des Systems festgelegt werden.

Beweis. Da $A \in L(X)$ diagonalisierbar ist, existiert ein $S, S^{-1} \in L(X)$, wobei $L(X)$ die Banachalgebra der Operatoren auf dem Vektorraum X ist (siehe [Nag12]), sodass $S^{-1}AS = D$ gilt, mit

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \quad S = (v_1, v_2).$$

Hierbei sind v_1, v_2 Eigenvektoren zu den Eigenwerten λ_1, λ_2 .

Die Lösung des Differenzialgleichungssystems ist also

$$z(t) = e^{tA} \cdot b = Se^{tD}S^{-1}b.$$

Fasst man $S^{-1} \cdot b = (c_1, c_2)^T$ zusammen und multipliziert diese Gleichung aus, erhält man die Lösung. \square

⁷Entwickelt anhand Vorlage in [Sto08], S. 7ff.

Es folgt die Lösung für das linearisierte Lotka-Volterra Modell in der Nähe der Gleichgewichtspunkte.

1. Für G_1 :

- Berechnung der Eigenvektoren:

- Für λ_1 erhalten wir

$$\begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix} \cdot v_1 = a \cdot v_1 \iff \begin{array}{lcl} a \cdot v_{11} & = & a \cdot v_{11} \\ -d \cdot v_{12} & = & a \cdot v_{12} \end{array} \iff v_1 = \begin{pmatrix} \xi_1 \\ 0 \end{pmatrix}.$$

- Für λ_2 erhalten

$$\begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix} \cdot v_2 = -d \cdot v_2 \iff \begin{array}{lcl} a \cdot v_{21} & = & -d \cdot v_{21} \\ -d \cdot v_{22} & = & -d \cdot v_{22} \end{array} \iff v_2 = \begin{pmatrix} 0 \\ \xi_2 \end{pmatrix}.$$

Wähle der Einfachheit halber $\xi_1 = 1$ und $\xi_2 = 1$.

- Die Lösung ist dann

$$\begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = c_1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} e^{ta} + c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot e^{-td}.$$

In der folgenden Grafik wurde für Beispielwerte $a = 2$, $b = 1.2$, $c = 1$, $d = 0.9$ und Anfangswert $A_0 = (1, 1)^T$ die Lösung von $N(t)$ sowie $P(t)$ gegen die Zeit aufgetragen.

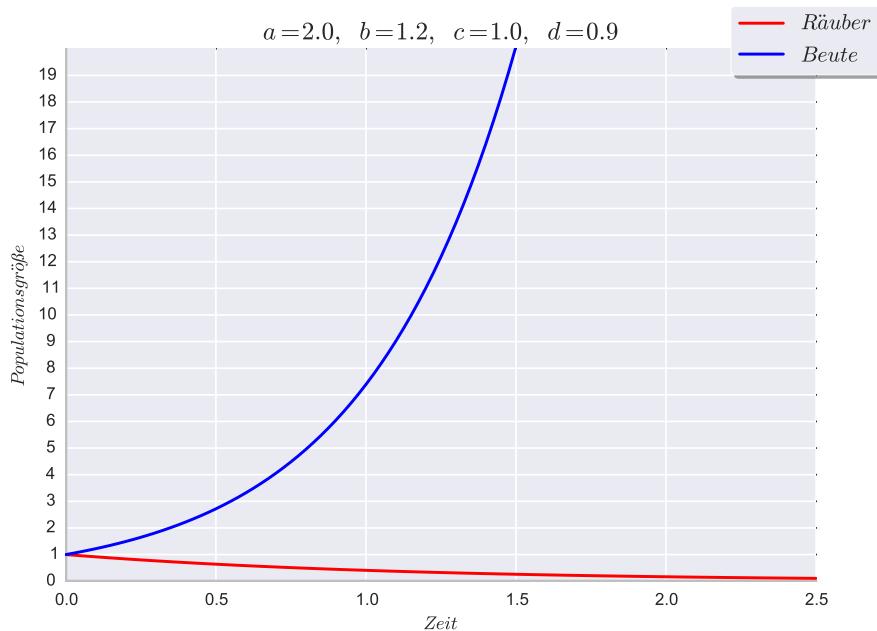


Abbildung 4: Linearisierte Lösung um G_1 , Populationsgrößen gegen die Zeit aufgetragen

2. Für G_2 :

- Berechnung der Eigenvektoren:

– Für λ_1 gilt

$$\begin{aligned}
 & \begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} \cdot v_1 = i\sqrt{ad} \cdot v_1 \\
 \iff & \begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} \cdot v_1 - i\sqrt{ad} \cdot v_1 = 0 \\
 \iff & \left(\begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} - i\sqrt{ad} \cdot I \right) \cdot v_1 = 0 \\
 \iff & \begin{pmatrix} -i\sqrt{ad} & -\frac{d}{c}b \\ \frac{a}{b}c & -i\sqrt{ad} \end{pmatrix} \cdot \begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix} = 0 \\
 \iff & \begin{pmatrix} -i\sqrt{ad} & -\frac{d}{c}b \\ 0 & -i\sqrt{ad} + i\sqrt{ad} \end{pmatrix} \cdot \begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix} = 0 \\
 \iff & v_1 = \begin{pmatrix} \frac{\frac{d}{c}b}{-i\sqrt{ad}} \\ 1 \end{pmatrix},
 \end{aligned}$$

wobei $v_{12} = 1$ gewählt wurde.

– Für λ_2 gilt

$$\begin{aligned}
 & \begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} \cdot v_2 = -i\sqrt{ad} \cdot v_2 \\
 \iff & \begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} \cdot v_2 + i\sqrt{ad} \cdot v_2 = 0 \\
 \iff & \left(\begin{pmatrix} 0 & -\frac{d}{c}b \\ \frac{a}{b}c & 0 \end{pmatrix} + i\sqrt{ad} \cdot I \right) \cdot v_2 = 0 \\
 \iff & \begin{pmatrix} i\sqrt{ad} & -\frac{d}{c}b \\ \frac{a}{b}c & i\sqrt{ad} \end{pmatrix} \cdot \begin{pmatrix} v_{21} \\ v_{22} \end{pmatrix} = 0
 \end{aligned}$$

$$\iff \begin{pmatrix} i\sqrt{ad} & -\frac{d}{c}b \\ 0 & -i\sqrt{ad} + i\sqrt{ad} \end{pmatrix} \cdot \begin{pmatrix} v_{21} \\ v_{22} \end{pmatrix} = 0$$

$$\iff v_2 = \begin{pmatrix} \frac{\frac{d}{c}b}{i\sqrt{ad}} \\ 1 \end{pmatrix},$$

wobei $v_{22} = 1$ gewählt wurde.

– Die Lösung ist dann

$$\begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = c'_1 \cdot \begin{pmatrix} \frac{\frac{d}{c}b}{-i\sqrt{ad}} \\ 1 \end{pmatrix} \cdot e^{i\sqrt{ad} \cdot t} + c'_2 \begin{pmatrix} \frac{\frac{d}{c}b}{i\sqrt{ad}} \\ 1 \end{pmatrix} \cdot e^{-i\sqrt{ad} \cdot t}.$$

In folgender Grafik wurde für Beispielwerte $a = 2$, $b = 1.2$, $c = 1$, $d = 0.9$ und Anfangswert $A_0 = (1, 1)^T$ die Lösung von $N(t)$ sowie $P(t)$ gegen die Zeit aufgetragen.

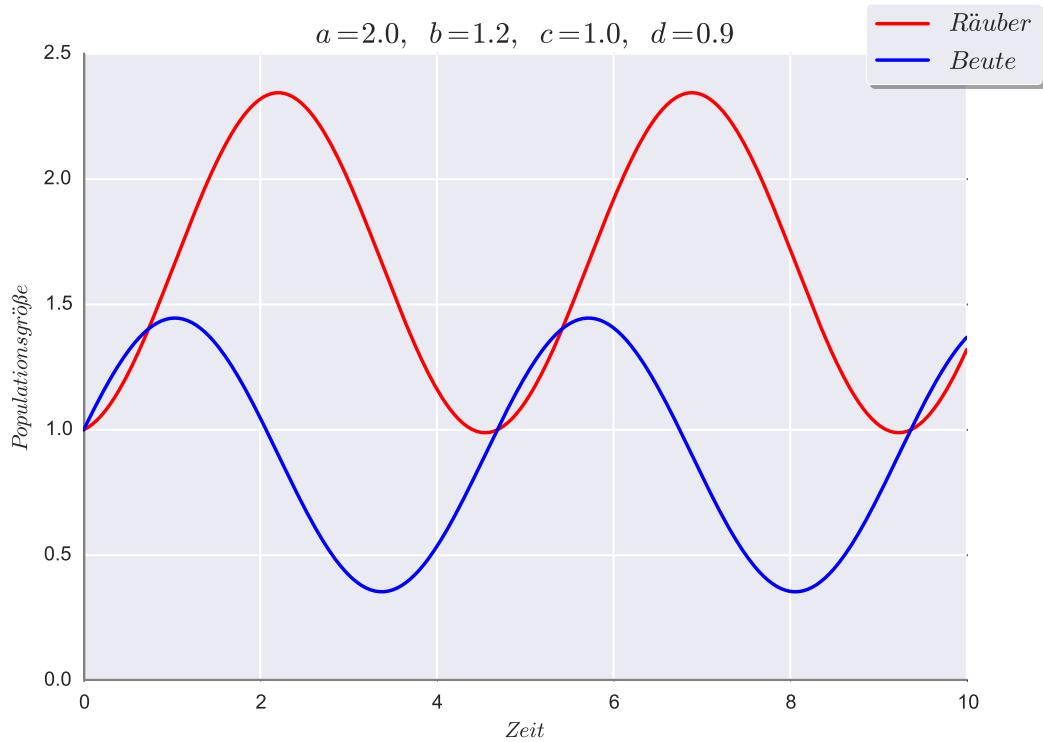


Abbildung 5: Linearisierte Lösung um G_2 , Populationsgrößen gegen die Zeit aufgetragen

In Abbildung 5 sind Oszillationen zu beobachten. Hierbei ist noch unklar, ob sie periodisch sind. Dies soll in der folgenden Eigenwertanalyse beantwortet werden.

5.3.5 Eigenwertanalyse

Unter Verwendung der Lösung aus oberem Kapitel und der berechneten Eigenwerte wird das Verhalten des linearisierten Lotka-Volterra Modells für $t \rightarrow \infty$ bestimmt.

Phasenportrait von G_1 : Die Lösung ist

$$\begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = c_1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} e^{ta} + c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot e^{-td}.$$

Betrachtung der Grenzwerte ergibt

$$\lim_{t \rightarrow \infty} \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = \underbrace{c_1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \underbrace{e^{ta}}_{\substack{\rightarrow \infty \\ \rightarrow 0}}}_{\rightarrow \begin{pmatrix} \infty \\ 0 \end{pmatrix}} + \underbrace{c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot \underbrace{e^{-td}}_{\substack{\rightarrow 0 \\ \rightarrow 0}}}_{\rightarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix}} = \begin{pmatrix} \infty \\ 0 \end{pmatrix},$$

$$\lim_{t \rightarrow -\infty} \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = \underbrace{c_1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \underbrace{e^{ta}}_{\substack{\rightarrow 0 \\ \rightarrow \infty}}}_{\rightarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix}} + \underbrace{c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot \underbrace{e^{-td}}_{\substack{\rightarrow -\infty \\ \rightarrow 0}}}_{\rightarrow \begin{pmatrix} 0 \\ -\infty \end{pmatrix}} = - \begin{pmatrix} 0 \\ \infty \end{pmatrix},$$

falls $c_1 \neq 0 \neq c_2$. Einen solchen Gleichgewichtspunkt nennt man **Sattelpunkt**.

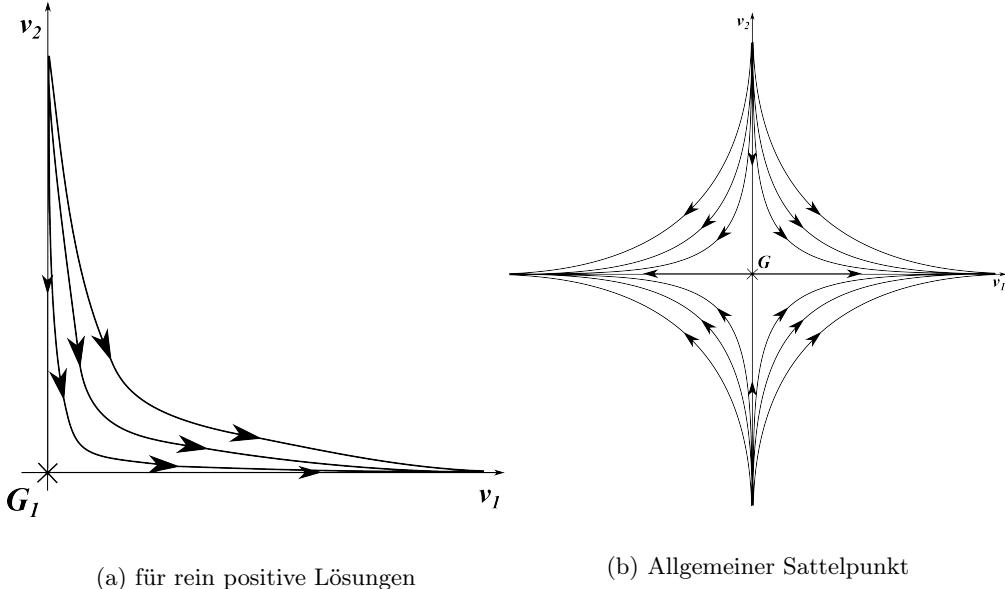


Abbildung 6: Grafische Darstellung eines Sattelpunkts

Die Abbildung 7 zeigt den Kurvenverlauf der Lösung des linearisierten Lotka-Volterra Modells um G_1 mit 3 unterschiedlichen Anfangswerten und Beispielparameter $a = 2, b = 1.2, c = 1, d = 0.9$ in der Phasenebene.

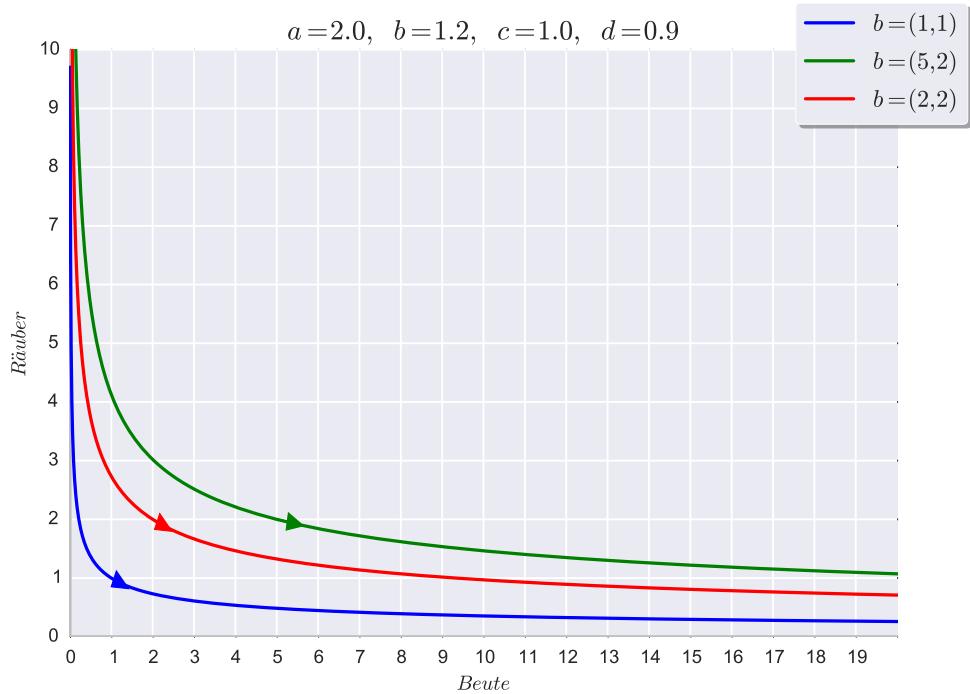


Abbildung 7: Kurvenverlauf für Lösungen des linearisierten Lotka-Volterra Modells nahe G_1 in der Phasenebene

Für die modellierten Populationen bedeutet dies, dass, falls beide Populationen nahe dem Aussterbepunkt sind, die Räuberpopulation gegen 0 strebt und die Beutepopulation exponentiell gegen unendlich wächst.

Da dies die Grundannahmen an das Modell widerspiegelt, konnte hierdurch kein Zugewinn erlangt werden. Viel interessanter dahingegen ist der Gleichgewichtspunkt G_2 .

Phasenporträt von G_2 : Die Lösung ist

$$\begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = c'_1 \cdot \begin{pmatrix} \frac{d}{c}b \\ -i\sqrt{ad} \end{pmatrix} \cdot e^{i\sqrt{ad} \cdot t} + c'_2 \begin{pmatrix} \frac{d}{c}b \\ i\sqrt{ad} \end{pmatrix} \cdot e^{-i\sqrt{ad} \cdot t}.$$

Die Grenzwertbetrachtung

$$\lim_{t \rightarrow \infty} \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = \lim_{t \rightarrow \infty} c'_1 \cdot \begin{pmatrix} \frac{d}{c}b \\ -i\sqrt{ad} \end{pmatrix} e^{i\sqrt{ad} \cdot t} + c'_2 \begin{pmatrix} \frac{d}{c}b \\ i\sqrt{ad} \end{pmatrix} \cdot e^{-i\sqrt{ad} \cdot t}$$

ergibt keinen spezifischen Wert, noch eine Tendenz in Richtung $\pm\infty$.

Die Eigenwerte sind imaginär und die entsprechende Exponentialfunktion ist periodisch, in diesem Fall $\frac{2\pi}{\sqrt{ad}}$ periodisch. Dies bedeutet, dass auch die Lösung $\frac{2\pi}{\sqrt{ad}}$ -periodisch sein muss. Auf der Phasenebene ist dieses Verhalten repräsentiert durch kreisförmige Bewegungen um den Gleichgewichtspunkt. Einen solchen Punkt nennt man **Wirbelpunkt**.

Abbildung 23 zeigt den Verlauf der Lösung des linearisierten Lotka-Volterra

Modells um G_2 mit 3 unterschiedlichen Anfangswerten und Beispielparameter $a = 2$, $b = 1.2$, $c = 1$, $d = 0.9$ in der Phasenebene.

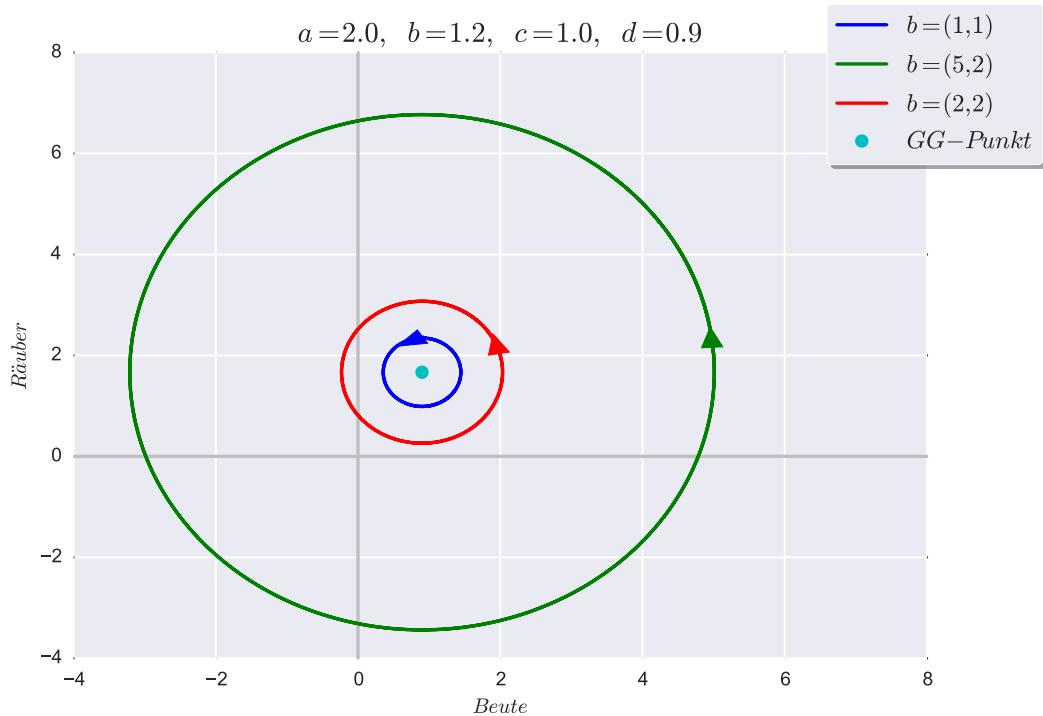


Abbildung 8: Kurvenverlauf der linearisierten Lösung um G_2 in der Phasenebene

Aus Abbildung 23 können zwei wichtige Aspekte heraus gelesen werden.

- Sind die Anfangswerte zu weit vom Gleichgewichtspunkt G_2 entfernt, kann die Lösung auch negative Werte annehmen.

Grund:

Die Linearisierung in G_2 kennt G_1 als Gleichgewichtspunkt nicht mehr. Für Anfangswerte $b = (2, 2)^T$ und $b = (5, 2)^T$, wenn Populationsgrößen (insbesondere der Räuberpopulation) zu nah an die 0 kommen, übernimmt im Verlauf des echten Lotka-Volterra Modells die in G_1 linearisierte Lotka-Volterra-Lösung und geht bei genügend großer Beutepopulation wieder in den Verlauf der in G_2 linearisierten Lösung über.

- Der Kurvenverlauf der linearisierten Lösung in der Phasenebene ist ein geschlossener Kreis.

Interpretation:

Die Populationen befinden sich in periodischer Fluktuation ohne zu einem Gleichgewichtspunkt hin zu tendieren.

Aber genau in der zweiten Beobachtung liegt der Schwachpunkt von Wirbelpunkten,

welcher sie nahezu nutzlos macht: Bereits kleine Abweichungen von dem idealen Modellverlauf können die Eigenwerte längs der reellen Achse verschieben. Dies kann stark unterschiedliche Auswirkungen haben. In folgender Abschätzung wird eine Fallunterscheidung für mögliches Verhalten für $t \rightarrow \infty$ durchgeführt.

Gegeben sei

$$e^{\Re(\lambda)t + \Im(\lambda)t} = e^{\Re(\lambda)t} \cdot e^{\Im(\lambda)t}.$$

Dabei gilt

$$\exists N \in \mathbb{R}, \text{ sodass } e^{\Im(\lambda)t} = e^{\Im(\lambda)(t+N)}$$

und

$$\begin{aligned} \forall i = 1, 2 \exists t_{p_i}, t_{m_i} \in [t, t+N] : \Re(e^{\Im(\lambda)t_{p_i}}) > 0 \text{ und } \Re(e^{\Im(\lambda)t_{m_i}}) < 0, \\ \iff \exists t_p, t_m \in [t, t+N] : \Re(x(t_p)) > 0 \text{ und } \Re(x(t_m)) < 0. \end{aligned}$$

Bei der Betrachtung des Grenzwertes ergibt sich dann

- $\Re(\lambda) > 0$

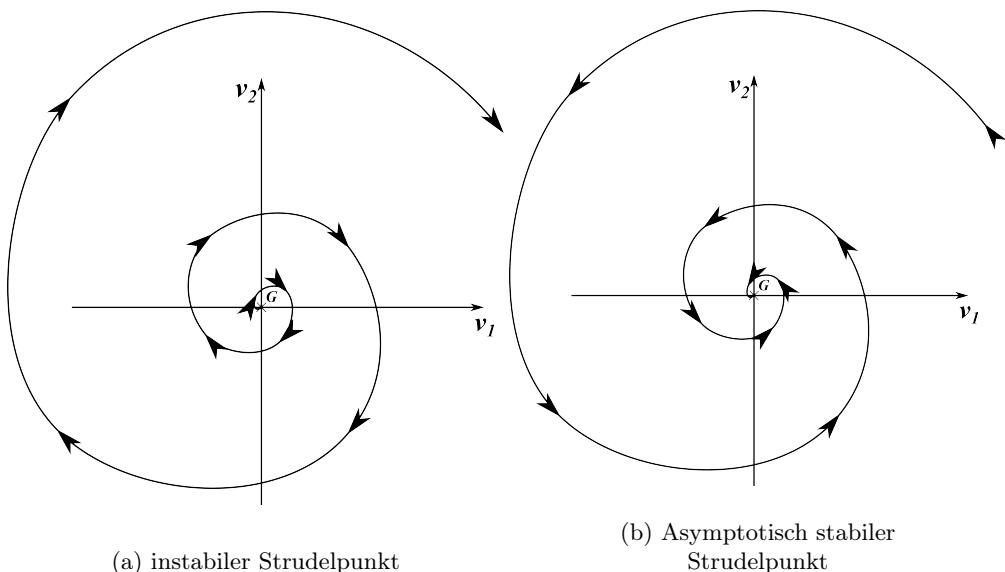
$$\lim_{t \rightarrow \infty} \underbrace{e^{\Re(\lambda)t}}_{\rightarrow \infty} \cdot \underbrace{\Re(e^{\Im(\lambda)t})}_{\in [-1,1]} = \infty,$$

- $\Re(\lambda) < 0$

$$\lim_{t \rightarrow \infty} \underbrace{e^{\Re(\lambda)t}}_{\rightarrow 0} \cdot \underbrace{\Re(e^{\Im(\lambda)t})}_{\in [-1,1]} = 0.$$

Diese Form eines Kurvenverlaufs in der Phasenebene nennt man dann einen **Strudelpunkt**.

Abbildung 9: Grafische Darstellung eines Strudelpunktes



5.3.6 Stabil, asymptotisch stabil, instabil

Alle oben angeführten Formen von Lösungskurven linearer Differenzialgleichungssysteme in der Phasenebene kann man in 3 allgemeine Kategorien einteilen.

Definition 5.3.6.1 (Stabilitätsbegriffe von Gleichgewichtspunkten).

Sei \tilde{x} ein Gleichgewichtspunkt des autonomen gewöhnlichen Differenzialgleichungssystems erster Ordnung und definiert wie in Definition 5.3.1.1. Man kann für das Verhalten der Lösung in der Nähe von $\tilde{x}(t)$ verschiedene Eigenschaften festhalten. Hierbei sei $x_0(t)$ eine Lösung des Differenzialgleichungssystems auf dem Intervall $I \subset \mathbb{R}$ in benachbarter Umgebung von $\tilde{x}(t)$.

Man nennt $\tilde{x}(t) \in I$

- **stabil** auf I , falls

$$\forall t \in I \forall \varepsilon > 0 \exists \delta > 0, \text{ sodass: } \forall x_0(t_0) : \|x_0(t_0) - \tilde{x}(t_0)\| < \delta \Rightarrow \|x_0(t) - \tilde{x}(t)\| < \varepsilon.$$

In Worten: Alle Lösungen des Differenzialgleichungssystems, deren Anfangswerte im δ -Umkreis von $\tilde{x}(t)$ liegen, bleiben in einem ε -Umkreis von $\tilde{x}(t)$.

- **attraktiv** auf I , falls $\exists \eta > 0$ mit: Für jede Differenzialgleichung $x_0(t)$ mit

$$\|x_0(t) - \tilde{x}\| < \eta \quad \forall t > t_0$$

existiert eine Lösung, für die gilt

$$\lim_{t \rightarrow \infty} x_0(t) = \tilde{x}(t).$$

- **asymptotisch stabil**, falls $\tilde{x}(t)$ stabil und attraktiv ist.
- **instabil**, falls $\tilde{x}(t)$ nicht stabil ist.

Folgerung 5.3.6.2.

Der **Sattelpunkt** ist also immer instabil.

Der **Wirbelpunkt** ist stabil, aber nicht asymptotisch stabil.

Der **Strudelpunkt** kann instabil oder asymptotisch stabil sein.

- Für $\Re(\lambda_{1,2}) > 0$ ist er instabil,
- für $\Re(\lambda_{1,2}) < 0$ ist er asymptotisch stabil.

Für eine ausführliche Einführung in die qualitative Analyse von Differenzialgleichungen und Übersicht der Phasenportraits für unterschiedliche Eigenschaften der Eigenwerte siehe [Sto08] und [Pan10], Chapter 4, S. 51 ff.

Nun haben wir das Verhalten der linearisierten Lösung des Lotka-Volterra Modells betrachtet. Es stellt sich nun die Frage, inwiefern diese Lösung tatsächlich auf das nichtlineare System übertragbar ist. Hierzu hilft das "Prinzip der linearisierten Stabilität".

Satz 5.3.6.3 (Prinzip der linearisierten Stabilität).

Sei A die Linearisierung des Systems F von nichtlinearen Differenzialgleichungen in einem Gleichgewichtspunkt \tilde{x} des Systems F .

Sei $\sigma(A) := \{\lambda \in \mathbb{C} : (A - \lambda \cdot I) = 0\}$ die Menge aller Eigenwerte von A . Dann gilt für alle $\lambda \in \sigma(A(\tilde{x}))$

- $\Re(\lambda) < 0 \quad \forall \lambda \in \sigma(A) \implies \tilde{x} \text{ ist asymptotisch stabil,}$
- $\exists \lambda \in \sigma(A) \text{ s.d. } \Re(\lambda) > 0 \implies \tilde{x} \text{ ist instabil.}$

Beweis. Siehe [Wer08] S. 38 ff Satz 3.7. □

Es kann also geschlossen werden, dass G_1 auch für das echte Lotka-Volterra Modell ein Sattelpunkt ist. Für G_2 ist bereits die Linearisierung mit ihrer minimalen Abweichung in direkter Nähe zum Gleichgewichtspunkt problematisch. Die Aussage aus der Linearisierung kann deswegen nicht auf das nichtlineare Problem übertragen werden.

Zu Frage 3: Durch die grafische, wie auch mathematische Analyse wurden mehrere Hinweise auf ein mögliches periodisches Verhalten der Lösungen um den Gleichgewichtspunkt G_2 ermittelt. Die folgende Analyse hat das Ziel eine möglichst exakte Skizze der Lösungen zu zeichnen und die Lösung auf Periodizität zu untersuchen.

5.3.7 Skizzieren der Lösung des nichtlinearen Lotka-Volterra Modells

Da die Ableitung von $N(t)$ sowie $P(t)$ durch ihre Differenzialgleichungen gegeben sind, können wir den Richtungsvektor

$$\frac{\frac{dP}{dt}}{\frac{dN}{dt}} = \frac{P(t)(cN(t) - d)}{N(t)(a - bP(t))} := \frac{dP}{dN}$$

für die Phasenebene in jedem Punkt (N_i, P_i) berechnen.

Dies ist nicht nur ein Richtungsvektor in einzelnen Punkten sondern die Ableitung eines Systems von autonomen Differenzialgleichungen reduziert auf die Phasenebene. Durch Integration können wir den Verlauf der Lösungskurve in der Phasenebene berechnen. Es gilt also

$$\begin{aligned} \frac{dP}{dN} &= \frac{P(t)(cN(t) - d)}{N(t)(a - bP(t))} & | \cdot \frac{1}{P(t)} \cdot (a - bP(t)), \\ \Leftrightarrow \frac{a - bP(t)}{P(t)} \frac{dP}{dN} &= \frac{cN(t) - d}{N(t)} & | \int \dots dN \\ \Leftrightarrow \int \frac{a - bP(t)}{P(t)} dP &= \int \frac{cN(t) - d}{N(t)} dN \\ \Leftrightarrow \int a \cdot \frac{1}{P(t)} - b dP &= \int c - d \frac{1}{N(t)} dP \\ \Leftrightarrow a \cdot \ln(P(t)) - bP(t) &= cN(t) - d \cdot \ln(N(t)) + c_1, \end{aligned}$$

wobei c_1 durch den Anfangswert des Lotka-Volterra Modells festgelegt wird.

Folgerung 5.3.7.1.

Sei das Lotka-Volterra Modell mit Anfangswert gegeben und die Konstante c_1 in der Lösung durch den Anfangswert bestimmt, dann kann der Lösungsverlauf in der Phasenebene dargestellt werden durch die Gleichung

$$a \cdot \ln(P(t)) - bP(t) - cN(t) + d \cdot \ln(N(t)) = c_1.$$

Diese Gleichung ist durch die dreidimensionale Funktion

$$E(N, P) = a \cdot \ln(P(t)) - bP(t) - cN(t) + d \cdot \ln(N(t)) + c_2$$

beschrieben, wobei $c_2 = -c_1$. In Abbildung 10 ist ein Graph dieser Funktion mit den Parametern $a = 5, b = 1, c = 0.3, d = 4$ und Anfangswert $A_0 = (3, 3)^T$, welches zu $c_1 = -6$ führt, zu sehen.

$$a = 5.0, \quad b = 1.0, \quad c = 0.3, \quad d = 4.0$$

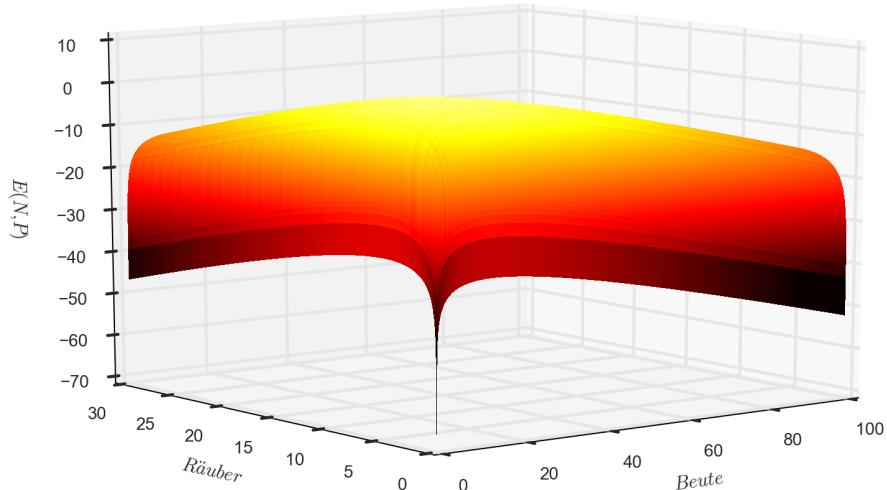


Abbildung 10: Dreidimensionaler Graph der Funktion $E(N, P)$

Für eine eindeutige Vorstellung dieser dreidimensionalen Funktion folgt eine Reihe von 4 Bildern, welche dieselbe Funktion aus 4 verschiedenen Blickwinkeln zeigt. Der Graph wurde im Uhrzeigersinn jeweils um 90 Grad gedreht.

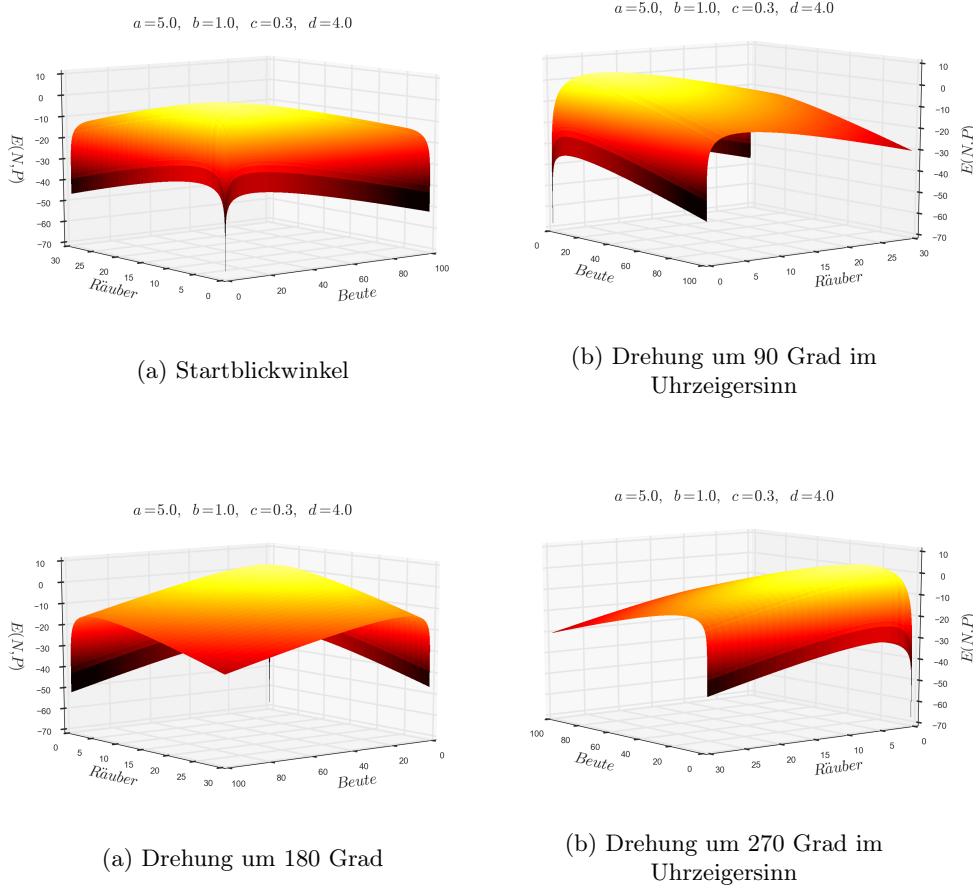


Abbildung 12: 360 Grad Betrachtung der dreidimensionalen Funktion aus 4 Blinkwinkeln im Abstand von 90 Grad

Die Konstante c_2 wird durch den Anfangswert bestimmt. Sei $b = (b_1, b_2)$ der Anfangswert für ein Lotka-Volterra Modell, dann gilt

$$\begin{aligned} E(b_1, b_2) = 0 &\Leftrightarrow a \cdot \ln(b_2) - bb_2 - cb_1 + d \cdot \ln(b_2) + c_2 = 0 \\ &\Leftrightarrow c_2 = -a \cdot \ln(b_2) + b \cdot b_2 + cb_1 - d \cdot \ln(b_2). \end{aligned}$$

Der Anfangswert b verschiebt somit die Funktion $E(N, P)$ entlang der z-Achse. Der Schnittpunkt der $E(N, P)$ -Funktion mit der N-P-Ebene ergibt dann den gesuchten Kurvenverlauf für gegebenen Anfangswert b . Abbildung 13 zeigt die Trajektorien für mehrere Werte von c_1 mit Parametern $a = 5, b = 1, c = 0.3, d = 4$. Für den Anfangswert $A_0 = (3, 3)^T$ ergibt sich $c_2 = -5, 9875 \approx -6$, welcher in Abbildung 13 als die äußere Trajektorielinie dargestellt ist. In folgender Grafik wurden die Trajektorielinien und Füllungen mithilfe der Funktion `contourf` in python (siehe [Pyta]) erstellt.

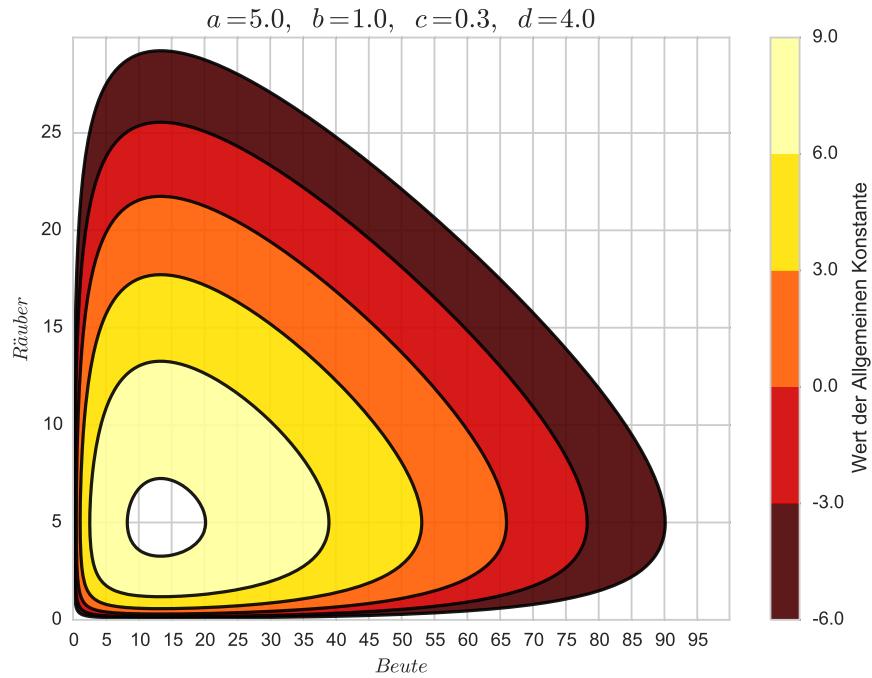


Abbildung 13: Trajektorien des Lotka-Volterra-Systems für verschiedene Werte von c_1 in der Phasenebene

5.3.8 Schlussfolgerung

Es folgt eine Zusammenfassung der Ergebnisse der qualitativen Analyse der Gleichgewichtspunkte G_1 und G_2 .

- G1:

Es konnte gezeigt werden, dass der Gleichgewichtspunkt G_1 ein Sattelpunkt ist und sein Verhalten für $t \rightarrow \infty$ bestimmt werden.

- G2:

Für G_2 hat die Linearisierung nicht ausgereicht um eine eindeutige Aussage über das Verhalten der Populationsgrößen zu machen. Um dies zu erreichen, musste eine weitere Methode zur Analyse genutzt werden. Durch das Bilden des Quotienten der 2 Differenzialgleichungen $\frac{dP}{dt}(t)$ und $\frac{dN}{dt}(t)$ wurde die Änderungsrate in der Phasenebene, sowie, durch Integration, eine dreidimensionale Funktion berechnet, deren Schnittebene mit der N-P-Ebene das Phasenportrait für G_2 ergab. Somit konnten auch alle 3 Fragen für G_2 beantwortet und ein Phasenportrait für das Verhalten des Lotka-Volterra Modells angefertigt werden.

5.4 Schwachpunkte des Lotka-Volterra Modells

Das Lotka Volterra Modell hat von Anfang an zwei Probleme:

1. **Die Modellannahme von exponentiellem Wachstum der Beute-Population ist unrealistisch.**

Natürliche Populationen haben immer ein beschränktes Wachstum aufgrund von limitierten Habitateigenschaften, wie Nahrung, Wasser und einem passenden Unterschlupf (siehe [GY]). Die Annahme des exponentiellen Wachstums der Beute-population ist daher ungeeignet zur Modellierung natürlicher Populationen.

Dazu im Zusammenhang stehend ist, dass die **Futter-Aufnahmekapazität der Räuber im Lotka-Volterra Modell nicht beschränkt ist**. Das Modell basiert auf der Annahme, dass jedes gestorbene Beuteindividuum vollständig von den Räubern zum Wachstum der eigenen Population genutzt werden kann, selbst dann, wenn pro Räuber-Individuum ein vielfaches seiner natürlichen Fress-Kapazität aufgenommen werden müsste.

2. **Die Populationsgrößen werden als kontinuierliche Zahlen modelliert.**

Laut Modell existieren Populationsgrößen von z.B. 1,2 Räuberindividuen. Dies hat zwei schwerwiegende Nachteile

- **Die modellierten Populationen bleiben nicht auf einer Trajektorie.**

Im Kapitel über die Lösung des linearisierten Lotka-Volterra Modells wurde auf die Instabilität von Wirbelpunkten eingegangen. Im Fall des echten Lotka-Volterra Modells ist diese Kritik genauso zutreffend. Da das zu modellierende Problem aber diskret ist und das Modell kontinuierlich, ergeben sich automatisch kleinere bis größere Abweichungen von den errechneten Trajektorien. Die Populationsdynamik springt also auf eine andere, nahe gelegene Trajektorie. Wenn dies in der Nähe einer Achse passiert, kann die neue Trajektorie weit von der ursprünglichen Trajektorie entfernte Populationsgrößen haben. Vergleiche die Trajektorien für $c_2 = -6$ und $c'_2 = 6$ in Abbildung 13. Diese sind nahe der Achsen dicht genug für einen Sprung, sind aber im sonstigen Verlauf stark unterschiedlich. Dies macht eine präzise Modellierung unmöglich und kann zu Fehlern in der Interpretation von Datenpunkten führen.

- **Die Modell-Populationen nehmen Werte von bis zu 0,001 Individuen an und erholen sich dann wieder.**

Laut Modell bewegen sich die Populationsgrößen periodisch um Gleichgewichtspunkt G_2 mit sehr starker Neigung zu den Achsen. Insbesondere große Populationen von Räubern und Beute werden zu bestimmten Zeitpunkten besonders klein (< 1). Im Anschluss erholen sich die Populationen wieder. Das ist bei natürlichen Populationen nicht möglich, weshalb das Lotka Volterra-Modell zur Vorhersage von Aussterbewahrscheinlichkeiten ungeeignet ist.

Ein weiterer Schwachpunkt ist, dass nur kleine Populationen keinem starken Zug in Richtung der Achsen ausgesetzt sind und nie den Zahlenbereich unterhalb der 2 erreichen. Laut Modell sind sie absolut sicher vor Aussterbegefahren, wohingegen große Populationen immer eine Populationsgröße unterhalb der 2 erreichen. Aus biologischen Studien (siehe [Thi12], Kapitel 2-3, einen guten Überblick bietet die Zusammenfassung auf S. 49) ist aber bekannt, dass gerade kleine Populationsgrößen einer besonders starken Extinktionsgefahr ausgesetzt sind und große Populationen besonders stabil und langlebig sind.

Der Grund liegt hierbei in einer stärkeren Anfälligkeit gegenüber demografischem Rauschen, Umweltfaktoren und einem kleineren Genpool von kleinen Populationen. Ersteres sind stochastische Prozesse innerhalb von Populationen, welche Geburtenraten, Sterberaten und Geschlechterverteilung beeinflussen, welche eine wichtige Rolle in der Fertilität der nächsten Generation spielen (so hat eine Population von 5 Individuen mit nur einem Weibchen schlechte Überlebenschancen als eine Population mit 5 Individuen und 2-3 Weibchen). Umweltfaktoren sind stochastische Schwankungen in der Umwelt und können eine Population überraschend und stark reduzieren. Die Größe und Qualität des Genpools ist der evolutionstreibende Faktor der Anpassungsfähigkeit einer Population an eine sich ständig ändernde Umwelt. Je größer der Genpool, desto besser die Anpassungsfähigkeit der Population.⁸

All diese Faktoren sind im Lotka-Volterra Modell nicht berücksichtigt, aber starke Prädiktoren für die Extinktionsgefahr einer Population (siehe [Thi12], Kapitel 2-3, einen guten Überblick bietet die Zusammenfassung auf S. 49).

Als Schlussfolgerung kann formuliert werden, dass das Lotka-Voterra Modell ein Start in Richtung der Modellierung von Räuber-Beute-Beziehungen ist, aber es noch viel Raum für Verbesserungen gibt.

Dennoch gibt es Situationen, in denen das Lotka-Voterra Modell eine gute Approximation liefert.

Es kann eine gute Approximation für stabile Räuber-Beute-Beziehungen sein, wenn beide Populationen viele Individuen haben und der Gleichgewichtspunkt des Modells weit vom Nullpunkt entfernt liegt. Findet man also Räuber-Beute-Beziehungen, die diese Voraussetzungen erfüllen, ist das Lotka-Voterra Modell eine gute und einfache Möglichkeit, diese Populationen zu modellieren.

Verbesserungsmöglichkeiten: Eine offensichtliche Verbesserungsmöglichkeit besteht darin, die Beutepopulation mit logistischem Wachstum zu versehen. Dies würde nach wie vor bei kleinen Beutepopulationen zu gewünschtem starkem Wachstum füh-

⁸Dieser Abschnitt ist meine Zusammenfassung zum Thema aus [Thi12] Kapitel 2-3 Aussterben von Arten, S. 32-53

ren, aber bei größer werdender Beute-Anzahl abnehmen und stagnieren (siehe [Log]). Ein weiterer bereits angesprochener Schwachpunkt im Modell ist die Aufnahmekapazität von Fressen der Räuber-Individuen. Diese in das Modell zu integrieren könnte die Sterberate der Beuteindividuen bei großen Beutepopulationen und kleinen Räuberpopulationen reduzieren und somit zu einem weniger starken Abfall der Beutepopulation und weniger starken Wachstum der Räuberpopulation führen. Dies wären Startpunkte, um das Modell dahingehend zu verbessern, die Anziehungs- kraft der Achsen auf die Populationsgrößen zu verringern.

6 Modellerweiterung – Logistisches Wachstum der Beutepopulation

Wie in Kapitel 5.4 hervorgehoben, sollten als ersten Schritt in Richtung eines akkurateeren Modells die Modellannahmen des Lotka-Volterra Modells verbessert werden. In diesem Kapitel soll eine der vielen Möglichkeiten hierfür betrachtet werden: **Das Hinzufügen einer oberen Schranke für die Beutepopulation.** Das Ziel ist, die Anpassung des Modells an die Beutepopulation zu optimieren.

Ein gutes Modell für eine Population, welche nicht bejagt wird, ist das logistische Modell (siehe [Log], Application). Es integriert eine obere Schranke, welche die Wachstumsschranke in natürlichen Populationen modelliert, aber bei kleinen Populationen nahezu exponentielles Wachstum zulässt. In natürlichen Populationen entsteht eine solche Schranke durch begrenzt vorhandene Nahrung und Platz (siehe [GY]). Ist die Population allerdings klein, herrscht kaum Konkurrenz um Nahrung und Platz und die Population wächst quasi-exponentiell.

In diesem Kapitel wird das logistische Modell in die Differenzialgleichung der Beute-Gleichung des Lotka-Volterra Modells integriert und, wann immer möglich, mit denselben Methoden analysiert.

6.1 Modellierung

6.1.1 Modellannahmen

Sei wieder P die Anzahl der Individuen einer Population von Räubern und N die Anzahl der Individuen von Beutetieren der Räuberpopulation P . Wir übernehmen zu großen Teilen die Parameter des Lotka-Volterra Modells. Sollten die Parameter abweichen, wird im Folgenden darauf hingewiesen. Es wird mit den Modellannahmen begonnen:

1. Das Wachstum der Beutepopulation hat in Abwesenheit des Räubers logistisches Verhalten.
2. In Anwesenheit der Räuberpopulation wird die Beutepopulation durch die Räuber proportional zur Interaktionshäufigkeit von Räuber- und Beutepopulation reduziert.
3. Die Räuberpopulation stirbt in Abwesenheit der Beutepopulation mit exponentieller Sterberate aus.
4. In Anwesenheit der Beutepopulation wächst die Räuberpopulation proportional zur Interaktionshäufigkeit von Räuber- und Beutepopulation.

Hieraus ergeben sich die folgenden vier Gleichungen

1. Das logistische Modell für eine nicht bejagte Population hat die Form

$$\frac{dN}{dt}(t) = a \cdot N(t) - \frac{aN(t)^2}{K} = aN(t) \left(1 - \frac{N(t)}{K}\right), \quad t \geq 0,$$

wobei $a > 0$ der Wachstumsfaktor und $K > 0$ die Kapazität der Beutepopulation ist. Die Kapazität einer Population ist die maximale Zahl von Individuen einer Population, die in dieser Umwelt überleben können.

Der Wachstumsfaktor unterscheidet sich von der Wachstumsrate a aus dem Lotka-Volterra Modell. Er enthält zusätzlich zur Wachstumsrate aus dem Lotka-Volterra Modell auch die natürliche Sterberate der Population und hat somit Einfluss auf die Gesamtsterberate. Der Parameter a ist in den zwei Modellen also unterschiedlich. Auf die Gesamtsterberate wird in 2 eingegangen. Für das logistische Wachstumsverhalten der Beutepopulation gilt:

Für $N(t) \approx K \implies \frac{dN}{dt}(t) \approx 0$.

Für $N(t) \approx 0 \implies \frac{dN}{dt}(t) \approx aN(t)$,

und somit exponentielles Wachstum für kleine Beutepopulationen.

2. Der direkte Einfluss der Räuberpopulation auf die Beutepopulation bleibt gleich mit

$$\frac{dN}{dt}(t) = -b \cdot N(t)P(t).$$

Möchte man die Gesamtsterberate der Beutepopulation, muss zu oberer Gleichung noch die natürliche Sterberate der Beute hinzugefügt werden und man erhält

$$\frac{dN}{dt}(t) = -\frac{aN(t)^2}{K} - bN(t)P(t) = -N(t) \left(\frac{aN(t)}{K} + bP(t) \right), \quad t \geq 0.$$

3. Die Sterberate der Räuberpopulation hat sich nicht verändert. Es gilt

$$\frac{dP}{dt}(t) = -d \cdot P(t), \quad t \geq 0.$$

4. Die Wachstumsrate der Räuberpopulation hat sich ebenfalls nicht verändert. Es gilt

$$\frac{dP}{dt}(t) = c \cdot P(t)N(t) - dP(t), \quad t \geq 0.$$

Für die bejagte Beutepopulation gilt also mit Wachstumsfaktor $a > 0$ und Sterberate aus der Bejagung $b > 0$ die Gleichung

$$\begin{aligned} \frac{dN}{dt}(t) &= a \cdot N(t) \left(1 - \frac{N(t)}{K} \right) - bN(t)P(t), \\ &= N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - bP(t) \right), \quad t \geq 0. \end{aligned}$$

6.1.2 Das Differenzialgleichungssystem

Folgerung 6.1.2.1.

Das Differenzialgleichungssystem für die logistische Räuber-Beute Beziehung hat die Form

$$\frac{dN}{dt}(t) = N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - bP(t) \right),$$

$$\frac{dP}{dt}(t) = P(t)(cN(t) - d).$$

Für die Vektorschreibweise setze

$$f_N(N(t), P(t)) = N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - bP(t) \right),$$

$$f_P(N(t), P(t)) = P(t)(cN(t) - d),$$

sodass für $x = (N(t), P(t))^T$ die Vektorschreibweise

$$\frac{dx}{dt}(t) = \begin{pmatrix} f_N(x(t)) \\ f_P(x(t)) \end{pmatrix} \quad (1 \log x)$$

gilt.

Bemerkung 6.1.2.2.

Das Differenzialgleichungssystem für die logistische Räuber-Beute Beziehung ist, wie auch die Lotka-Volterra Gleichung, ein System von autonomen gewöhnlichen Differenzialgleichungen.

6.2 Grafische Analyse des logistischen Räuber-Beute Modells

6.2.1 Gleichgewichtslösungen

Theorem 6.2.1.1.

Die Gleichgewichtslösungen zum logistischen Räuber-Beute Modell aus Folgerung 6.1.2.1 sind

$$G_1(t) = \begin{cases} N(t) = 0 \\ P(t) = 0 \end{cases}, \quad G_2(t) = \begin{cases} N(t) = \frac{d}{c} \\ P(t) = \frac{a}{b} \cdot \left(1 - \frac{\frac{d}{c}}{K}\right) \end{cases},$$

$$G_3(t) = \begin{cases} N(t) = K \\ P(t) = 0 \end{cases}.$$

Dabei ist zu beachten, dass die Geradengleichungen der einzelnen Gleichgewichtslösungen für $G_2(t)$ durch

$$G_{2_{allg}}(t) = \begin{cases} N(t) = \frac{d}{c} \\ P(t) = \frac{a}{b} \cdot \left(1 - \frac{N(t)}{K}\right) \end{cases}$$

gegeben sind. Hierbei ist

1. G_1 der Schnittpunkt der Nulllösungen,
2. G_2 der Schnittpunkt der 2 Geraden $N(t) = \frac{d}{c}$ und $P(t) = \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$,
3. G_3 der Schnittpunkt der 2 Geraden $N(t) = 0$ und $P(t) = \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$.

Beweis. • Lösung $G_1(t)$ ist klar.

- Aus $N(t) = 0$ folgt direkt $P(t) = 0$ und wird somit zu $G_1(t)$.

- Für $G_2(t)$ untersuche den Fall $P(t) \neq 0$ und $N(t) \neq 0$. Dann ist

$$0 = N(t) \left(a \left(1 - \frac{N(t)}{K}\right) - bP(t) \right),$$

$$0 = P(t)(cN(t) - d).$$

Somit

$$0 = cN(t) - d \Leftrightarrow d = cN(t) \Leftrightarrow \frac{d}{c} =: N(t),$$

$$0 = a \left(1 - \frac{N(t)}{K}\right) - bP(t) \Leftrightarrow a \left(1 - \frac{N(t)}{K}\right) = bP(t) \Leftrightarrow \frac{a}{b} \left(1 - \frac{N(t)}{K}\right) =: P(t).$$

Setze zum Schluss $N(t) = \frac{d}{c}$ in der Gleichgewichtsgerade von $P(t)$, da der Gleichgewichtspunkt des Systems der Schnittpunkt der beiden Gleichgewichtsgeradengleichungen von $P(t)$ und $N(t)$ ist.

- Für G_3 zeige den Fall $P(t) = 0$ und $N(t) \neq 0$. Es ist

$$\begin{aligned}
 0 &= \frac{dN}{dt}(t) = N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - b \cdot 0 \right), \\
 &= N(t) \left(a \left(1 - \frac{N(t)}{K} \right) \right), \\
 &= 1 - \frac{N(t)}{K}, \quad \text{da } N(t) \neq 0 \neq a, \\
 \iff \frac{N(t)}{K} &= 1, \\
 \iff N(t) &= K.
 \end{aligned}$$

Und somit gilt für G_3 $P(t) = 0$ und $N(t) = K$.

□

Bemerkung 6.2.1.2.

In folgender Grafik sind alle 3 Gleichgewichtslösungen als Schnittpunkte ihrer Geraden grafisch dargestellt. Die Parameter sind $a = 10$, $b = 0.5$, $c = 0.3$, $d = 3.8$ mit Kapazität $K = 20$.

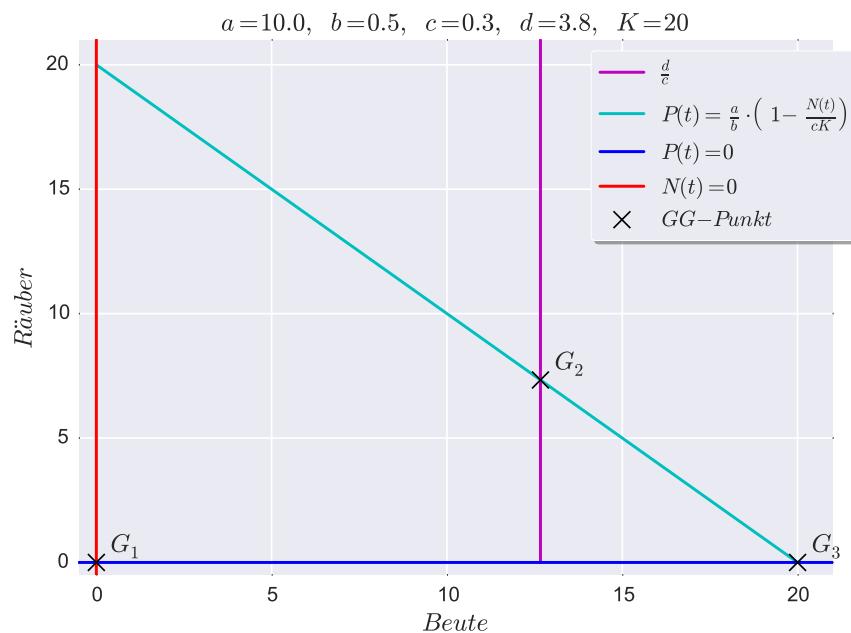


Abbildung 14: Grafik der Gleichgewichtspunkte als Schnittpunkte der jeweiligen Gleichgewichtsgeraden

Punkte $(0, 20)$ und $(\frac{d}{c}, 0)$ sind keine Gleichgewichtslösungen. Siehe Beweis von Theorem 6.2.1.1.

6.2.2 Richtungvektoren von Lösungen mit Anfangswerten außerhalb der Gleichgewichtslösungen

Als nächsten Schritt soll das Verhalten des Differenzialgleichungssystems für Punkte außerhalb der Gleichgewichtspunkte betrachtet werden. Hierfür werden die Differenzialgleichungen wieder einzeln untersucht.

Abschätzung: Da G_3 ebenfalls auf den Geraden $P_1(t) = \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$ und $N(t) = 0$ liegt, ist keine zusätzliche Betrachtung der Gleichgewichtsgeraden von $G_3(t)$ notwendig.

$$1. \frac{dN}{dt}(t) = N(t) \left(a \left(1 - \frac{N(t)}{K}\right) - bP(t) \right)$$

- Für $P(t) > \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$, gilt

$$\frac{dN}{dt}(t) = N(t) \left(a \left(1 - \frac{N(t)}{K}\right) - bP(t) \right)$$

$$< N(t) \underbrace{\left(a \left(1 - \frac{N(t)}{K}\right) - b \cdot \frac{a}{b} \left(1 - \frac{N(t)}{K}\right) \right)}_{=0},$$

$$\text{also } \frac{dN}{dt}(t) < 0.$$

- Für $P(t) < \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$, gilt

$$\frac{dN}{dt}(t) = N(t) \left(a \left(1 - \frac{N(t)}{K}\right) - bP(t) \right)$$

$$> N(t) \underbrace{\left(a \left(1 - \frac{N(t)}{K}\right) - b \cdot \frac{a}{b} \left(1 - \frac{N(t)}{K}\right) \right)}_{=0},$$

$$\text{also } \frac{dN}{dt}(t) > 0.$$

$$2. \frac{dP}{dt}(t) = P(t)(cN(t) - d)$$

- $N(t) > \frac{d}{c}$, gilt

$$\frac{dP}{dt}(t) = P(t)(cN(t) - d) > P(t)(c \cdot \frac{d}{c} - d) = 0, \quad \text{also } \frac{dP}{dt}(t) > 0.$$

- $N(t) < \frac{d}{c}$, gilt

$$\frac{dP}{dt}(t) = P(t)(cN(t) - d) < P(t)(c \cdot \frac{d}{c} - d) = 0, \quad \text{also} \quad \frac{dP}{dt}(t) < 0.$$

Die folgende Abbildung zeigt diese Abschätzungen. Hierbei hat die Länge der Pfeile keine Bedeutung.

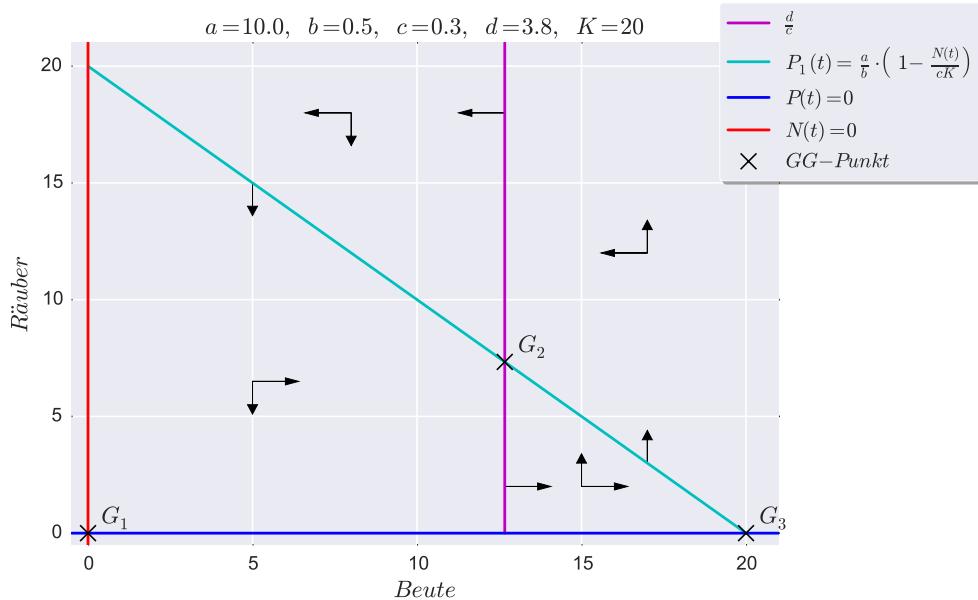


Abbildung 15: Richtungspfeile aufgrund Abschätzung in Paragraph [Abschätzung](#) im Kapitel [6.2.2](#).

Es ist wieder eine kreisförmige Bewegung um G_2 zu vermuten.

Mithilfe der Funktion `quiver()` in python (siehe [\[Pytb\]](#)) konnte das Richtungsfeld für das logistische Räuber-Beute Modell erstellt werden. Zur Betrachtung des Codes mit dem die Grafiken [16](#) und [17](#) erstellt wurden, siehe [D.2](#), Zeilen 202-262 und die Aufrufe in Zeilen 687-694.

Die folgende Grafik ist das Richtungsfeld für gegebene Parameter bei einer Vektor-Skalierung von 0,025.

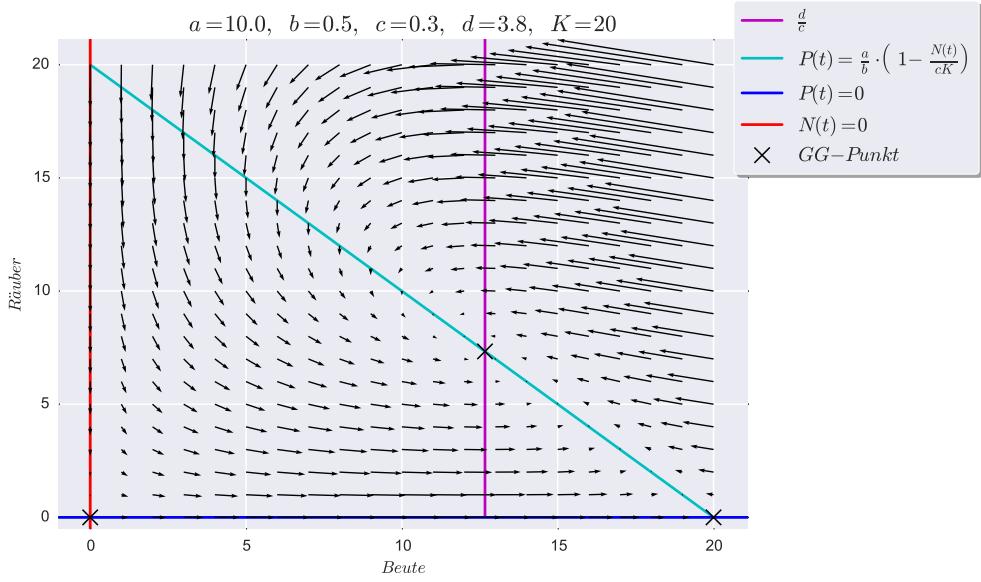


Abbildung 16: 0,025-fache Skalierung der Richtungsvektoren

Das Vektorfeld bestätigt unsere Vermutung der kreiförmigen Bewegung der Populationsgrößen um G_2 .

Auffällig ist, dass entlang der Gleichgewichtsgeraden $P_1(t) = \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$ für $N(t) > \frac{d}{c}$ kaum ein Vektor auszumachen ist.

Dies ist erstmal überraschend, da aus dem Paragraph [Abschätzung](#) im Kapitel 6.2.2 klar wurde, dass $\frac{dP}{dt}(t) > 0$ für $N(t) > \frac{d}{c}$.

Der Grund hierfür liegt in der Skalierung. Diese ist notwendig, da eine 1:1 Darstellung des Vektors im Punkt $N(t) = 20$, $P(t) = 20$ mit dem größten Gradienten

$$\begin{pmatrix} f_N(N(t), P(t)) \\ f_P(N(t), P(t)) \end{pmatrix} = \begin{pmatrix} -200 \\ 44 \end{pmatrix}$$

nicht sinnvoll ist.

Der Gradient ist am geringsten auf der Gleichgewichtsgeraden $P_1(t) = \frac{a}{b} \left(1 - \frac{N(t)}{K}\right)$ für $N(t) > \frac{d}{c}$. Es wird hier als Beispiel der Punkt $N(t) = 15$, $P(t) = 5$ gewählt mit dem Gradienten

$$\begin{pmatrix} f_N(N(t), P(t)) \\ f_P(N(t), P(t)) \end{pmatrix} = \begin{pmatrix} 0 \\ 2, 1 \end{pmatrix}.$$

Eine Skalierung dieser Gradientenvektoren mit 0,025 ergibt

$$\begin{pmatrix} f_N(20, 20) \\ f_P(20, 20) \end{pmatrix} = \begin{pmatrix} -200 \\ 44 \end{pmatrix} \cdot 0,025 = \begin{pmatrix} -5 \\ 1, 1 \end{pmatrix}$$

$$\begin{pmatrix} f_N(15, 5) \\ f_P(15, 5) \end{pmatrix} \cdot 0,025 = \begin{pmatrix} 0 \\ 2,1 \end{pmatrix} \cdot 0,025 = \begin{pmatrix} 0 \\ 0,0525 \end{pmatrix}.$$

Somit wird der Gradient im Punkt $N(t) = 15, P(t) = 5$ als 0 dargestellt.

Durch vervierfachen der Skalierung (auf das 0,1 fache der tatsächlichen Länge) kann man auch diese Gradienten sichtbar machen, verliert aber die Übersichtlichkeit der Abbildung, siehe nächste Grafik.

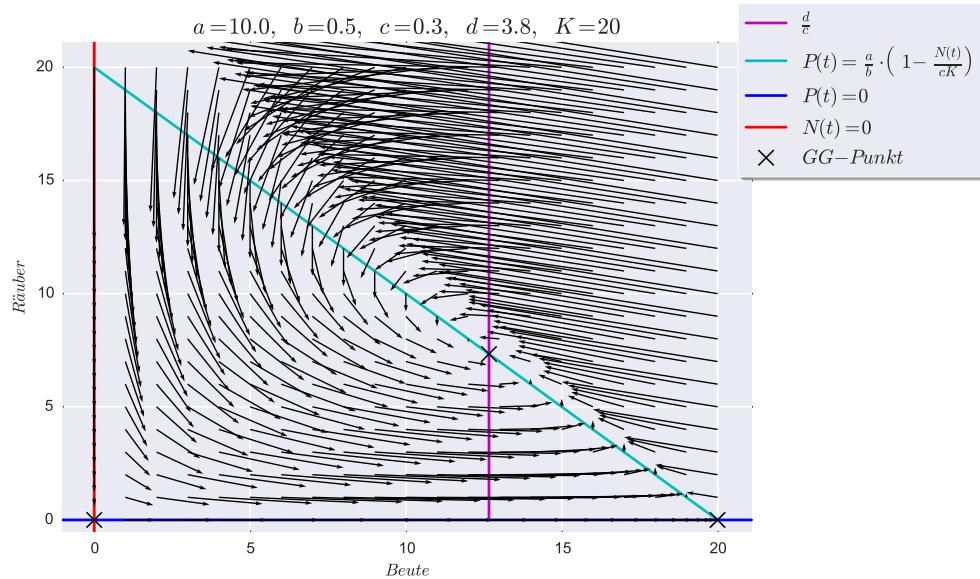


Abbildung 17: 0,1-fache Skalierung der Richtungsvektoren

Als nächsten Analyseschritt soll, wie auch im Lotka-Volterra Modell, die **qualitative Analyse** folgen. Hierbei liegt der Schwerpunkt wieder auf der Stabilität des Gleichgewichtspunktes.

6.3 Qualitative Analyse des logistischen Modells

6.3.1 Stabilität der Gleichgewichtspunkte

Es soll nun das Verhalten der Lösungskurven mit unterschiedlichen Anfangswerten außerhalb der Gleichgewichtslösungen analysiert werden. Um Vergleichbarkeit zum Lotka-Volterra Modell zu schaffen, wird, wenn möglich, auf dieselben Analysetechniken zurück gegriffen und dieselben 3 Fragen als Grundlage genommen.

1. Sind Gleichgewichtslösungen anziehend oder abstoßend für Lösungen mit Anfangswerten in der Nähe der Gleichgewichtslösungen?
2. Wie sieht das Verhalten für $t \rightarrow \infty$ aus?
3. Können Lösungskurven skizziert werden?

Für die Fragen 1 und 2 wird das Differenzialgleichungssystem für das logistische Räuber-Beute Modell wieder mithilfe der Taylorreihe linearisiert und das Verhalten der linearisierten Modelle mithilfe der Eigenwertanalyse betrachtet. Falls weitere Analyseschritte notwendig sind, werden diese im Anschluss besprochen.

6.3.2 Linearisierung im Gleichgewichtspunkt

Nutze die allgemeine Formel der Linearisierung mithilfe der Taylorreihe aus dem Lotka-Volterra Modell.

Mit der Jacobi-Matrix

$$J_{N,P} = \begin{pmatrix} \frac{\partial f_N}{\partial N(t)}(g_1, g_2) & \frac{\partial f_N}{\partial P(t)}(g_1, g_2) \\ \frac{\partial f_P}{\partial N(t)}(g_1, g_2) & \frac{\partial f_P}{\partial P(t)}(g_1, g_2) \end{pmatrix}$$

gilt

$$\begin{pmatrix} \frac{dN}{dt}(t) \\ \frac{dP}{dt}(t) \end{pmatrix} \approx J_{N,P} \cdot \begin{pmatrix} N(t) - g_1 \\ P(t) - g_2 \end{pmatrix}. \quad (\text{L1})$$

Für die Berechnung von $J_{N,P}$ wird hier nur auf die partiellen Ableitungen von $f_N(t)$ eingegangen. Die partiellen Ableitungen von $f_P(t)$ aus dem Lotka-Volterra Modell ändern sich nicht und werden daher übernommen.

$$\frac{\partial f_N}{\partial N(t)}(N(t), P(t)) = \frac{N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - bP(t) \right)}{\partial N(t)} = \frac{a \left(N(t) - \frac{N(t)^2}{K} \right) - bP(t)N(t)}{\partial N(t)}$$

$$= a \left(1 - \frac{2N(t)}{K} \right) - bP(t)$$

$$\frac{\partial f_N}{\partial P(t)}(N(t), P(t)) = \frac{N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - bP(t) \right)}{\partial P(t)} = \frac{\left(aN(t) \left(1 - \frac{N(t)}{K} \right) - bP(t)N(t) \right)}{\partial P(t)}$$

$$= -bN(t).$$

Das logistische Modell wird wieder (wie bereits beim Lotka-Volterra Modell) für den jeweiligen Gleichgewichtspunkt (hier noch allgemein mit $G = (g_1, g_2)$ bezeichnet) in den Ursprung verschoben. Somit gilt

$$\begin{aligned}
\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} &= \begin{pmatrix} \frac{dN}{dt}(t) \\ \frac{dP}{dt}(t) \end{pmatrix} + J_{N,P} \cdot \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \\
&\approx J_{N,P} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} \\
&= \begin{pmatrix} a(1 - \frac{2g_1}{K}) - g_2 b & -g_1 b \\ g_2 c & c g_1 - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix}. \tag{logL2}
\end{aligned}$$

Setzen wir $G_1(t)$, $G_2(t)$ und $G_3(t)$ in die Gleichung **logL2** ein, erhalten wir die folgenden Gleichungssysteme.

$$1. \quad G_1 = (0, 0)$$

$$\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} \approx \begin{pmatrix} a(1 - 0) - 0b & -0b \\ 0c & 0c - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix}.$$

$$2. \quad G_2 = (\frac{d}{c}, \frac{a}{b} \left(1 - \frac{\frac{d}{c}}{K}\right))$$

$$\begin{aligned}
\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} &\approx \begin{pmatrix} a \left(1 - \frac{2 \cdot \frac{d}{c}}{K}\right) - \frac{a}{b} \left(1 - \frac{\frac{d}{c}}{K}\right) \cdot b & -\frac{d}{c} \cdot b \\ \frac{a}{b} \left(1 - \frac{\frac{d}{c}}{K}\right) \cdot c & c \cdot \frac{d}{c} - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} \\
&= \begin{pmatrix} a \cdot (\frac{\frac{d}{c} - 2 \cdot \frac{d}{c}}{K}) & -\frac{d}{c} b \\ \frac{a}{b} \left(c - \frac{d}{K}\right) & 0 \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} \\
&= \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c} b \\ \frac{a}{b} \left(c - \frac{d}{K}\right) & 0 \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} := A \cdot x(t).
\end{aligned}$$

3. $G_3 = (K, 0)$

$$\begin{aligned}
\begin{pmatrix} \frac{d\tilde{N}}{dt}(t) \\ \frac{d\tilde{P}}{dt}(t) \end{pmatrix} &\approx \begin{pmatrix} a(1 - \frac{2K}{K}) - 0 \cdot b & -K \cdot b \\ 0 \cdot c & c \cdot K - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} \\
&= \begin{pmatrix} a \cdot (-1) & -Kb \\ 0 & cK - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} \\
&= \begin{pmatrix} -a & -Kb \\ 0 & cK - d \end{pmatrix} \cdot \begin{pmatrix} N(t) \\ P(t) \end{pmatrix} = B \cdot x(t).
\end{aligned}$$

Das linearisierte Differenzialgleichungssystem in der Umgebung von G_1 ist genau das selbe, wie das des Lotka-Volterra Modells. Es ist also nicht nötig, hier die Analyse ein weiteres mal durchzuführen. Für eine Skizze und das Verhalten für $t \rightarrow \infty$ des linearisierten logistischen Modells um G_1 siehe Kapitel 5.3.4 bis 5.3.6.

6.3.3 Die Lösung der Linearisierung um G_2 und G_3

Für die Lösung der Linearisierung des logistischen Räuber-Beute Modells um G_2 und G_3 wird zuerst auf Diagonalisierbarkeit getestet, anschließend diagonalisiert, um dann die zugehörige Lösung analog zum Kapitel 5.3.4 des linearisierten Lotka-Volterra Modells zu berechnen.

- G_2 :

Das charakteristische Polynom von A ist

$$\begin{aligned}
p_A(\lambda) &= \det(A - \lambda I) = \det \begin{pmatrix} -\left(\frac{ad}{cK} + \lambda\right) & -\frac{d}{c}b \\ \frac{a}{b}\left(c - \frac{d}{K}\right) & -\lambda \end{pmatrix}, \\
&= -\left(\frac{ad}{cK} + \lambda\right) \cdot (-\lambda) - \frac{a}{b} \left(c - \frac{d}{K}\right) \left(-\frac{d}{c}b\right), \\
&= \left(\frac{ad}{cK} + \lambda\right) \cdot \lambda + \frac{a}{b} \left(c - \frac{d}{K}\right) \frac{d}{c}b, \\
&= \lambda^2 + \frac{ad}{cK}\lambda + ad \left(1 - \frac{d}{cK}\right).
\end{aligned}$$

Für $p_A(\lambda) = 0$ ergibt sich

$$\begin{aligned}
\lambda_{1,2} &= \frac{-\frac{ad}{cK} \pm \sqrt{\frac{(ad)^2}{c^2K^2} - 4 \cdot 1 \cdot ad \left(1 - \frac{d}{cK}\right)}}{2}, \\
&= \frac{-\frac{ad}{cK} \pm \sqrt{\frac{(ad)^2}{c^2K^2} - \frac{4adc^2K^2}{c^2K^2} + \frac{4ad^2cK}{c^2K^2}}}{2}, \\
&= \frac{-\frac{ad}{cK} \pm \frac{1}{cK} \sqrt{ad(ad - 4c^2K^2 + 4dcK)}}{2}, \\
&= -\frac{ad}{2cK} \pm \frac{1}{2cK} \cdot \sqrt{ad(ad - 4c^2K^2 + 4dcK)}.
\end{aligned}$$

Für $ad + 4dcK < 4c^2K^2$ sind die Eigenwerte

$$\begin{aligned}
\lambda_1 &= -\frac{ad}{2cK} + i \frac{1}{2cK} \sqrt{ad(ad - 4c^2K^2 + 4dcK)}, \\
\lambda_2 &= -\frac{ad}{2cK} - i \frac{1}{2cK} \sqrt{ad(ad - 4c^2K^2 + 4dcK)}.
\end{aligned}$$

Für $ad + 4dcK \geq 4c^2K^2$ sind die Eigenwerte

$$\begin{aligned}
\lambda_1 &= -\frac{ad}{2cK} + \frac{1}{2cK} \sqrt{ad(ad - 4c^2K^2 + 4dcK)}, \\
\lambda_2 &= -\frac{ad}{2cK} - \frac{1}{2cK} \sqrt{ad(ad - 4c^2K^2 + 4dcK)}.
\end{aligned}$$

- G_3 : Da B bereits in oberer Dreiecksform ist, wissen wir, dass

$$\lambda_1 = -a, \quad \lambda_2 = cK - d.$$

Es folgt die Berechnung der Eigenvektoren.

- G_2 :

Für die Berechnung der Eigenvektoren setze

$$M := \sqrt{ad(ad - 4c^2K^2 + 4dcK)}.$$

– Für λ_1 gilt

$$\begin{aligned}
& \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 = \left(\frac{-ad+iM}{2cK} \right) \cdot v_1, \\
\iff & \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 - \left(\frac{-ad+iM}{2cK} \right) \cdot v_1 = 0, \\
\iff & \begin{pmatrix} \frac{-2ad+ad-iM}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad-iM}{2cK} \end{pmatrix} \cdot v_1 = 0, \\
\iff & \begin{pmatrix} -\frac{ad+iM}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad-iM}{2cK} \end{pmatrix} \cdot v_1 = 0, \quad \left| \cdot \begin{pmatrix} 1 & 0 \\ \frac{c}{db} \cdot \frac{ad-iM}{2cK} & 1 \end{pmatrix} \text{ v.l.} \right. \\
\iff & \begin{pmatrix} -\frac{ad+iM}{2cK} & -\frac{d}{c}b \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix}.
\end{aligned}$$

Der Eigenvektor v_1 ist also

$$v_1 = \begin{pmatrix} 1 \\ -\frac{ad+iM}{2Kbd} \end{pmatrix},$$

wobei $v_{11} = 1$ frei gewählt wurde.

– Für λ_2 gilt

$$\begin{aligned}
& \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 = \left(-\frac{ad+iM}{2cK} \right) \cdot v_1 \\
\iff & \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 - \left(-\frac{ad+iM}{2cK} \right) \cdot v_1 = 0 \\
\iff & \begin{pmatrix} \frac{-2ad+ad+iM}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad+iM}{2cK} \end{pmatrix} \cdot v_1 = 0 \\
\iff & \begin{pmatrix} \frac{-ad+iM}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad+iM}{2cK} \end{pmatrix} \cdot v_1 = 0, \quad \left| \cdot \begin{pmatrix} 1 & 0 \\ \frac{c}{db} \cdot \frac{ad+iM}{2cK} & 1 \end{pmatrix} \text{ v.l.} \right. \\
\iff & \begin{pmatrix} \frac{-ad+iM}{2cK} & -\frac{d}{c}b \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix}.
\end{aligned}$$

Der Eigenvektor v_1 ist also

$$v_2 = \begin{pmatrix} 1 \\ \frac{-ad+iMc}{2Kbd} \end{pmatrix},$$

wobei $v_{21} = 1$ gewählt wurde.

– Die Lösung lautet also

$$x(t) = c_1 \cdot \begin{pmatrix} 1 \\ \frac{-ad+iMc}{2Kbd} \end{pmatrix} \cdot e^{(-\frac{ad+iM}{2cK})t} + c_2 \begin{pmatrix} 1 \\ \frac{-ad+iMc}{2Kbd} \end{pmatrix} \cdot e^{(-\frac{ad+iM}{2cK})t}.$$

Im Fall von reellen Eigenwerten gilt Folgendes:

– Für λ_1 gilt

$$\begin{aligned} & \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 = \left(\frac{-ad+M}{2cK} \right) \cdot v_1, \\ \iff & \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 - \left(\frac{-ad+M}{2cK} \right) \cdot v_1 = 0, \\ \iff & \begin{pmatrix} \frac{-2ad+ad-M}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad-M}{2cK} \end{pmatrix} \cdot v_1 = 0, \\ \iff & \begin{pmatrix} -\frac{ad+M}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad-M}{2cK} \end{pmatrix} \cdot v_1 = 0, \quad \left| \begin{pmatrix} 1 & 0 \\ \frac{c}{db} \cdot \frac{ad-M}{2cK} & 1 \end{pmatrix} \text{ v.l.} \right. \\ \iff & \begin{pmatrix} -\frac{ad+M}{2cK} & -\frac{d}{c}b \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix} = 0. \end{aligned}$$

Der Eigenvektor v_1 ist also

$$v_1 = \begin{pmatrix} -1 \\ \frac{ad+M}{2Kdb} \end{pmatrix},$$

wobei $v_{11} = 1$ frei gewählt wurde.

– Für λ_2 gilt

$$\begin{aligned}
& \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 = \left(\frac{-ad-M}{2cK} \right) \cdot v_2 \\
\iff & \begin{pmatrix} -\frac{ad}{cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & 0 \end{pmatrix} \cdot v_1 - \left(\frac{-ad-M}{2cK} \right) \cdot v_2 = 0 \\
\iff & \begin{pmatrix} \frac{-2ad+ad+M}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad+M}{2cK} \end{pmatrix} \cdot v_2 = 0 \\
\iff & \begin{pmatrix} -\frac{ad-M}{2cK} & -\frac{d}{c}b \\ \frac{a}{b}(c - \frac{d}{K}) & \frac{ad+M}{2cK} \end{pmatrix} \cdot v_2 = 0 \quad \left| \cdot \begin{pmatrix} 1 & 0 \\ \frac{c}{db} \cdot \frac{ad+M}{2cK} & 1 \end{pmatrix} \text{ v.l.} \right. \\
\iff & \begin{pmatrix} -\frac{ad+M}{2cK} & -\frac{d}{c}b \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_{21} \\ v_{22} \end{pmatrix}.
\end{aligned}$$

Der Eigenvektor v_1 ist also

$$v_2 = \begin{pmatrix} 1 \\ \frac{-ad+M}{2Kdb} \end{pmatrix},$$

wobei $v_{21} = 1$ frei gewählt wurde.

– Die Lösung lautet also

$$x(t) = c_1'' \cdot \begin{pmatrix} -1 \\ \frac{ad+M}{2Kdb} \end{pmatrix} \cdot e^{\left(\frac{-ad+M}{2cK} \right)t} + c_2'' \begin{pmatrix} 1 \\ \frac{-ad+M}{2Kdb} \end{pmatrix} \cdot e^{\left(\frac{-ad+M}{2cK} \right)t}.$$

- G_3 :

– Für λ_1

$$\begin{aligned} & \begin{pmatrix} -a & -Kb \\ 0 & cK - d \end{pmatrix} \cdot v_1 = -a \cdot v_1 \\ \iff & \begin{pmatrix} -a + a & -Kb \\ 0 & cK - d - a \end{pmatrix} \cdot v_1 = 0 \\ \iff & \begin{pmatrix} 0 & -Kb \\ 0 & cK - d + a \end{pmatrix} \cdot v_1 = 0. \end{aligned}$$

Und somit gilt

$$v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

wobei $v_{11} = 1$ gewählt wurde.

– Für λ_2

$$\begin{aligned} & \begin{pmatrix} -a & -Kb \\ 0 & cK - d \end{pmatrix} \cdot v_1 = (cK - d) \cdot v_1 \\ \iff & \begin{pmatrix} -a - cK + d & -Kb \\ 0 & 0 \end{pmatrix} \cdot v_1 = 0. \end{aligned}$$

Und somit gilt

$$v_2 = \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix},$$

wobei $v_{21} = -1$ gewählt wurde.

– Die Lösung lautet also

$$x(t) = c_1''' \begin{pmatrix} 1 \\ 0 \end{pmatrix} e^{-a} + c_2''' \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix} \cdot e^{cK-d}.$$

6.3.4 Eigenwertanalyse von \mathbf{G}_2 und \mathbf{G}_3

Es folgt die Eigenwertanalyse, beginnend mit der Einordnung der Gleichgewichtspunkte G_2 und G_3 in die 3 Oberkategorien stabil, instabil und asymptotisch stabil und darauf folgend veranschaulicht durch Auftagen der Lösungen gegen die Zeit. Im Anschluss wird das Verhalten für $t \rightarrow \infty$ besprochen und Grafiken der zugehörigen Lösungskurven in der Phasenebene angefertigt.

Einordnen in die 3 Oberkategorien

- Für G_2 :

1. Fall: Es gilt $ad + 4dcK < 4c^2K^2$. Es sind

$$\lambda_1 = -\frac{ad}{2cK} + i\frac{1}{2cK}\sqrt{ad(ad - 4cK(cK - d))},$$

$$\lambda_2 = -\frac{ad}{2cK} - i\frac{1}{2cK}\sqrt{ad(ad - 4cK(cK - d))},$$

also

$$\Re(\lambda_1) = \Re(\lambda_2) < 0 \Rightarrow e^{-\frac{ad}{2cK}t} \xrightarrow{t \rightarrow \infty} 0,$$

und somit ist **\mathbf{G}_2 asymptotisch stabil**. Aus dem Prinzip der linearisierten Stabilität 5.3.6.3 ist bekannt, dass dieses Verhalten auf das nichtlineare logistische Ursprungsmodell übertragen werden kann und somit **\mathbf{G}_2 im logistischen Modell ebenfalls asymptotisch stabil** ist.

In folgender Grafik sind die Lösung von $N(t)$ sowie $P(t)$ gegen die Zeit aufgetragen, wobei die Parameter ($a = 10$, $b = 0.5$, $c = 0.3$, $d = 3.8$ und Kapazität $K = 20$) aus der grafischen Analyse beibehalten wurden mit Anfangswert $A_0 = (4, 10)^T$.

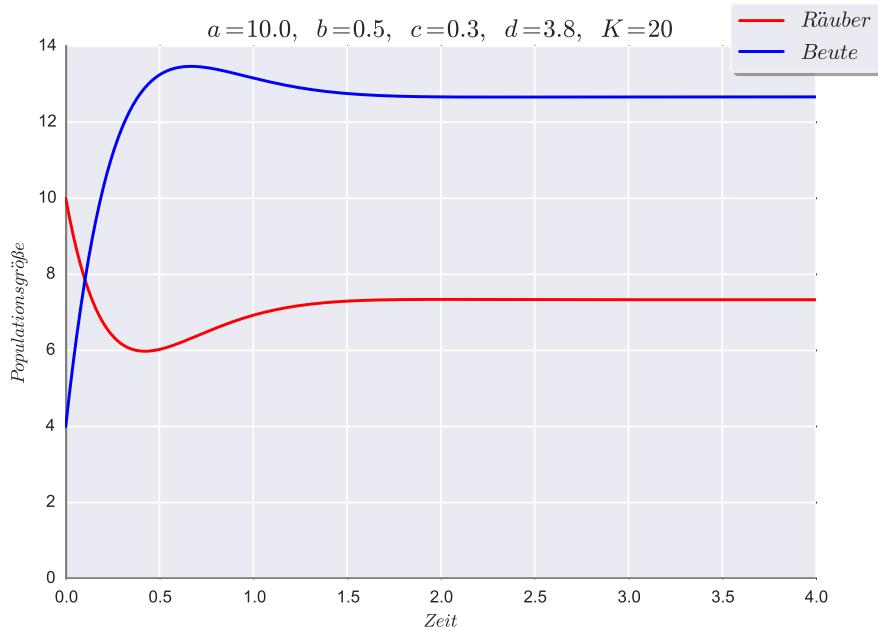


Abbildung 18: Linearisierte Lösung um G_2 , Populationsgrößen gegen die Zeit aufgetragen

Die Anziehung von G_2 scheint sehr stark. Es stellt sich die Frage, ob dies bei anderen Parametern geringer ist. Es wird deswegen $a = 2$, $b = 1.2$, $c = 0.5$, $d = 0.9$, $K = 10$ und den Anfangswert $b = (5, 2)^T$ für eine andere mögliche Gestalt der Lösungskurve gewählt.

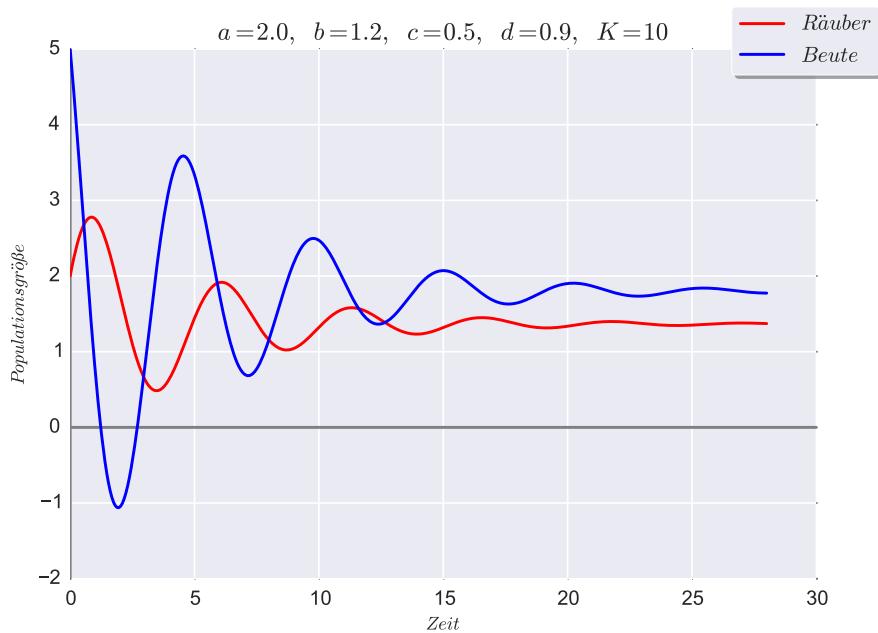


Abbildung 19: Linearisierte Lösung um G_2 , Populationsgrößen gegen die Zeit aufgetragen

Nun ist eine periodisch asymptotische Bewegung zu beobachten. Dies geht konform mit den Resultaten der grafischen Analyse, welche eine kreisförmige Bewegung um den Gleichgewichtspunkt G_2 zeigte mit eventuell asymptotischem Verhalten. Der Gleichgewichtspunkt G_2 könnte also ein Strudelpunkt sein. Siehe die Eigenwertanalyse und Phasenportrait im folgenden Kapitel 6.3.4 für eindeutige Ergebnisse.

2. Fall: Es gilt $ad + 4dcK \geq 4c^2K^2$. Es sind

$$\lambda_1 = \frac{-ad + M}{2cK},$$

$$\lambda_2 = -\frac{ad + M}{2cK} \Leftrightarrow \Re(\lambda_2) < 0.$$

Für die Grenzwertbetrachtung muss eine weitere Fallunterscheidung gemacht werden.

- $ad > M \Rightarrow \Re(\lambda_1) < 0$ und G_2 ist asymptotisch stabil.
- $ad \leq M \Rightarrow \Re(\lambda_1) > 0$ und G_2 ist instabil für $c_1 \neq 0$.

Beide Fäll sind möglich, da für

$$ad + 4dcK - 4c^2K^2 \geq 0$$

entweder

$$– ad > M \Leftrightarrow a^2d^2 > \underbrace{a^2d^2 - 4adc^2K^2 + 4ad^2cK}_{>0} \Leftrightarrow cK > d,$$

oder

$$– ad \leq M \Leftrightarrow a^2d^2 < \underbrace{a^2d^2 - 4adc^2K^2 + 4ad^2cK}_{>0} \Leftrightarrow cK \leq d$$

gilt. Es können alle Fälle abhängig von den Parametern vorkommen.

- Für G_3 : Die Eigenwerte sind

$$\lambda_1 = -a, \quad \lambda_2 = cK - d.$$

Und somit gilt

$$\Re(\lambda_1) < 0 \implies e^{-at} \xrightarrow{t \rightarrow \infty} 0.$$

Betrachte im folgenden 2 Fälle für λ_2 :

Fall 1: Für $\mathbf{cK} < \mathbf{d}$ gilt

$$\Re(\lambda_2) < 0 \implies e^{(cK-d)t} \xrightarrow{t \rightarrow \infty} 0$$

und somit ist G_3 stabil.

Die folgende Abbildung zeigt den Verlauf für Anfangswerte $a = 2$, $b = 1.2$, $c = 0.7$, $d = 9.1$, $K = 10$. Die Wahl der Parameter zeigt, wie stark d von den anderen Parametern abweichen muss, um diese Lösung möglich zu machen. Eine solche

Parameterzusammenstellung wird bei der Betrachtung einer natürlichen Population nur dann vorkommen, wenn die Räuberpopulation aufgrund verschieden möglicher Faktoren aussterben wird.

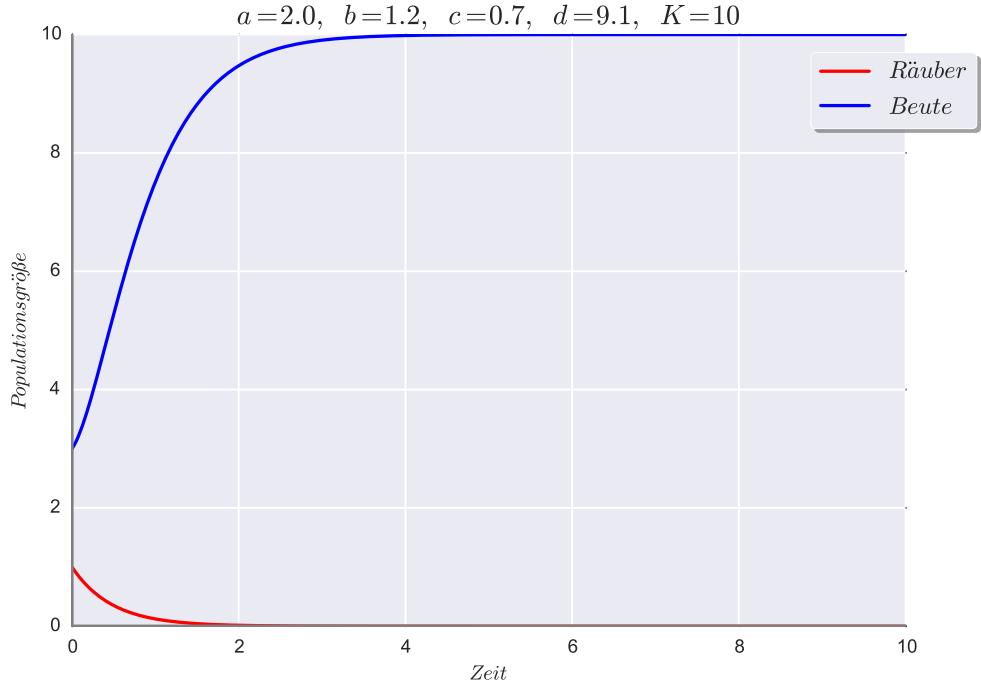


Abbildung 20: Linearisierte Lösung um G_2 , Populationsgrößen gegen die Zeit aufgetragen

Fall 2: Für $\mathbf{cK} \geq \mathbf{d}$ gilt

$$\Re(\lambda_2) \geq 0 \implies e^{(cK-d)t} \xrightarrow{t \rightarrow \infty} \infty$$

und für $c_2 \neq 0$ ist G_3 instabil.

Zusammenfassung: Im Fall von **reellen Eigenwerten** der Jacobi-Matrix von G_2 und in der allgemeinen Eigenwertbetrachtung von G_3 muss jeweils die selbe Fallunterscheidung

$$cK > d, cK < d$$

oder

$$cK = d$$

durchgeführt werden. Zusammenfassend gilt

1. $cK > d$: G_2 ist asymptotisch stabil und G_3 ist instabil,
2. $cK < d$: G_2 ist instabil und G_3 ist stabil.
3. $cK = d$: G_2 und G_3 sind instabil

Im Fall der **komplexen Eigenwerte** der Jacobi-Matrix von G_2 ist klar: G_2 ist immer stabil. Bringt dies eine Begrenzung für die Eigenschaften von G_3 ?

Es gilt

$$ad + 4dcK - 4c^2K^2 < 0.$$

- Für $cK = d$ gilt

$$0 > ad + 4dcK - 4c^2K^2 = ad + 4c^2K^2 - 4c^2K^2 = ad. \quad \nexists \text{ zu } a > 0, d > 0.$$

- Für $cK < d$ gilt

$$\begin{aligned} 0 &> ad + 4dcK - 4c^2K^2 \\ &> acK + 4cKcK - 4c^2K^2 \\ &= acK. \quad \nexists \text{ zu } a > 0, c > 0, K > 0. \end{aligned}$$

- Für $cK > d$ gilt

$$0 > ad + 4dcK - 4c^2K^2 < ad.$$

Nur der letzte Fall widerspricht der Voraussetzung $ad + 4dcK < 4c^2K^2$ nicht und ist somit der einzige Fall, der eintreten kann. G_3 ist für $cK > d$ instabil.

Zusammenfassend gilt also: Wenn G_2 komplexe Eigenwerte hat, ist G_3 instabil.

Kurvenverlauf in der Phasenebene

- Für G_2 wird zuerst der Fall mit komplexen Eigenwerten betrachtet.

1. Fall, also $\lambda_{1,2} \in \mathbb{C}$. Dann

$$\exists N \in \mathbb{R}, \text{ sodass } e^{iMt} = e^{iM(t+N)} \text{ und } \Re(e^{iMt}) \in [-1, 1]$$

und

$$\begin{aligned} \forall i = 1, 2 \ \exists t_{p_i}, t_{m_i} \in [t, t+N] : \Re(e^{iMt_{p_i}}) > 0 \text{ und } \Re(e^{iMt_{m_i}}) < 0 \\ \implies \exists t_p, t_m \in [t, t+N] : \Re(x(t_p)) > 0 \text{ und } \Re(x(t_m)) < 0, \end{aligned}$$

da G_2 in den Ursprung verschoben wurde.

G_2 ist also ein **asymptotisch stabiler Strudelpunkt**. In der folgenden Grafik ist das Verhalten des linearisierten Modells für verschiedene Anfangswerte dargestellt. Benutzt man dieselben Parameter, wie in der grafischen Analyse ($a = 10$, $b = 0.5$, $c = 0.3$, $d = 3.8$ und Kapazität $K = 20$), erhält man folgende Grafik.

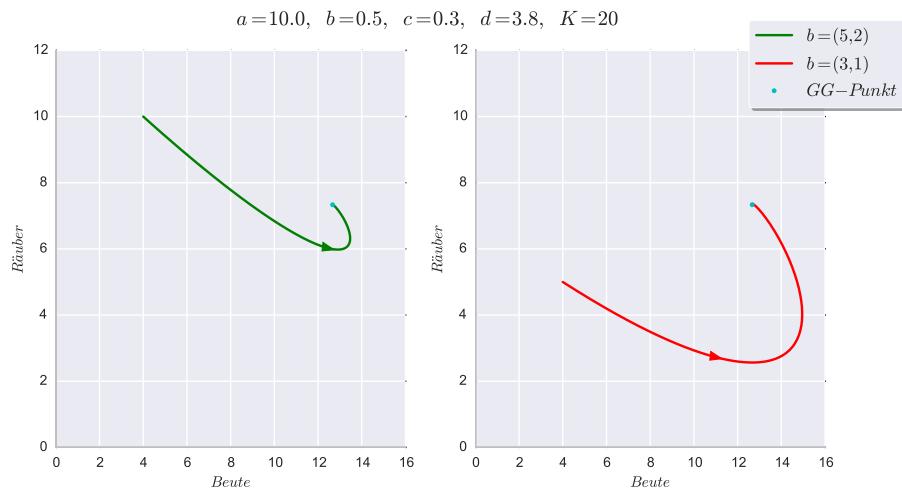


Abbildung 21: Kurvenverlauf der linearisierten Lösung um G_2 in der Phasenebene

Da die Spiraleigenschaft in oberer Grafik nicht deutlich wird, wird in folgender Grafik 2 Beispiele mit jeweils den Parametern $a = 2$, $b = 1.2$, $c = 0.5$, $d = 0.9$, $K = 10$ für 2 andere Anfangswerte dargestellt. Die Anfangswerte wechselten, da der Gleichgewichtspunkt durch die Veränderung der Parameter verschoben wird. Er ist nun deutlich näher bei der 0.

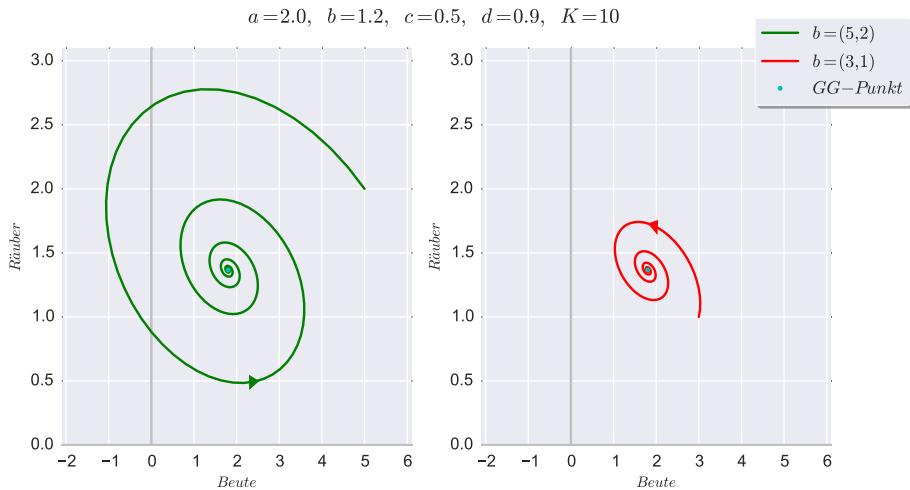


Abbildung 22: Kurvenverlauf der linearisierten Lösung um G_2 in der Phasenebene

2. Fall mit $\lambda_{1,2} \in \mathbb{R}$.

Der Grenzwert der in der Stabilitätsanalyse angesprochenen 2 Fälle für λ_1 wird zuerst analysiert:

– $ad > M$: Es gilt

$$\lim_{t \rightarrow \infty} x(t) = \lim_{t \rightarrow \infty} c_1'' \cdot \begin{pmatrix} -1 \\ \frac{ad+M}{2Kdb} \end{pmatrix} \cdot \underbrace{e^{(-\frac{ad+M}{2cK})t}}_{\rightarrow 0} + c_2'' \begin{pmatrix} 1 \\ \frac{-ad+M}{2Kdb} \end{pmatrix} \cdot \underbrace{e^{(-\frac{ad+M}{2cK})t}}_{\rightarrow 0} = 0.$$

Die Lösungen konvergieren gegen die Eigenvektoren v_1, v_2 zum Gleichgewichtspunkt G_2 hin. G_2 ist ein **stabiler Knotenpunkt**.

– $ad < M$: Für $c_2 \neq 0$ gilt

$$\lim_{t \rightarrow \infty} x(t) = \lim_{t \rightarrow \infty} c_1'' \cdot \begin{pmatrix} -1 \\ \frac{ad+M}{2Kdb} \end{pmatrix} \cdot \underbrace{e^{(-\frac{ad+M}{2cK})t}}_{\rightarrow \infty} + c_2'' \begin{pmatrix} 1 \\ \frac{-ad+M}{2Kdb} \end{pmatrix} \cdot \underbrace{e^{(-\frac{ad+M}{2cK})t}}_{\rightarrow 0}.$$

Somit ist G_2 ein **Sattelpunkt**.

- Für G_3 wird direkt in die Grenzwertbetrachtung der Fallunterscheidung für λ_2 über gegangen.

1. Fall: $cK < d$

Für den Limes gilt

$$\lim_{t \rightarrow \infty} x(t) = \lim_{t \rightarrow \infty} c_1' \begin{pmatrix} 1 \\ 0 \end{pmatrix} \underbrace{e^{-at}}_{\rightarrow 0} + c_2' \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix} \cdot \underbrace{e^{(cK-d)t}}_{\rightarrow 0} = 0.$$

Die Lösungen konvergieren gegen die Eigenvektoren v_1, v_2 zum Gleichgewichtspunkt G_3 , welcher ein **stabiler Knotenpunkt** ist. In folgender Grafik ist dieser Fall dargestellt. Die Parameter wurden als $a = 2$, $b = 1.2$, $c = 0.7$, $d = 9.1$, $K = 10$ gewählt, sodass $0 < Kc - d = 1.9 \neq 2 = a$. Letztere Bedingung ist nötig, da S sonst singulär wird.

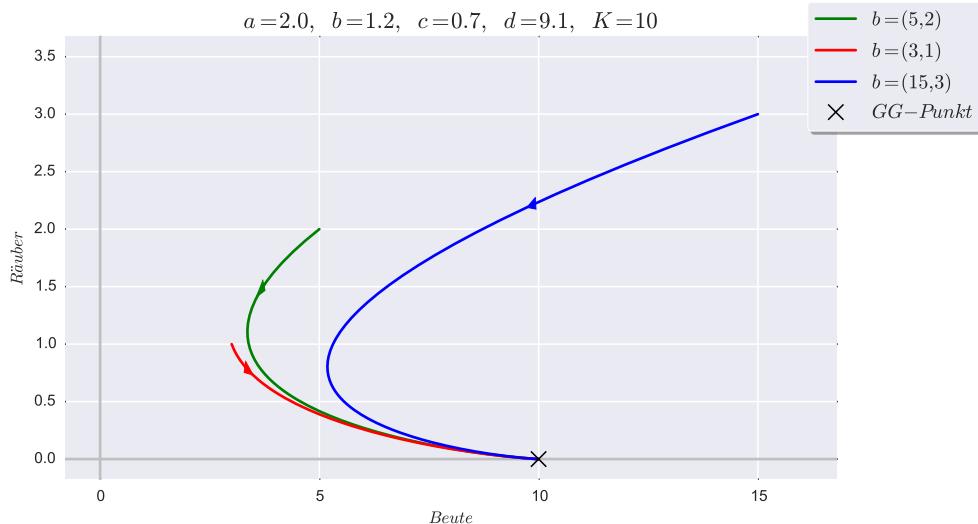


Abbildung 23: Phasenplot der linearisierten Lösung um G_3

2. Fall: $cK > d$:

Es gilt

$$\lim_{t \rightarrow \infty} x(t) = \lim_{t \rightarrow \infty} c'_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \underbrace{e^{-at}}_{\rightarrow 0} + c'_2 \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix} \cdot \underbrace{e^{(cK-d)t}}_{\rightarrow \infty} = c'_2 \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix} \cdot \infty.$$

Hierbei ist zu beachten, dass nicht für jeden Anfangswert in der Nähe von G_3 mit $cK > d$ der Gleichgewichtspunkt instabil ist. Im Folgenden wird der Fall $b = (b_1, 0)^T$ sowie ein zweiter Fall mit $b_1 \neq 0 \neq b_2$ betrachtet und in der Fallunterscheidung das Wissen genutzt, dass

$$S = (v_1, v_2) \text{ und } S^{-1} = \begin{pmatrix} 1 & \frac{Kb}{a+cK-d} \\ 0 & \frac{Kb}{a+cK-d} \end{pmatrix}$$

sind und das in G_3 verschobene System untersucht.

- Der Anfangswert ist $(b_1, 0)^T$:

Dann ist

$$c'_1 = b_1 - K \text{ und } c'_2 = 0,$$

somit gilt für den Grenzwert

$$\lim_{t \rightarrow \infty} x_3(t) = \lim_{t \rightarrow \infty} \underbrace{c'_2}_{=0} \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix} \cdot \underbrace{e^{(cK-d)t}}_{\rightarrow \infty} + \begin{pmatrix} K \\ 0 \end{pmatrix} = \begin{pmatrix} K \\ 0 \end{pmatrix}.$$

Die Lösung konvergiert also entlang v_1 zum Punkt $G_3 = (K, 0)$ und ist somit ein **stabiler Knotenpunkt**.

- Der Anfangswert ist $(b_1, b_2)^T$:

Dann ist

$$\lim_{t \rightarrow \infty} x(t) = \lim_{t \rightarrow \infty} c'_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \underbrace{e^{-at}}_{\rightarrow 0} + \underbrace{c'_2}_{\neq 0} \begin{pmatrix} -1 \\ \frac{a+cK-d}{Kb} \end{pmatrix} \cdot \underbrace{e^{(cK-d)t}}_{\rightarrow \infty},$$

G_3 ist also ein **Sattelpunkt**.

In allen Fällen kann immer das Prinzip der linearisierten Stabilität aus Satz 5.3.6.3 angewendet werden. Wir wissen also, dass die Gleichgewichtspunkte auch im logistischen Modell die jeweiligen Eigenschaften haben, welche in diesem Kapitel für das linearisierte logistische Modell berechnet wurden.

6.3.5 Skizzieren der nichtlinearen Lösung des logistischen Räuber-Beute Modells

Es bleibt die Frage, ob eine Skizze der Lösungskurve des nichtlinearen logistischen Räuber-Beute Modells angefertigt werden kann. In diesem Fall kann die Steigung des

Graphen auf der Phasenebene nicht weiter helfen, da die resultierende Gleichung

$$\frac{dP}{dN} = \frac{P(t)(cN(t) - d)}{N(t) \left(a \left(1 - \frac{N(t)}{K} \right) - bP(t) \right)}$$

nicht separabel ist und somit eine Lösung nicht leicht zu finden ist.

Für eine Skizze wird in diesem Fall auf numerische Methoden zurück gegriffen. Als ersten Schritt wurde das Stromlinienfeld des Differenzialgleichungssystems mithilfe der Funktion `streamplot()` in python erstellt (siehe [Pytc]). Für die Betrachtung der Programme mit dem die folgenden 2 Grafiken erstellt wurden, siehe D.2, Zeilen 140-200 und die Aufrufe in Zeile 736 und Zeile 777 für unterschiedliche Parameter. Die Farbcodierung zeigt die Stärke des Richtungsvektors in N-Richtung an. Es werden wieder beide oben bereits benutzten Parameter dargestellt. In den folgenden Grafiken sind die Gleichgewichtslösungen $P(t) = 0$ und $N(t) = 0$ nicht mehr farbig hervorgehoben, um die Grafik übersichtlicher zu gestalten.

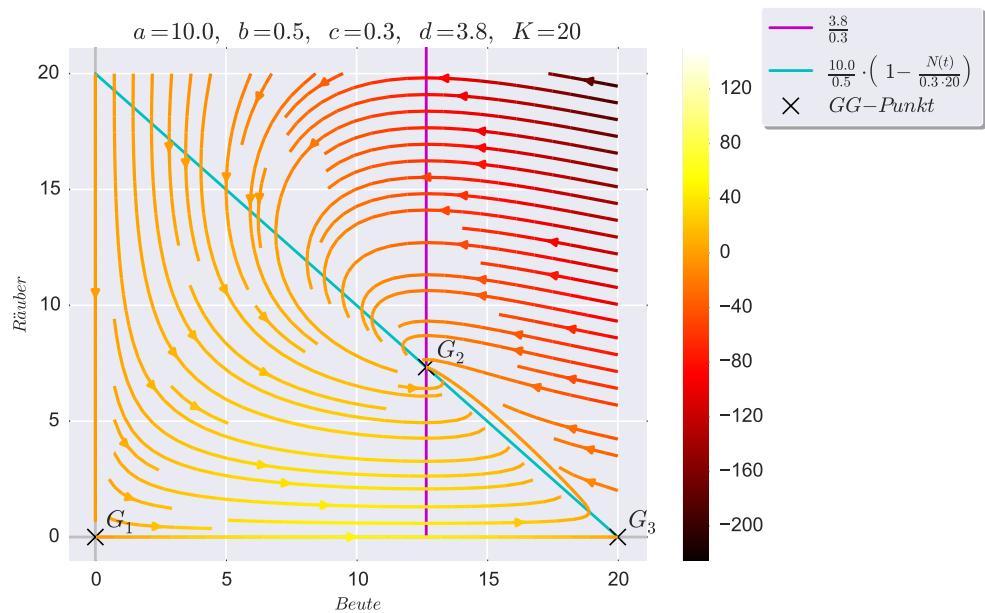


Abbildung 24: Stromlinienfeld des logistischen Lotka-Volterra Modells für Parametergruppe 1.

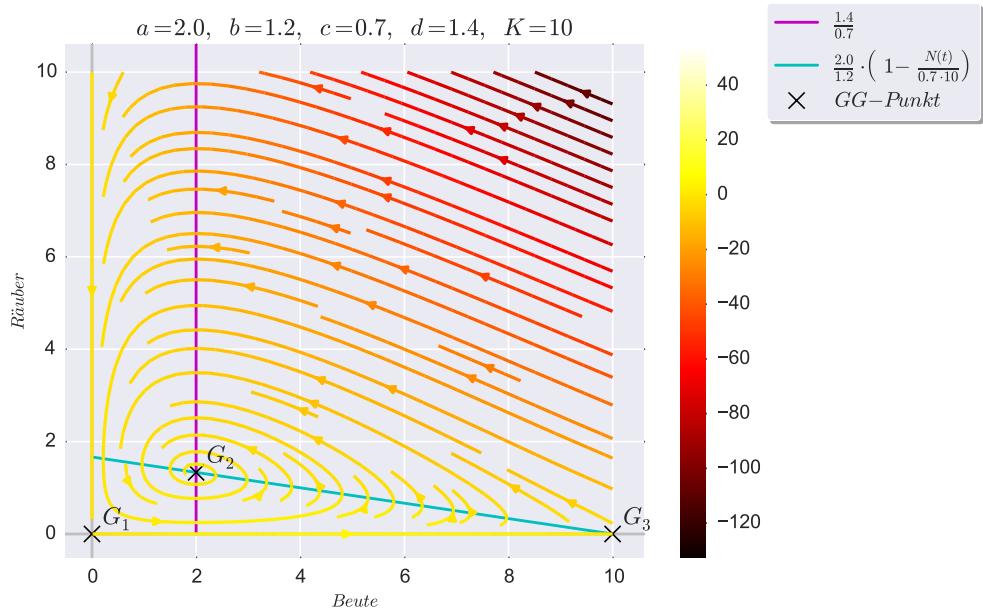


Abbildung 25: Stromlinienfeld des logistischen Lotka-Volterra Modells für Parametergruppe 2.

Als weiteren Schritt kann man die Funktion `odeint()` in python (siehe [Pytd]) benutzen. Diese kann Systeme von Differentialgleichungen numerisch lösen. In folgender Grafik sind mehrere Lösungen zu 4 verschiedenen Anfangswerten, welche von `odeint()` berechnet wurden, in die Phasenebene gezeichnet. Zur Betrachtung des Codes mit welchem die Lösungen berechnet wurden, sowie die folgenden Grafiken erstellt wurden, siehe D.2, Zeilen 591-654 mit `odeint()`-Aufrufen in den Zeilen 609-612 und dem Aufruf der Zeichen-Funktion in der Zeile 733.

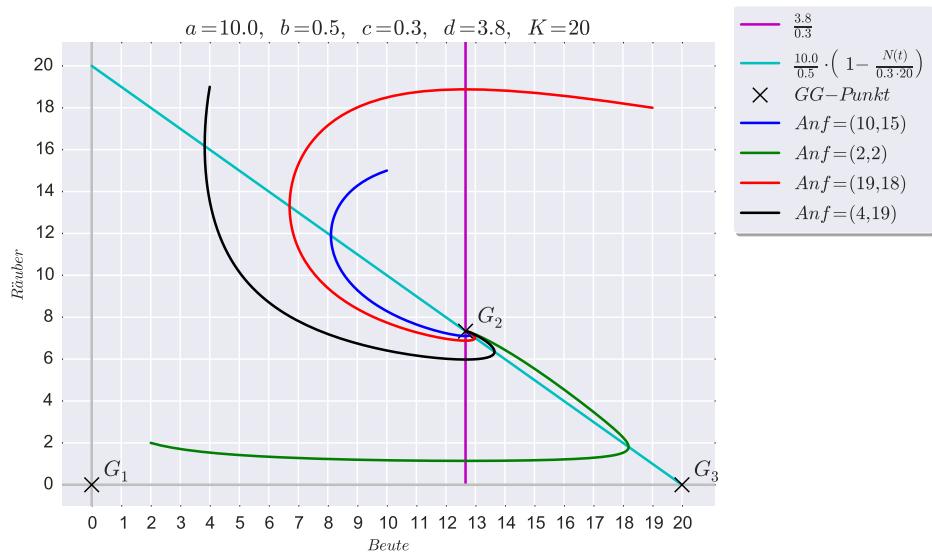
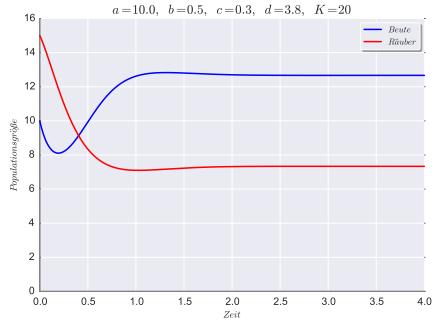
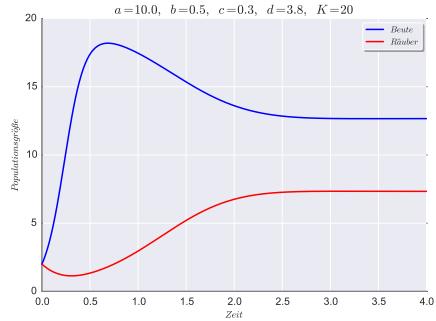


Abbildung 26: Phasenplot der Lösung mit unterschiedlichen Anfangswerten

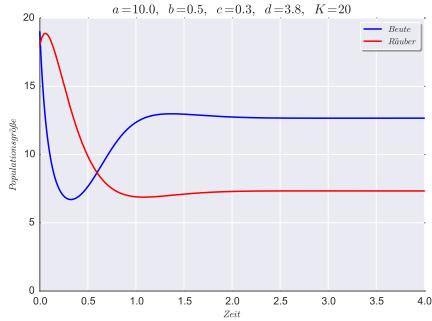
Die folgende Grafik zeigen zu genau diesen 4 Anfangswerten jeweils die 2 Lösungen für $N(t)$ und $P(t)$ gegen die Zeit aufgetragen.



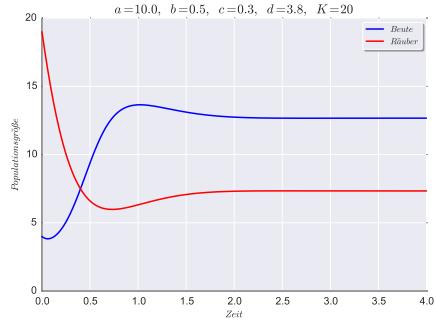
(a) Lösung zum Anfangswert $A_0 = (10, 15)$ gegen die Zeit



(b) Lösung zum Anfangswert $A_0 = (2, 2)$ gegen die Zeit



(c) Lösung zum Anfangswert $A_0 = (19, 18)$ gegen die Zeit



(d) Lösung zum Anfangswert $A_0 = (4, 19)$ gegen die Zeit

Abbildung 27: Lösungen des logistischen Differenzialgleichungssystems gegen die Zeit aufgetragen

6.4 Diskussion des logistischen Lotka-Volterra Modell

Die Veränderung durch das logistische Wachstum im Modell hat die Modelleigenschaften deutlich verändert.

6.4.1 Vergleich zu Lotka-Volterra Modell

In diesem Kapitel sollen die zwei in dieser wissenschaftlichen Arbeit behandelten Modelle verglichen werden. Hierzu wurde der Datensatz über die Schneeschuhhasen und kanadischen Luchs benutzt (siehe [Texa]).

Die Beziehung der beiden Arten ist eine in der Natur selten vorkommende nahezu reine Räuber-Beute Beziehung von Seiten des kanadischen Luchses (siehe [Texb],[Kan], Hunting and diet). Er ernährt sich hauptsächlich von einer Beuteart, dem Schneeschuhhasen ([Kan], Hunting and diet). Diese stellen 60-97% der Nahrung des Luchses ([Kan], Hunting and diet). Die Räuber-Beute Beziehung ist aufgrund dieser starken einseitigen Abhängigkeit des Luchses an den Schneeschuhhasen gut geeignet um ein zwei-Populations-Räuber-Beute Modell zu testen. Ein Erhalt der Luchspopulation kann nur durch genügend Schneeschuhhasen gewährleistet werden, dies ist in beiden Modellen eine Grundannahme an die Räuberpopulation.

Der hier verwendete Datensatz wurde von 1845 bis 1867 dokumentiert durch die Zählung der Felle beider Populationen (siehe [Texb] und [Web]).

Die Abbildung 28 zeigt den Graphen für das Lotka-Volterra Modell, mit den am besten passenden Parametern⁹ und den Originaldaten in einem Graph, um bestmögliche Vergleichbarkeit zu erhalten. Die Abbildung 29 zeigt die Graphen für das logistische Modell für denselben Grundaufbau wie in Abbildung 28.

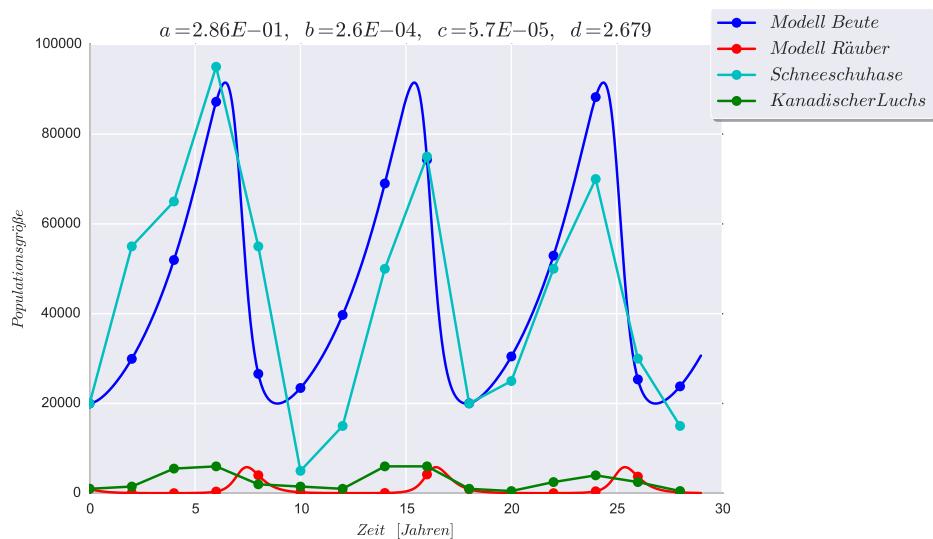


Abbildung 28: Vergleich **Lotka-Volterra Modell** zu bekannten Datenpunkten

⁹Parameterwerte wurden durch eigenhändige Anpassung der Modelle an die Datenpunkte bestimmt. Hierbei wurde das Ziel, die Parameter für beide Modelle mit der besten Anpassung an die Datenpunkte zu finden, verfolgt. Die Wahl der Parameter fand objektiv und ohne Präferenz für eines der Modelle statt.

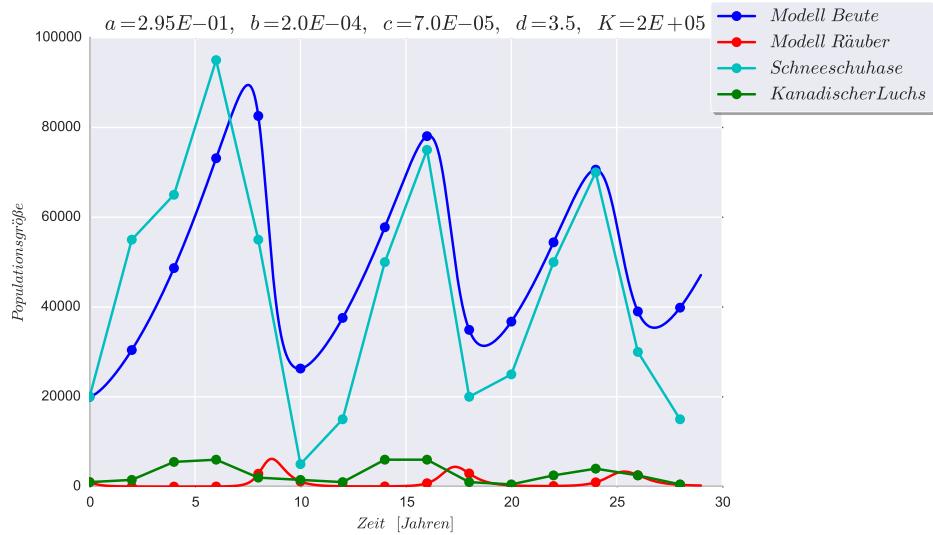


Abbildung 29: Vergleich **logistisches Modell** zu bekannten Datenpunkten

Das logistische Modell wurde erstellt, um das Lotka-Volterra Modell zu verbessern. Hierbei wurde das Augenmerk auf die Beutepopulation gelegt.

Vergleich der Modelle für die Beutepopulation: In den Abbildungen 28 und 29 ist ein deutlicher Unterschied in Bezug auf die Güte der Modelle für die Beutepopulation sichtbar.

Das **Lotka-Volterra Modell** erreichte eine gute Anpassung für das mittlere Minimum, eine Aussage über das dritte Minimum ist schlecht zu treffen, da der Anstieg in den Datenpunkten leider nicht mehr zu sehen ist. Die deutlich sichtbare Reduktion der Amplitude der Schwankung in den Datenpunkten in Maximum 2 und 3 konnte durch das Lotka-Volterra Modell nicht gut modelliert werden. Die 2 letzten Maxima sind nicht nur im Bezug auf die Amplitude nicht gut repräsentiert sondern auch im Bezug auf das zeitliche Auftreten der Maxima. Besonders das zweite Maxima liegt leicht links vom tatsächlichen Maximum, aber die Datenpunkte stimmen überein. Dies kann zu Fehlern in der Berechnung der Güte der Approximation führen und ist ein Effekt des geringen Datenvolumens.

Das **logistische Modell** zeigt eine sehr gute Anpassung an die Reduktion der Amplitude in Maxima 2 und 3, ist aber leider leicht verschoben im ersten Maximum. Die Anpassung an die Minima ist nicht so gut wie beim Lotka-Volterra Modell. Betrachtet man die Werte des Modells der Beutepopulation mit den tatsächlichen Datenpunkten sind die Differenzen der letzten 2 Maxima relativ gering (besonders bevor, an und nach den letzten 2 Maxima). Die Datenpunkte für das erste Maximum zeigen relativ große Differenzen, obwohl die Anpassung relativ gut ist. Dies ist ein Resultat der leichten Verschiebung zu Beginn des Maximums und der geringen Datenpunkte.

Vergleich der Modelle für die Räuberpopulation: Da in der Anpassung der Parameter für die gegebenen Datenpunkte der Schwerpunkt auf der Beutepopulation lag, sind die Räuberpopulationen in beiden Fällen schlechter modelliert als die Beutepopulation.

Im **Lotka-Volterra Modell** passt die Amplitude für die ersten 2 Maxima gut, ist aber zu groß für das letzte Maximum. Die Maxima der Räuber Modelle sind für das erste und letzte Maximum verschoben. Man kann dies durch die Anpassung der Parameter nach links verschieben, dadurch verschiebt sich aber auch die Beutepopulation und wird somit zu einer extrem schlechten Approximation für die Beute.

Generell herauslesen kann man hierbei, dass im Lotka-Volterra Modell die Räuberpopulation immer der Beutepopulation folgt und ein deutlich kürzer anhaltendes Maximum hat, als die Datenpunkte. In den Datenpunkten scheint die Räuberpopulation relativ zeitgleich mit der Beutepopulation zu wachsen, allerdings nicht schnell sondern eher langanhaltend.

Dasselbe gilt für das **logistische Modell**. Zusätzlich hat es im allgemeinen eine gute Anpassung an die Amplituden, etwas zu gering im Maximum zwei. Die beste Anpassung findet sich in Maximum drei, in dem die letzte Hälfte des Maximums nahezu exakt modelliert wurde.

Aber auch hier bleibt das Problem bestehen, dass die Räuberpopulation der Beutepopulation folgt mit einem kurzen Maximum.

Zusammenfassung und Verbesserungsvorschläge: Es kann gefolgert werden, dass das Ziel des logistischen Modells, das Lotka-Volterra Modell zu verbessern, zu weiten Teilen funktioniert hat. Sowohl die Beutepopulation als auch die Räuberpopulation konnten für diesen Datensatz über diese kurze gemessene Zeitspanne eine gute Repräsentation der Realität geben.

Es gibt allerdings zwei Problemstellen:

1. der gemessene Zeitraum ist zu kurz.
2. die Räuberpopulation ist in beiden Modellen zu schlecht modelliert.

Zu 1: In einem Zeitraum von 30 Jahren zeigen die Populationen 3 Schwankungen mit langsam abnehmender Amplitude. Dies ist zu wenig, um auf ein Langzeitverhalten schließen zu können. Das logistische Modell ist eine gute Anpassung für diese Datenpunkte für diese 30 Jahre, aber wir wissen, dass das Modell zu seinem Gleichgewichtspunkt konvergieren wird. Das Lotka-Volterra Modell ist keine so gute Anpassung an die Datenpunkte in diesen 30 Jahren, und wir wissen, dass das Modell periodisch genau dieselben Zyklen vorhersagen wird, wie den für diese 30 Jahre.

In der Diskussion um das bessere Modell ist es essenziell zu wissen, ob die Populationen weiter periodisch schwanken werden, oder ebenfalls zu einem Gleichgewichtspunkt hin tendieren werden. Die Abnahme in den Datenpunkten kann ein vorübergehender Effekt aufgrund von veränderten Umweltbedingungen sein, oder ein fester Bestandteil

einer Koevolution dieser zwei Populationen zu diesem Zeitpunkt, wie es das logistische Modell annimmt. Ein weiterer wichtiger Faktor ist, dass die Populationen, startend 1845 bejagt wurden ([Texb]) und somit für beide Populationen ein neuer zusätzlicher Selektionsdruck ausgeübt wird.

Zu 2: Die Modellierung der Räuberpopulation ist zu schlecht. Wie in der Diskussion des Lotka-Volterra Modells schon angedeutet, gibt es noch weitere Verbesserungsmöglichkeiten. In beiden in dieser wissenschaftlichen Arbeit behandelten Modellen sind die Räuberpopulationen "Nimmersatts". Sie investieren außerdem keine Zeit in die Jagd und die zufälligen Begegnungen mit Beutetieren haben den einzigen und direkten Einfluss auf die Reproduktionsrate der Räuber.

Verbesserungsmöglichkeiten sind hier

1. den Faktor der Jagd,
2. die Begrenzung der Nahrungsaufnahme der Räuber

in die Gleichung der Räuber im logistischen Modell zu integrieren.

Der Faktor der Jagd bedeutet, dass Räuber Zeit und Energie für die Jagd von Beutetieren investieren müssen. Hierbei können die Räuber die Interaktionshäufigkeit mit den Beutetieren erhöhen, ohne dass dabei die Beutepopulation groß sein muss. Somit kann eventuell der Anstieg der Räuberpopulationsgröße nach links verschoben werden.

Um den Anstieg und das Maximum langanhaltender zu gestalten, wäre eine Verbesserung hin zu einer verzögerten Reproduktion und einer Schranke in der Fähigkeit zur Nahrungsaufnahme sinnvoll. Dies wurde bereits in der Diskussion über die Schwachpunkte des Lotka-Volterra Modells besprochen (siehe Unterkapitel 5.4). Da für das zweite Modell der Fokus auf die Verbesserung der Beutepopulationen gelegt wurde, wurde es in dieser wissenschaftlichen Arbeit nicht weiter verfolgt.

7 Abschluss

In dieser wissenschaftlichen Arbeit wurden zwei der ersten und einfachsten Räuber-Beute-Modelle erstellt, auf ihre Eigenschaften analysiert und anhand eines Datensatzes auf Brauchbarkeit getestet. Obwohl es zahlreiche Möglichkeiten zur Verbesserung gibt, waren beide Modelle bereits eine sehr gute Anpassung an eine nahezu reine Räuber-Beute Beziehung, wie die des kanadischen Luchses und Schneeschuhhasen. Heutzutage werden jedoch weder das Lotka-Volterra Modell, noch das logistische Modell häufig zur Modellierung genutzt.

Seit ihrer Entwicklung gab es viele Verbesserungen und Erweiterungen des logistischen Lotka-Volterra Modells. Die berühmteste Verbesserung resultierte in das so genannte Rosenzweig-McArthur Modell (Publiziert 1963 [RM]). Es nutzt eine bestimmte Gruppe von Funktionen, die sogenannten "functional response"-Funktionen zur Modellierung der Bejagungs- und Tötungsrate der Räuber an der Beute, wobei der Term als "functional-response"-Funktionen 1949 von Solomon¹⁰ bestimmt wurde ([Hol59] S. 303). Diese Tötungsrate wurde in die Räuber- sowie Beutegleichung des logistischen Modells integriert (siehe auch [Tur13], S. 46). Die im Rosenzweig-McArthur-Modell verwendeten "functional response"-Funktionen wurden von Holling 1959 entwickelt ([Hol59]). Sie beschreiben in der Natur häufig beobachte Konsumverhaltensänderungen der Räuberpopulationen bei unterschiedlicher Beutepopulationsdichte. Diese Funktionen sind beschränkte Wachstumskurven und können in ihrer Form variieren. Somit sind die Wachstumsraten der zwei Populationen nicht mehr proportional zur jeweiligen Größe der anderen Population, wie beim reinen, sowie logistischen Lotka-Volterra Modell der Fall.

Das Rosenzweig-McArthur Modell wurde erfolgreich für verschiedene "real-life" Anwendungen verwendet. (übersetzt aus [Tur13], S.46). Die bisher in ökologischen Modellierungen meist verwendete "functional response"-Funktion, die sogenannte "Holling Type III functional response" (siehe [Tur13], S.82) hat einen sigmoiden Kurvenverlauf, bei linear steigender Beute-Dichte. Diese Form einer "functional response"-Funktion ist typisch für Generalisten (siehe [Tur13], S.82f), da sie, bei geringer Populationsdichte der aktuellen "Lieblings"-Beuteart, diese typischerweise wechseln (siehe [Tur13], S.82f). Eine Population mit wenigen Individuen zu bejagen ist sehr aufwändig und für Generalisten deswegen nicht lohnenswert. Für Spezialisten ist eher die "Holling type II functional response" mit hyperbolischem Kurvenverlauf die passende Wahl (siehe [Tur13], S.82f), da sie auf diese eine Beute stark angewiesen sind und deswegen selbst bei geringer Beutepopulationsdichte ihre Nahrung von dieser Population beziehen (siehe [Tur13], S.82f).

Diese Form der Modellierung der Räuber-Beute Beziehung war lange die Standardmethode, bis 1989 ein paper mit der sogenannten "Arditi-Ginzburg-ratio-dependent-functional-response" (siehe [JSB07], S. 7) publiziert wurde, welches sogenannte "ratio-dependent functional response"-Funktionen einführte (siehe [JSB07], S. 7). Diese verursachten eine große Kontroverse über die besser geeignete "functional-

¹⁰Solomon, M. E. "The Natural Control of Animal Populations." Journal of Animal Ecology. 19.1 (1949). 1-35

response"-Funktion für die Modellierung der Räuber-Beute Beziehungen, da der Gebrauch der Einen den Gebrauch der anderen ausschließt (Zusammenfassung aus [JSB07], S.8). Seither wurden viele paper produziert, mit der Hoffnung, die Kontroverse auflösen zu können (siehe [JSB07], S.8). Eine Dissertation ([JSB07]) die zu diesem Thema 2007 erschienen ist, nennt diese Kontroverse "the unresolved controversy" ([JSB07], S.7). In dieser Dissertation wurden mehrere Gründe für den weiteren Bestand der Kontroverse, trotz der vielen Mühen, diese zu beseitigen, angeführt. Einer der Hauptgründe ist die Schwierigkeit, kontrollierte empirische Daten von zwei-Populations-Räuber-Beute-Beziehungen zu erstellen (siehe [JSB07], S.8). Durch einen Vergleich der "functional-response"- und "ratio-dependent-functional-response"-Funktionen anhand empirisch erhobener Datensätze könnte geklärt werden, welches Modell letztendlich besser passt und solange dies nicht möglich ist, wird die Kontroverse wohl weiter bestehen.

Hiermit schließe ich meine wissenschaftliche Arbeit ab. Dargestellt wurde die historische Entwicklung der Modellierung von Räuber-Beute Modellen bis in die heutige Zeit, darüber hinaus gewährte diese Arbeit einen allgemeinen Einblick in die Modellierung in der Biologie. Im zentralen Abschnitt wurden die ersten 2 wichtigen und wegbereitenden Räuber-Beute Modelle entwickelt und analysiert und deren historische Verbesserungen im abschließenden Kapitel besprochen.

Appendix

Dieser Appendix beinhaltet eine kurze Einführung in Differenzialgleichungen, sowie detaillierte Einblicke in die Exponentialfunktion und die Lösung von Differenzialgleichungen. Als letztes Kapitel wurden die Programme, mit denen alle Grafiken in dieser wissenschaftlichen Arbeit, bis auf die allgemeine Darstellung eines Strudel- und Sattelpunktes (Abbildung 9b und Abbildung 6), erstellt wurden, eingefügt. Diese habe ich mit der open-source Software "inkscape"¹¹ selbstständig entworfen und erstellt.

A Differenzialgleichungen

A.1 Gewöhnliche Differenzialgleichung erster Ordnung

Definition A.1.1 (Differenzialgleichung).

Sei g eine Funktion von t , $x(t)$ und $y(t)$ und $\frac{dx}{dt}(t)$, wobei $x(t)$ und $y(t)$ Funktionen auf $(a, b) \in \mathbb{R}$ sind und $t \in (a, b) \subseteq \mathbb{R}$.

Dann nennt man eine Gleichung der Form

$$0 = g\left(t, x(t), y(t), \frac{dx}{dt}(t)\right)$$

eine **gewöhnliche Differenzialgleichung erster Ordnung**.

Hierbei nennt man t die **abhängige Variable** und $x(t)$, sowie $y(t)$ die **unabhängigen Variablen**.

Die **Ordnung einer Differenzialgleichung** ist die Ordnung der höchsten Ableitung in der Gleichung.

Durch Umformen kann man die Differenzialgleichung oft auf die bekanntere Schreibweise

$$\frac{dx}{dt}(t) = f_x(t, x(t), y(t))$$

bringen. Hierbei ist f_x eine Funktion von t , $x(t)$ und $y(t)$. Da die Differenzialgleichungen, welche in dieser wissenschaftlichen Arbeit behandelt werden, immer auf die obere Form umgeformt werden können, werden die Formen äquivalent behandelt.

Definition A.1.2 (Autonome Differenzialgleichung).

Man nennt eine **Differenzialgleichung autonom**, wenn die Funktion g nicht explizit von der Variable t abhängt.

Beispiel A.1.3.

Sei f eine Funktion von $x(t)$ und $y(t)$, wobei $x(t)$ und $y(t)$ Funktionen auf $(a, b) \in \mathbb{R}$ sind und $t \in (a, b) \subseteq \mathbb{R}$.

Sei weiterhin die Differenzialgleichung

$$\frac{dx}{dt}(t) = x(t) + y(t) - 1 := f(x(t), y(t))$$

¹¹Homepage: <https://inkscape.org/en/>

gegeben.

Diese Differenzialgleichung ist eine **autonome** gewöhnliche Differenzialgleichung erster Ordnung.

Bemerkung A.1.4.

Autonome Differenzialgleichungen hängen nicht direkt von t ab. Die Differenzialgleichung in Beispiel A.1.3 hat also für feste $y(t)$ und $x(t)$ immer dieselbe Ableitung. Dies wird im Folgenden grafisch verdeutlicht.

Definition A.1.5 (Steigungsfeld).

Sei eine gewöhnliche Differenzialgleichung erster Ordnung gegeben. Eine Grafik, welche die Steigung von $x(t)$ in Punkt t in einem bestimmten Intervall $t \in (a, b)$ für regelmäßige $t \in (a, b)$ anzeigt, nennt man **Steigungsfeld**.

Definition A.1.6 (Isokline).

Für eine Differenzialgleichung

$$\frac{dx}{dt}(t) = f(x(t))$$

nennt man jede Kurve, die einer Gleichung des Typs $f(x) = a$ (a konstant) genügt, ein **Isokline**.

Isokline leitet sich aus dem griechischen (isos = gleich, klinein = neigen)¹² ab und kann übersetzt werden als "gleiche Steigung".

Beispiel A.1.7.

Im Folgenden sollen 2 Beispiele eines Steigungsfeldes gezeigt werden.

1. Sei

$$\frac{dy}{dt}(t) = t + y(t)$$

eine gewöhnliche Differenzialgleichung erster Ordnung.

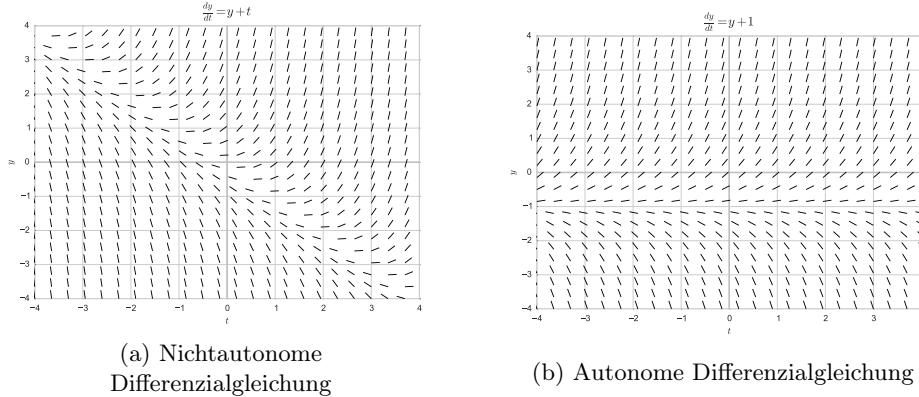
2. Sei

$$\frac{dy}{dt}(t) = y(t) + 1$$

eine autonome Differenzialgleichung erster Ordnung.

¹²siehe <https://de.wikipedia.org/wiki/Isokline> 16.09.2015, 15.17 Uhr PST

Abbildung 30: Grafische Darstellung von Beispiel 1 und 2



Bemerkung A.1.8.

Im oberen Beispiel wird deutlich, dass für autonome Differenzialgleichungen **der Wert für $\frac{dy}{dt}(t)$ für bestimmtes y immer gleich bleibt für alle $t \in (-4, 4)$** . Dies gilt für alle autonomen Differenzialgleichungen, da die Funktion g , wie in Definition A.1.2 nicht mehr von t abhängt.

Ein Isokline von autonomen Differentialgleichungen ist also waagrecht.

A.2 Lösungen von Differenzialgleichungen

Definition A.2.1 (Lösung einer gewöhnlichen Differenzialgleichung).

Es sei eine gewöhnliche Differenzialgleichung erster Ordnung gegeben. Dann nennt man

$$x(t)$$

eine **Lösung** der Differenzialgleichung, falls $x(t)$ die Gleichung

$$\frac{dx}{dt}(t) = f(t, x(t), y(t)) \quad \forall t \in (a, b)$$

erfüllt.

Eine Differenzialgleichung kann mehrere Lösungen haben. Man nennt ein einzelnes $z(\bullet)$ eine **spezielle Lösung** von g . Die **allgemeine Lösung** einer Differenzialgleichung ist die Menge aller möglichen Lösungen dieser Differenzialgleichung.

Beispiel A.2.2.

Nehme die autonome Differenzialgleichung

$$\frac{dy}{dt}(t) = y(t) + 1.$$

Die Lösung ist

$$y(t) = e^{t+c} - 1,$$

wobei $c \in \mathbb{R}$ eine beliebige Konstante ist. Somit ist $y(t)$ die **allgemeine Lösung** der Differenzialgleichung und es gibt unendlich viele Lösungen.

Sei $c = 0$, dann ist

$$y_0(t) = e^t - 1$$

eine **spezielle Lösung** der Differenzialgleichung.

Definition A.2.3 (System von gewöhnlichen Differenzialgleichungen erster Ordnung).

Ein **System von gewöhnlichen Differenzialgleichungen erster Ordnung** ist ein System von n Gleichungen

$$\frac{dy_1}{dt}(t) = f_1(t, y_1(t), y_2(t), \dots, y_n(t))$$

$$\frac{dy_2}{dt}(t) = f_2(t, y_1(t), y_2(t), \dots, y_n(t))$$

⋮

$$\frac{dy_n}{dt}(t) = f_n(t, y_1(t), y_2(t), \dots, y_n(t)),$$

wobei f_1, f_2, \dots, f_n Funktionen der abhängigen Variablen t und der unabhängigen Variablen $y_1(t), y_2(t), \dots, y_n(t)$ sind. Hierbei sind f_1, f_2, \dots, f_n definiert auf einer geeigneten Menge $S \in \mathbb{R}^{n+1}$.

Definition A.2.4 (Lösung eines Systems an Differenzialgleichungen).

Eine **Lösung eines Differenzialgleichungssystems** ist eine Familie von Funktionen $y_1(\bullet), y_2(\bullet), \dots, y_n(\bullet)$, jeweils definiert auf einem gemeinsamen Intervall in der Menge \mathbb{R} , welche alle Differenzialgleichungen im System erfüllen.

B Matrixexponentialfunktion

B.1 Die Matrixexponentialfunktion

Definition B.1.1 (Die Matrixexponentialfunktion).

Sei $A \in L(X)$, $X = \mathbb{C}^k$, $t \in \mathbb{R}$. Dann ist die zu A gehörige Matrixexponentialfunktion definiert als

$$\mathbb{R} \rightarrow L(X),$$

$$t \mapsto e^{tA} = \sum_{n=0}^{\infty} \frac{(tA)^n}{n!} = \sum_{n=0}^{\infty} \frac{t^n A^n}{n!}.$$

Bemerkung B.1.2.

Die Matrixexponentialfunktion e^{tA} von $A \in L(X)$ ist wohldefiniert für alle $t \in \mathbb{R}$.

Beweis. siehe LDS [Nag12], Satz 6.4, S.28. □

B.2 Sätze und Definitionen für die Matrixexponentialfunktion und ihre Asymptotik

Satz B.2.1 (Eigenschaften von e^{tA}).

Die Matrixexponentialfunktion hat folgende Eigenschaften.

1. $e^{0A} = Id.$
2. $e^{tSAS^{-1}} = Se^{tA}S^{-1}$, für $S \in \mathcal{L}(X)$ invertierbar.
3. $e^{(t+s)A} = e^{tA+sA} = e^{tA}e^{sA}$ für alle $s, t \in \mathbb{R}$ (Funktionalgleichung)
4. Die Abbildung $\phi : \mathbb{R} \rightarrow L(X), t \mapsto e^{tA}$ ist stetig.
5. ϕ ist außerdem stetig differenzierbar mit

$$\frac{d}{dt}e^{tA} = Ae^{tA} = e^{tA}A$$

für alle $t \in \mathbb{R}$.

6. Seien 2 kommutierende Matrizen $D, N \in L(X)$ (d.h. $DN = ND$) gegeben. Dann gilt für die Matrixexponentialfunktion

$$e^{tD} \cdot e^{tN} = e^{t(D+N)} = e^{tN}e^{tD}.$$

Die Matrizen

$$D = \begin{pmatrix} d & 0 & \dots & 0 \\ 0 & d & & \\ \vdots & & \ddots & \vdots \\ & \dots & d & 0 \\ 0 & \dots & \dots & 0 & d \end{pmatrix}, d \in \mathbb{R} \text{ und } N = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ & & & 0 & 1 \\ 0 & \dots & \dots & 0 & 0 \end{pmatrix}$$

kommutieren.

Beweis. Beweise einzeln nacheinander.

1.

$$e^{0A} = \sum_{n=0}^{\infty} \frac{0^n A^n}{n!} = 1 \cdot Id + \underbrace{\sum_{n=1}^{\infty} \frac{0 A^n}{n!}}_{=0} = Id$$

2.

$$\begin{aligned}
 e^{tSAS^{-1}} &= \sum_{n=0}^{\infty} \frac{t^n (SAS^{-1})^n}{n!} \\
 &\stackrel{*}{=} \sum_{n=0}^{\infty} \frac{t^n S A^n S^{-1}}{n!} & * \underbrace{SAS^{-1} S \dots AS^{-1}}_{n \text{ mal}} = SA^n S^{-1} \\
 &\stackrel{**}{=} S \left(\sum_{n=0}^{\infty} \frac{t^n A^n}{n!} \right) \cdot S^{-1} & ** S \text{ unabhängig von } n \\
 &= Se^{tA}S^{-1}
 \end{aligned}$$

3. Funktionalgleichung

$$\begin{aligned}
 e^{tA} e^{sA} &= \left(\sum_{n=0}^{\infty} \frac{t^n A^n}{n!} \right) \cdot \left(\sum_{n=0}^{\infty} \frac{s^n A^n}{n!} \right) \\
 &\stackrel{*}{=} \sum_{n=0}^{\infty} \sum_{l=0}^n \frac{(tA)^{n-l} (sA)^l}{(n-l)! \cdot l!} & * \text{ Cauchy Produkt} \\
 &= \sum_{n=0}^{\infty} \sum_{l=0}^n \frac{n!}{(n-l)! \cdot l!} \frac{(tA)^{n-l} (sA)^l}{n!} \\
 &= \sum_{n=0}^{\infty} \sum_{l=0}^n \binom{n}{l} \frac{(tA)^{n-l} (sA)^l}{n!} \\
 &\stackrel{**}{=} \sum_{n=0}^{\infty} \frac{(tA + sA)^n}{n!} & ** \text{ binomische Formel} \\
 &= \sum_{n=0}^{\infty} \frac{(t+s)^n A^n}{n!} \\
 &= e^{(t+s)A}.
 \end{aligned}$$

4. Eine Funktion $\phi : \mathbb{R} \rightarrow L(X)$ ist genau dann stetig auf \mathbb{R} , falls für alle $t_0 \in \mathbb{R}$ gilt $\lim_{t \rightarrow t_0} \phi(t) = \phi(t_0)$. Dies ist der Fall, wenn für alle Folgen $(t_n)_{n \in \mathbb{N}} \in \mathbb{R}$ mit $\lim_{n \rightarrow \infty} t_n = t_0$ gilt, dass $\lim_{n \rightarrow \infty} \phi(t_n) = \phi(t_0)$, $\forall t_0 \in \mathbb{R}$.

Sei also $(t_n)_{n \in \mathbb{N}} \in \mathbb{R}$ eine beliebige Folge mit $\lim_{n \rightarrow \infty} t_n = t_0$.

$$\begin{aligned}
 &\Rightarrow \lim_{n \rightarrow \infty} t_n - t_0 = 0 \\
 &\Rightarrow 1 = e^{\lim_{n \rightarrow \infty} (t_n - t_0)A} \\
 &\stackrel{*}{=} \lim_{n \rightarrow \infty} e^{(t_n - t_0)A}. & * \text{Summe ist stetig}
 \end{aligned}$$

Wegen der Funktionalgleichung gilt nun

$$\begin{aligned}
e^{t_0 A} &= e^{t_0 A} \cdot \lim_{n \rightarrow \infty} e^{(t_n - t_0) A} \\
&= \underbrace{e^{t_0 A} \cdot e^{-t_0 A}}_{=1} \cdot \lim_{n \rightarrow \infty} e^{t_n A} \\
&= \lim_{n \rightarrow \infty} e^{t_n A}.
\end{aligned}$$

5. siehe Seite 29 [Nag12].

6. Für $e^{tD} e^{tN}$ gilt

$$\begin{aligned}
e^{tD} e^{tN} &= \left(\sum_{n=0}^{\infty} \frac{t^n D^n}{n!} \right) \cdot \left(\sum_{n=0}^{\infty} \frac{t^n N^n}{n!} \right) \\
&\stackrel{*}{=} \sum_{n=0}^{\infty} \sum_{l=0}^n \frac{(tD)^{n-l} (tN)^l}{(n-l)! \cdot l!} && * \text{ Cauchy Produkt} \\
&= \sum_{n=0}^{\infty} \sum_{l=0}^n \frac{n!}{(n-l)! \cdot l!} \frac{(tD)^{n-l} (tN)^l}{n!} \\
&= \sum_{n=0}^{\infty} \sum_{l=0}^n \binom{n}{l} \frac{(tD)^{n-l} (tN)^l}{n!} \\
&\stackrel{**}{=} \sum_{n=0}^{\infty} \frac{(tD + tN)^n}{n!} && ** \text{ binomische Formel} \\
&= \sum_{n=0}^{\infty} \frac{t^n (D + N)^n}{n!} \\
&= e^{t(D+N)}.
\end{aligned}$$

Für $D \in L(X)$ beliebige Diagonalmatrix gilt

$$D \cdot N = \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & & \\ \vdots & & \ddots & \vdots \\ & \dots & d & 0 \\ 0 & \dots & \dots & 0 & d_n \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ & & & 0 & 1 \\ 0 & \dots & \dots & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & d_1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ & & & 0 & d_{n-1} \\ 0 & \dots & \dots & 0 & 0 \end{pmatrix}$$

und

$$N \cdot D = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ & & 0 & 1 & \\ 0 & \dots & \dots & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & & \\ \vdots & & \ddots & \vdots \\ & & \dots & d & 0 \\ 0 & \dots & \dots & 0 & d_n \end{pmatrix} = \begin{pmatrix} 0 & d_2 & 0 & \dots & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ & & & 0 & d_n \\ 0 & \dots & \dots & 0 & 0 \end{pmatrix},$$

und somit kommutieren $D = \text{diag}(d)$ und N .

□

Satz B.2.2.

Sei ein System von autonomen linearen Differenzialgleichungen wie in A.2.3 mit Anfangswert wie in 5.3.1.1 gegeben, welches beschrieben werden kann als

$$\frac{dy}{dt}(t) = A \cdot y(t), \quad y(t_0) = b,$$

mit $b = (b_1, \dots, b_n)$, $y(t) = (y_1(t), \dots, y_n(t))$, wobei $(A \cdot y(t))_k = f_k(y_1(t), \dots, y_n(t))$. Die Einträge in A sind also die Koeffizienten für die lineare Komposition der $y_i(t)$, $\forall i = 1, \dots, n$. Dann ist für alle $t \in \mathbb{R}$ die eindeutige Lösung des Anfangswertproblems gegeben als

$$y(t) = e^{tA} \cdot b.$$

Hierbei ist $b \in \mathbb{R}$ der Anfangswert.

Beweis. Sei $X = \mathbb{C}^k$, $A \in L(X)$, $b = (b_1, b_2, \dots, b_k)$, sodass

$$\frac{dy}{dt}(t) = Ay(t), \quad t \in \mathbb{R},$$

$$x(0) = b.$$

Dann ist $e^{tA} \cdot b = y(t)$ eine Lösung des Anfangswertproblems, denn nach Satz B.2.1 gilt

$$\frac{dy}{dt}(t) = \frac{d}{dt} e^{tA} \cdot b = Ae^{tA} \cdot b = Ay(t).$$

Außerdem gilt

$$x(t_0) = e^{0 \cdot A} \cdot b = 1 \cdot b = b.$$

Aufgrund des Existenz- und Eindeutigkeitssatzes 5.3.1.5 ist $x(t) = e^{tA} \cdot b$ die einzige Lösung des Anfangswertproblems. \square

C Die Lösung eines zweidimensionalen linearen Systems von Differenzialgleichungen

Der Vollständigkeit halber wird zuerst die Definition eines zweidimensionalen Systems von linearen Differenzialgleichungen angegeben und im Anschluss Eigenschaften davon diskutiert.

Es werden zuerst 2 Sätze dargestellt, die einen Überblick über die Anzahl an möglichen Eigenwerten und deren Eigenschaften verschafft. Im Anschluss folgen 2 Sätze, welche die Lösungen von zweidimensionalen linearen Differenzialgleichungen darstellen. Hierbei muss zwischen diagonalisierbaren und nicht diagonalisierbaren Jacobimatrizen unterschieden werden.

Definition C.0.1.

Gegeben sei

$$\begin{aligned}\frac{dx}{dt}(t) &= ax(t) + by(t) \\ \frac{dy}{dt}(t) &= cx(t) + dy(t),\end{aligned}$$

wobei $a, b, c, d > 0$, $x(t)$ und $y(t)$ die unabhängigen Variablen sind. Ein solches System von linearen Gleichungen kann man auch in Matrixschreibweise mit der Matrix

$$A := \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

umformulieren. Es hat dann die Form

$$\frac{dz}{dt}(t) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot z(t) = A \cdot z(t), \quad \text{mit } z(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}. \quad (\text{LDS 1})$$

Dies nennt man ein **System von linearen Differenzialgleichungen** oder **Lineares dynamisches System**.

Lemma C.0.2.

Sei $p(x) = \sum_{i=1}^n a_i x^i$, $i \in \mathbb{N}$, $a_i \in \mathbb{R}$ ein reelles Polynom. Dann

1. existieren genau $n \in \mathbb{N}$ Nullstellen $\{x_1, \dots, x_n\} \in \mathbb{C}$ von $p(x)$, wenn die Vielfachheit gezählt wird.
2. gilt für jedes $x_i \in \mathbb{C}$ komplex, dass \bar{x}_i , (das komplex konjugierte von x) auch eine Nullstelle von $p(x)$ ist.

Beweis. Die erste Aussage ist der *Fundamentalsatz der Algebra*. Siehe Satz und Beweis in [LM11] S. 202 ff.

Die zweite Aussage ist eine direkte Folgerung des Fundamentalsatzes der Algebra.

Sei hierbei $z \in \mathbb{C}$ eine Lösung des Polynoms $p(x)$, dann gilt

$$\begin{aligned}
p(z) &= 0 \\
&= \bar{0} \\
&= \overline{p(z)} \\
&= \overline{\sum_{i=1}^n a_i z^i} \\
&= \sum_{i=1}^n a_i \bar{z}^i \quad \text{wg. Linearität der Abb. } \phi : \mathbb{C} \rightarrow \mathbb{C}, z \mapsto \bar{z} \text{ in Add. \& Mult.} \\
&= p(\bar{z}).
\end{aligned}$$

Und somit gilt für jedes $z \in \mathbb{C}$ mit $p(z) = 0$ auch $p(\bar{z}) = 0$. \square

Die folgenden beiden Sätze sind anhand [Sto08] als Vorlage entwickelt worden und benutzen das Wissen aus der Vorlesung von Prof. Rainer Nagel ([Nag12]). Für die Stabilitätsanalyse einer Matrix müssen 2 Fälle unterschieden werden.

1. A ist diagonalisierbar,
2. A ist nicht diagonalisierbar.

Satz C.0.3 (A diagonalisierbar).

Sei ein zweidimensionales lineares Differenzialgleichungssystem wie in C.0.1 gegeben und A sei diagonalisierbar. Dann ist die Lösung dieses Differenzialgleichungssystems gegeben als

$$z(t) = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t}.$$

Dabei sind v_1, v_2 Eigenvektoren zu Eigenwerten λ_1, λ_2 sind und c_1, c_2 allgemeine Konstanten, welche durch die Anfangswerte des Systems festgelegt werden.

Beweis. Da $A \in L(X)$ diagonalisierbar ist, existiert ein $S, S^{-1} \in L(X)$, wobei S invertierbar (siehe [Nag12]), sodass $S^{-1}AS = D$ gilt mit

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \quad S = (v_1, v_2).$$

Hierbei sind v_1, v_2 Eigenvektoren zu den Eigenwerten λ_1, λ_2 (bekannt aus Lineare Algebra).

Die Lösung des Differenzialgleichungssystems ist also

$$z(t) = e^{tA} \cdot b = S e^{tD} S^{-1} b.$$

Fasst man $S^{-1} \cdot b = (c_1, c_2)^T$ zusammen und multipliziert diese Gleichung aus, erhält man die Lösung. \square

Satz C.0.4 (A nicht diagonalisierbar).

Sei ein zweidimensionales lineares Differenzialgleichungssystem wie in C.0.1 gegeben und A sei **nicht** diagonalisierbar. Dann ist die Lösung dieses Differenzialgleichungssystems gegeben als

$$z(t) = (c_1 v_1 + c_2 (tv_1 + u)) e^{\lambda t}.$$

Dabei sind v_1 der Eigenvektor zum Eigenwert λ ist und c_1, c_2 allgemeine Konstanten, welche durch die Anfangswerte des Systems festgelegt werden. u ist hierbei der Hauptvektor der Stufe 2 (Bekannt aus Lineare Algebra II, Thema Jordan Normalform).

Beweis. Es existieren $S \in \mathcal{L}(X)$ invertierbar, sodass $S^{-1}AS = J$ gilt mit

$$J = \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix} = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} := D + N, \quad S = (v_1, u) = \begin{pmatrix} s_1 & s_3 \\ s_2 & s_4 \end{pmatrix},$$

d.h. J besitzt Jordan Normalform und v_1 ist ein Eigenvektor zu Eigenwert λ und u ein Hauptvektor zweiter Stufe.

Dann gilt für die Lösung des Differenzialgleichungssystems

$$z(t) = e^{tA} \cdot b = Se^{tJ}S^{-1}b = Se^{tD+tN}S^{-1}b.$$

Da D und N kommutieren, gilt der Binomische Lehrsatz und es folgt $e^{tD+tN} = e^{tD} \cdot e^{tN}$ (siehe B.2.1 Nummer 6).

Da $N^n = 0$, $\forall n > 1$, gilt

$$e^{tN} = Id + t \cdot N = \begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix}.$$

Da die Eigenwerte gleich sind, hat e^{tD} in diesem Fall die Form

$$e^{tD} = Id \cdot e^{t\lambda}.$$

Somit gilt für die Lösung des Differenzialgleichungssystems

$$z(t) = Se^{tD}e^{tN}S^{-1}b = Se^{t\lambda} \cdot \begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix} S^{-1}b = e^{t\lambda} \cdot \begin{pmatrix} s_1 & ts_1 + s_3 \\ s_2 & ts_2 + s_4 \end{pmatrix} \cdot c = e^{t\lambda} (v_1, v_1 \cdot t + u) \cdot c.$$

Fasst man wieder $S^{-1}b = (c_1, c_2)^T := c$ zusammen, erhält man die Lösung. \square

D Programme zur Erstellung der Graphiken

D.1 Lotka-Volterra Modell

D.1.1 2D-Grafiken

```
1 #-----
2 # Zeichne Grafiken der Loesungen fuer G_1 und G_2 des Lotka-Volterra
3 #----- Modells.
4 #-----
5 # Importieren der noetigen python-libraries.
6 import numpy as np
7 from scipy import linalg
8 import pylab as pl
9 import seaborn
10 import cmath as cm
11 from math import *
12 from matplotlib import rcParams
13
14 def GG_points(a,b,c,d,maxi):
15     """
16         Zeichnet und speichert eine Grafik der Gleichgewichtsgeraden
17         fuer gegebene Parameter und einem Maximalwert "maxi",
18         welcher als oberste Grenze fuer die x- und y-Achse dient.
19     """
20
21     # Initialisiere numpy-Arrays fuer x- und y-Achse.
22     N_1 = pl.arange(0,maxi+1.3,maxi*0.05)
23     P_1 = pl.arange(0,maxi+1.3,maxi*0.05)
24
25     # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
26     GG_x = np.array([0,d/c])
27     GG_y = np.array([0,a/b])
28
29     # Festlegen des Formats der Grafik.
30     fig = plt.figure(figsize=(8,8))
31     rcParams['xtick.labelsize'] = 12
32     rcParams['ytick.labelsize'] = 12
33     rcParams.update({'font.size': 15})
34     plt.xlabel(r'$Beute$', fontsize=15)
35     plt.ylabel(r'$R\"uber$', fontsize=15)
36     plt.grid(True)
37     plt.ylim(-.1,maxi+1)
38     plt.xlim(-.1,maxi+1)
39
40     # Zeichne die 4 Gleichgewichtsgeraden und speichere Namen
41     # fuer Legende.
```



```

72      # Festlegen des Formats der Grafik.
73      fig = plt.figure(figsize=(8,8))
74      rcParams['xtick.labelsize'] = 12
75      rcParams['ytick.labelsize'] = 12
76      rcParams.update({'font.size': 15})
77      plt.xlabel(r'$Beute$', fontsize=15)
78      plt.ylabel(r'$R\"{a}uber$', fontsize=15)
79      plt.grid(True)
80      plt.ylim(-.1,2*a/b)
81      plt.xlim(-.1,2*a/b)
82
83      # Zeichne die 4 Gleichgewichtsgeraden und speichere Name fuer
84      # Legende.
85      plt.plot([d/c]*len(P_1), P_1, color='m', label=r'$\frac{d}{c}x$')
86      plt.plot(N_1, [a/b]*len(N_1), color="c", label=r'$\frac{a}{b}x$')
87      plt.axhline(color="b", label=r'$P(t) = 0$')
88      plt.axvline(color="r", label=r'$N(t) = 0$')
89
90      # Zeichne und beschrifte die Schnittpunkte.
91      plt.plot(GG_x,GG_y,marker="x",color="k",linestyle="",mec="k",
92      ms=10,mew=1,label=r'$GG$-Punkt')
93      plt.text(GG_x[0]+0.02*maxi,GG_y[0]+0.02*maxi,'$G_1$', fontsize
94      =15)
95      plt.text(GG_x[1]+0.02*maxi,GG_y[1]+0.02*maxi,'$G_2$', fontsize
96      =15)
97
98      ## Zeichne die jeweiligen Vektoren in jedes der 4 Felder.
99
100     # Unteres rechtes Feld.
101     pl.arrow(1.5* d/c,0.5*a/b,d/c*0.1*maxi, 0, fc="k", ec="k",
102     head_width=0.05, head_length=0.05,color="k",
103     length_includes_head=True )
104     pl.arrow(1.5* d/c,0.5*a/b,0, d/c*0.1*maxi, fc="k", ec="k",
105     head_width=0.05, head_length=0.05,color="k",
106     length_includes_head=True )
107     pl.arrow(d/c,0.5*a/b,d/c*0.1*maxi, 0, fc="k", ec="k",
108     head_width=0.05, head_length=0.05,color="k",
109     length_includes_head=True )
110
111     # Unteres linkes Feld.
112     pl.arrow(0.5* d/c,0.5*a/b,d/c*0.1*maxi, 0, fc="k", ec="k",
113     head_width=0.05, head_length=0.05,color="k",
114     length_includes_head=True )
115     pl.arrow(0.5* d/c,0.5*a/b,0, -d/c*0.1*maxi, fc="k", ec="k",

```

```

    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )
105 pl.arrow(0.5*d/c,a/b,0, -d/c*0.1*maxi, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )

106
107 # Oberes linkes Feld.
108 pl.arrow(0.5* d/c,1.5*a/b,-d/c*0.1*maxi, 0, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )
109 pl.arrow(0.5* d/c,1.5*a/b,0, -d/c*0.1*maxi, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )
110 pl.arrow(d/c,1.5*a/b,-d/c*0.1*maxi, 0, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )

111
112 # Oberes rechtes Feld.
113 pl.arrow(1.5* d/c,1.5*a/b,-d/c*0.1*maxi, 0, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )
114 pl.arrow(1.5* d/c,1.5*a/b,0, d/c*0.1*maxi, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )
115 pl.arrow(1.5*d/c,a/b,0, d/c*0.1*maxi, fc="k", ec="k",
    head_width=0.05, head_length=0.05,color="k",
    length_includes_head=True )

116
117
118 # Zeichnen der Legende und schreibe die Parameter ueber die
    Grafik.
119 lgd = plt.legend(loc="upper right", fancybox=True, frameon=
    True, shadow=True, bbox_to_anchor=(1, 1), fontsize=13)
120 plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f$%"(
    a,b,c,d), fontsize=15)

121
122 # Speichere die Grafik und zeige sie direkt an.
123 plt.savefig("GG_points_vectors_lotvolt.pdf",
    bbox_extra_artists=(lgd,), bbox_inches='tight')
124 plt.show()

125
126 def vectorfield(a,b,c,d,maxi,scalar):
127     """
128     Zeichnet ein Vektorfeld fuer die gegebenen Parameter und dem
        Maximalen Wert maxi.
129     "scalar" ist ein Skalar, mit welchem die Skalierung der
        Vektorlaengen festgelegt wird.

```

```

130     Die Funktion gibt den Aufruf zum Zeichnen der Legende zurueck
131     und haelt die Grafik im Speicher ausserhalb des
132     Funktionsspeichers.
133     """
134
135     # Initialisiere ein numpy-Array fuer die x- und y-Achse und
136     # baue aus letzterem ein Netzwerk aus Punkten zur Berechnung
137     # der Richtungsvektoren in jedem Punkt dieses Netzwerkes.
138     # Speichere dies zwei mal als je eine Koordinaten-Matrix.
139     N_1 = pl.arange(0,maxi+0.02*maxi,maxi*0.05)
140     P_1 = pl.arange(0,maxi+0.2*maxi,maxi*0.05)
141     N,P = pl.meshgrid(N_1, N_1)
142
143     # Berechne die Steigung der Beute als U und die der Rauber
144     # als V.
145     U = N*(a-b*P)
146     V = P*(c*N-d)
147
148     # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
149     GG_x = np.array([0,d/c])
150     GG_y = np.array([0,a/b])
151
152     # Festlegen des Formats der Grafik.
153     pl.figure(figsize=(8,8))
154     plotstyleset()
155     plt.ylim(0,maxi)
156     plt.xlim(0,maxi)
157
158     # Verschiebe die ganze Grafik um 0.05 nach rechts oben fuer
159     # uebersichtlichere und ansprechendere Grafik.
160     l,r,b1,t = pl.axis()
161     dx, dy = r-l, t-b1
162     pl.axis([l-0.05*dx, r+0.05*dx, b1-0.05*dy, t+0.05*dy])
163
164     # Zeichne die 4 Gleichgewichtsgeraden und speichere Name fuer
165     # Legende.
166     plt.plot([d/c]*len(P_1),P_1,color='m',zorder=1, label=r"\$\
167             frac{d}{c}\$")
168     plt.plot(N_1,[a/b]*len(N_1), color="c",zorder=2,label=r"\$\
169             frac{a}{b}\$")
170     plt.axhline(color="b",label=r'\$P(t) = 0\$',zorder=3)
171     plt.axvline(color="r",label=r'\$N(t) = 0\$',zorder=4)
172
173     # Zeichne die Schnittpunkte der Gleichgewichtsgeraden
174     plt.plot(GG_x,GG_y,marker="x",color="k",linestyle="",mec="k",
175             ms=10,mew=1,label=r"\$GG-Punkt\$",zorder=5)

```



```

199      # Berechne c=S*Anf, da die Inverse der Einheitsmatrix wieder
199      # die Einheitsmatrix ergibt.
200      c = S.dot(Anf)
201
202      # Berechne die Loesungen fuer die Beute N und die Raeuber P
202      # und ersetze eine weitere 0 pro Iteration in den Arrays N
202      # und P.
203      for t_0,t in enumerate(time):
204          N[t_0] = c[0]*np.exp(t*a)
205          P[t_0] = c[1]*np.exp(t*d*(-1.0))
206
207      # Gebe die Arrays der Beutepopulation, Raeuberpopulation und
207      # Zeit zurueck, sodass die Funktion zum erstellen der
207      # Grafik die Loesungsfunktion aufrufen kann.
208      return N,P,time
209
210 def solution_G2(a,b,c,d,Anf):
211     """
212     Berechne die Loesung zum Anfangswert "Anf" mit den gegebenen
212     Parametern des linearisierten Lotka-Volterra Modells im
212     Gleichgewichtspunkt G2.
213     """
214
215     # Festlegen der Eigenwerte zur Loesung der linearisierung um
215     # G2.
216     lambda_1 = 1j*sqrt(a*d)
217     lambda_2 = -1.0 * lambda_1
218
219     # Festlegen der Eigenvektoren zur Loesung der linearisierung
219     # um G2.
220     v_11 = (d/c*b) / lambda_2
221     v_12 = (d/c*b) / lambda_1
222
223     # Festlegen von S und berechne die Inverse von S und
223     # definiere als S_1.
224     S = np.array([[v_11,v_12],[1,1]])
225     S_1 = np.linalg.inv(S)
226
227     # Aufgrund der Verschiebung in den Ursprung fuer die
227     # Berechnung der Matrix bei der Linearisierung muss fuer
227     # die Berechnung der tatsaechliche Anfangswert um den
227     # Gleichgewichtspunkt verschoben werden.
228     Anf = np.array(Anf)-np.array([d/c,a/b])
229
230     # Berechne c_1 = inv(S)*Anf
231     c_1 = S_1.dot(Anf)
232

```



```

269     pl.ylabel(r"$Populationsgr\"o\ss{}e$")
270     plt.axhline(color="0.5")
271     plt.axvline(color="0.5")
272     pl.grid(True)
273
274     # Zeichne die Rauber- und Beutepopulationen in Abhaengigkeit
275     # von der Zeit.
276     pl.plot(t_1,P_1,label=r"$R\$auber$",color="r")
277     pl.plot(t_1,N_1,label=r"$Beute$",color = "b")
278
279     # Speichern des Aufrufkommandos fuer die Legende und
280     # darstellen der gewaehlten Parameter oberhalb der Grafik,
281     # sowie speichern und anzeigen der Grafik.
282     lgd = plt.legend(loc="upper right", fancybox=True,frameon=
283     True, shadow=True, bbox_to_anchor=(1.1, 1.1), fontsize=13)
284     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 03.1f, \; d=% 3.1f$"
285     "%(a,b,c,d), fontsize=15)
286     pl.savefig('g2_time.pdf', bbox_extra_artists=(lgd, ),
287     bbox_inches='tight')
288     pl.show()
289
290     #####
291     # Grafik in der Phasenebene
292     #####
293
294     # Berechne die Loesungen fuer Anf, b_2 und b_3.
295     N,P,t = solution_G2(a,b,c,d,Anf)
296     N_2,P_2,t_2 = solution_G2(a,b,c,d,b_2)
297     N_3, P_3,t_3 = solution_G2(a,b,c,d,b_3)
298
299     # Bestimmen des Formats der Grafik.
300     plotstyleset()
301     pl.xlim([-4,8])
302
303     # Moeglichkeit Pfeilspitzen ans "Ende" der Achsen zu setzen.
304     # pl.arrow(0,0, 5.6, 0, fc="0.75", ec="0.75", head_width=0.3,
305     #         head_length=0.5 ,color="0.75")
306     # pl.arrow( 0, 0, 0, 7.6, fc="0.75", ec="0.75", head_width
307     #         =0.3, head_length=0.5 ,color="0.75")
308
309     # Zeichne die 3 Loesungen ein.
310     pl.plot(N, P, label = r"$b = (1,1)$",color="b")
311     pl.plot(N_2,P_2, label = r"$b = (5,2)$",color="g")
312     pl.plot(N_3,P_3, label = r"$b = (2,2)$",color="r")
313
314     # Einzeichnen des Gleichgewichtspunkts.
315     pl.plot([0.9/1.0],[2/1.2],"co",label=r"$GG$-Punkt$")

```

```

308
309     # Einzeichnen der Pfeile in die jeweiligen Loesungengraphen
310     # zur Darstellung der Veraenderungsrichtung der
311     # Populationsgroessen ueber die Zeit.
312
313     pl.arrow( N[101] + 0.25, P[101] +0.1, N[101]-N[100], P[101]-P
314         [100], fc="b", ec="b",      head_width=0.3, head_length
315         =0.4 ,color="b")
316
317     pl.arrow( N_2[0] -0.01, P_2[0] -0.01, N_2[1]-N_2[0], P_2[1]-
318         P_2[0], fc="g", ec="g",      head_width=0.3, head_length
319         =0.4,color="g" )
320
321     pl.arrow( N_3[0] -0.01, P_3[0] -0.01, N_3[1]-N_3[0], P_3[1]-
322         P_3[0], fc="r", ec="r",      head_width=0.3, head_length
323         =0.4,color="r" )
324
325
326     # Speichern des Aufrufkommandos fuer die Legende und
327     # darstellen der gewaehlten Parameter oberhalb der Grafik,
328     # sowie speichern und anzeigen der Grafik.
329
330     lgd = plt.legend(loc="upper right", fancybox=True,frameon=
331         True,shadow=True,bbox_to_anchor=(1.15, 1.1),fontsize=13)
332
333     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f$%"(
334         a,b,c,d),fontsize=15)
335
336     pl.savefig("g_2.pdf", bbox_extra_artists=(lgd,), bbox_inches=
337         'tight')
338
339     pl.show()

```

321 def plot_G1(a,b,c,d):
322 """
323 Zeichnet die Loesungen der Linearisierung um G2 in die
324 Phasenebene, sowie als Veraenderung ueber die Zeit zu
325 gegebenen Parametern.
326 """
327
328 # Initialisiere 4 Anfangswerte: Anf, b_1, b_2, b_3
329 Anf = np.array([1,1])
330 b_2 = np.array([5,2])
331 b_3 = np.array([2,2])
332 b_1 = np.array([1,1])
333
334 #####
335 # Grafik fuer Darstellung der Loesung entlang der Zeitachse
336 #####
337
338 # Berechnung der Loesung fuer den Anfangswert b_1.
339 N_1,P_1,t_1 = solution_G1(2,1.2,1,0.9,b_1)

```

340     pl.xlabel(r"$Zeit$")
341     pl.ylabel(r"$Populationsgr\"o\ss{}e$")
342     pl.ylim([0,20])
343     plt.yticks(np.arange(0, 20, 1.0))
344     pl.grid(True)
345     plt.axhline(color="0.75")
346     plt.axvline(color="0.75")
347
348     # Zeichne die Rauber- und Beutepopulationen in Abhaengigkeit
349     # von der Zeit.
350     pl.plot(t_1[101:202],P_1[101:202],label=r"$R\$auber",color =
351     "r")
352     pl.plot(t_1[101:202],N_1[101:202],label=r"$Beute$",color = "b"
353     )
354
355     # Speichern des Aufrufkommandos fuer die Legende und
356     # darstellen der gewaehlten Parameter oberhalb der Grafik,
357     # sowie speichern und anzeigen der Grafik.
358     lgd = plt.legend(loc="upper right", fancybox=True,frameon=
359     True, shadow=True, bbox_to_anchor=(1.1, 1.1), fontsize=13)
360     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f\$%"(
361     a,b,c,d), fontsize=15)
362     pl.savefig('g1_time.pdf', bbox_extra_artists=(lgd, ),
363     bbox_inches='tight')
364     pl.show()
365
366
367     #####
368     # Grafik in der Phasenebene
369     #####
370
371
372     # Berechne die Loesungen fuer Anf, b_2 und b_3.
373     N,P,t = solution_G1(a,b,c,d,Anf)
374     N_2,P_2,t_2 = solution_G1(a,b,c,d,b_2)
375     N_3, P_3,t_3 = solution_G1(a,b,c,d,b_3)
376
377
378     # Bestimmen des Formats der Grafik.
379     plotstyleset()
380     pl.xlim([0,20])
381     pl.ylim([0,10])
382     plt.yticks(np.arange(0, 11, 1.0))
383     plt.xticks(np.arange(0, 20, 1.0))
384
385     # Zeichne die 3 Loesungen ein.
386     pl.plot(N, P, label = r"$b = (1,1)$",color="b")
387     pl.plot(N_2,P_2, label = r"$b = (5,2)$",color="g")
388     pl.plot(N_3,P_3, label = r"$b = (2,2)$",color="r")

```



```

411      # Bestimme maxi als maximalen Wert fuer GG_points und
412      # GG_points_vectors.
413      maxi = 2.5
414
415      # Aufruf der Funktionen, welche fuer die gegebenen Parameter
416      # ihre speziellen Graphen zeichnen und speichern
417      plot_G2(a,b,c,d)
418      plot_G1(a,b,c,d)
419      GG_points(a,b,c,d,maxi)
420      GG_points_vectors(a,b,c,d,maxi)
421
422      # Aufrufen der Funktion zum Zeichnen des Vektorfelds fuer das
423      # Lotka-Volterra Modell und gegebenen Parametern.
424      # Speichere den Aufruf zum Zeichnen der Legende in lgd
425      # Waehlte diesen Weg um unterschiedliche Skalierungen in
426      # unterschiedlichen Dateien speichern zu koennen.
427      lgd = vectorfield(a,b,c,d,maxi,0.05)
428      # Speichere und zeige die Grafik.
429      plt.savefig("vectorfield_lotvolt.pdf", bbox_extra_artists=(
430          lgd,), bbox_inches='tight')
431      plt.show(lgd)

```

D.1.2 3D-Grafiken und Trajektorie

```

1 #-----
2 # Zeichne 3D-Grafiken und Trajektorie fuer das Lotka-Volterra Modell.
3 #-----
4 # Importieren der noetigen python-libraries.
5 from mpl_toolkits.mplot3d import Axes3D
6 import matplotlib.pyplot as plt
7 from matplotlib import cm
8 import matplotlib as mpl
9 import numpy as np
10 from scipy import linalg
11 import pylab as pl
12 import seaborn
13
14 def fig_3d(N,P,E,c_2,par):
15     """
16         Zeichnet und speichert 3d-Plots der Funktion E mit der
17         allgemeinen Konstante c_2 aus vier verschiedenen
18         Blickwinkeln zu gegebenen Parametern par.
19
20         """
21
22         # Addiere zu dem gegebenen E(N,P) die Allgemeine Konstante,
23         # welche aus dem Anfangswert berechnet wurde, um die E(N,P)
24         # -Funktion auf die Richtige Hoehe zu positionieren.
25         E = E + c_2

```

```

21
22     # Festlegen des Formats der Grafik.
23     fig_1 = pl.figure()
24     seaborn.set_style("whitegrid")
25
26     # Starte eine Untergrafik mit der Projektion 3D, kreiere also
27     # ein 3d-Projekt.
28     bx = fig_1.add_subplot(111, projection='3d')
29     bx.set_ylabel(r'$R\backslash auber$', fontsize=18)
30     bx.set_xlabel(r'$Beute$', fontsize=18)
31     bx.set_zlabel(r'$E(N,P)$', fontsize=18)
32     bx.auto_scale_xyz([0, 15], [0, 15], [0, 20])
33
34     # Finde die maximalen und minimalen Werte der E(N,P)-Funktion
35     # zum festlegen der Randwerte fuer die Farbcodierung.
36     chmax = npamax(E)
37     chmin = npamin(E)
38
39     # Festlegen der Randwerte fuer die Normalisierung fuer die
40     # Farbcodierung in Norm
41     norm = mpl.colors.Normalize(vmin=chmin+ 1/3*(chmax - chmin),
42                                 vmax=chmax+1/10*(chmax-chmin))
43
44     # Zeichne eine Oberflaechengrafik um die 3D-Eigenschaften von
45     # E(N,P) darzustellen.
46     surf = bx.plot_surface(N, P, E, rstride=5, cstride=5, cmap='hot',
47                           norm=norm, linewidth=0, antialiased=False)
48
49     # Initialisiere einen Array um 4 verschiedenen Blickwinkel
50     # erstellen zu koennen.
51     view = np.array([-128]*4)
52
53     # Nutze die unten stehende if-else-Entscheidung um 4
54     # Blickwindel in je 90 Grad Winkel.
55     for i in range(len(view)):
56         view[i] += 90*i
57         if view[i] > 0:
58             view[i] -= 360
59
60     # Erhalte die Parameter aus den gegebenen und schreibe sie
61     # oberhalb die Grafik. Reduziere ausserdem die Raender der
62     # erstellen Grafiken.
63     a,b,c,d = par
64     bx.set_title(r'$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f$'
65                  '%(a,b,c,d)', fontsize=18)
66     fig_1.tight_layout()
67
68     # Erstelle die Grafiken aus 4 Verschiedenen Blickwinkeln und

```

```

        speichere sie.

57     for i in view:
58         bx.view_init(elev=11,azim=i)
59         pl.savefig("view%i_ed.png"%i,format='png',dpi=200)
60
61 def fig_3d_big(N,P,E,c_2,par):
62     """
63     Zeichnet und speichert einen 3d-Plot der Funktion E mit der
64     allgemeinen Konstante c_2 zu gegebenen Parametern par.
65     """
66
67     # Addiere zu dem gegebenen E(N,P) die Allgemeine Konstante,
68     # welche aus dem Anfangswert berechnet wurde, um die E(N,P)
69     # -Funktion auf die Richtige Hoehe zu positionieren.
70     E = E + c_2
71
72
73     # Festlegen des Formats der Grafik.
74     fig_1 = pl.figure()
75     seaborn.set_style("whitegrid")
76
77     # Starte eine Untergrafik mit der Projektion 3D, kreiere also
78     # ein 3d-Projekt.
79     bx = fig_1.add_subplot(111, projection='3d')
80     bx.set_ylabel(r'$R\$auber$')
81     bx.set_xlabel(r'$Beute$')
82     bx.set_zlabel(r'$E(N,P)$')
83     bx.auto_scale_xyz([0, 15], [0, 15], [0, 20])
84
85
86     # Finde die maximalen und minimalen Werte der E(N,P)-Funktion
87     # zum festlegen der Randwerte fuer die Farbcodierung.
88     chmax = npamax(E)
89     chmin = npamin(E)
90
91     # Festlegen der Randwerte fuer die Normalisierung fuer die
92     # Farbcodierung in Norm
93     norm = mpl.colors.Normalize(vmin=chmin+ 1/3*(chmax - chmin),
94                                 vmax=chmax+1/10*(chmax-chmin))
95
96     # Zeichne eine Oberflaechengrafik um die 3D-Eigenschaften von
97     # E(N,P) darzustellen.
98     surf = bx.plot_surface(N, P, E, rstride=5, cstride=5, cmap='hot',
99                            norm=norm, linewidth=0, antialiased=False)
100
101    # Erhalte die Parameter aus den gegebenen und schreibe sie
102    # oberhalb die Grafik. Reduziere außerdem die Raender der
103    # erstellen Grafiken.

```



```

130     # Setze die Farbcodierung rechts von der Grafik.
131     fig_2.colorbar(cset1, format=".1f", label="Wert der
132         Allgemeinen Konstante")
133
134     # Schreibe die verwendeten Parameter ueber die Grafik.
135     a,b,c,d = par
136     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f$%"(
137         a,b,c,d), fontsize=15)
138
139     # Speichere und zeige die Grafik.
140     pl.savefig("Trajektorie.pdf")
141     pl.show()
142
143     def params(a,b,c,d,Anf):
144         """
145             Die Funktion nimmt die Parameter und den Anfangswert des
146                 Lotka-Volterra Modells und gibt 2 meshgrids fuer die
147                 Werte von N und P zurueck, sowie E(N,P).
148             Als Zusatz wurde die Allgemeine Konstante fuer E(N,P)
149                 berechnet, und zusammen mit den Parametern als Tupel und
150                 oben genannten Objekten zurueckgegeben.
151         """
152
153         # Berechne die allgemeine Konstante fuer E(N,P)
154         c_2 = - a * np.log(Anf[1])+b * Anf[1] + c*Anf[0] - d * np.log
155             (Anf[0])
156
157         # Drucke den Wert der Allgemeinen Konstante in der Konsole
158         print("Allgemeine Konsante:", c_2)
159
160         # Initialisiere ein numpy-Array fuer die x- und y-Achse und
161             baue aus letzterem ein Netzwerk aus Punkten zur Berechnung
162                 der Richtungsvektoren in jedem Punkt dieses Netzwerkes.
163
164         # Speichere dies zwei mal als je eine Koordinaten-Matrix.
165         N = np.arange(0.001,1,0.001)
166         N = np.append(N, np.arange(1.1, 100, 0.2))
167         P = np.arange(0.001,1,0.001)
168         P = np.append(P, np.arange(1.1, 30, 0.2))
169         N, P = np.meshgrid(N, P)
170
171         # Berechne die einzelnen Terme fuer und im Anschluss E(N,P)
172             selbst.
173         N_1 = np.log(N)*d - c*N
174         P_1 = np.log(P)*a - b*P
175         E = P_1+N_1
176
177         # Gebe fuer das Zeichnen der Grafik die Matrizen N und P und
178             E zurueck, sowie die allgemeine Konstante c_2 und die

```

```

        Parameter als Tupel.
166     return N,P,E,c_2,(a,b,c,d)
167
168 if __name__ == '__main__':
169     # Berechne E(N,P) und alle wichtigen zugehoerigen Parameter
170     # und uebergebe sie in die Grafikzeichefunktionen fig_3d
171     # und Trajektorie.
172     N,P,E,c_2,par = params(5,1,0.3,4,[3,3])
173
174     fig_3d(N,P,E,c_2,par)
175     fig_3d_big(N,P,E,c_2,par)
176     Trajektorie(N,P,E,c_2,par)

```

D.2 Logistisches Lotka-Volterra Modell

```

1 #-----
2 # Zeichne Grafiken der Loesungen fuer G_1 und G_2 des Logistischen
3 #-----#
4 # Importieren der noetigen python-libraries.
5 import numpy as np
6 from scipy import linalg
7 import pylab as pl
8 import matplotlib.pyplot as plt
9 import seaborn
10 import cmath as cm
11 from math import *
12 from matplotlib import rcParams
13 import matplotlib as mpl
14 from scipy.integrate import odeint
15
16
17 def GG_points(a,b,c,d,K):
18     """
19         Zeichnet und speichert eine Grafik der Gleichgewichtsgeraden
20         fuer gegebene Parameter.
21         Nutze K ebenfalls als Maximalwert fuer die Grafik.
22     """
23
24     # Initialisiere numpy-Arrays fuer x- und y-Achse.
25     N_1 = pl.arange(0,K+0.1,K*0.05)
26     P_1 = pl.arange(0,K+2,K*0.05)
27
28     # Bestimme die Gleichgewichtsgerade fuer G2.
29     G_2 = a/b *(1- N_1 /K)
30
31     # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
32     GG_x = np.array([0,d/c,K])

```

```

32     GG_y_2 = a/b*(1-(d/c)/K)
33     GG_y = np.array([0,GG_y_2,0])
34
35     # Festlegen des Formats der Grafik.
36     fig = plt.figure()
37     rcParams['xtick.labelsize'] = 12
38     rcParams['ytick.labelsize'] = 12
39     rcParams.update({'font.size': 15})
40     plt.xlabel(r'$Beute$', fontsize=15)
41     plt.ylabel(r'$R\backslash auber$', fontsize=15)
42     plt.grid(True)
43     plt.ylim(-.5,K+1)
44     plt.xlim(-.5,K+1)
45
46     # Zeichne die 4 Gleichgewichtsgeraden und speichere Namen
47     # fuer Legende.
48     plt.plot([d/c]*len(P_1),P_1,color='m', label=r'$\frac{d}{c}x$')
49     plt.plot(N_1,G_2, color="c",label=r'$P(t) = \frac{a}{b+e^{-(N(t)-cK)}}$')
50     plt.axhline(color="b",label=r'$P(t) = 0$')
51     plt.axvline(color="r",label=r'$N(t) = 0$')
52
53     # Zeichne und beschreife die Schnittpunkte.
54     plt.plot(GG_x,GG_y,marker="x",color="k",linestyle="",mec="k",
55               ms=10,mew=1,label=r'$GG$-Punkt')
56     plt.text(GG_x[0]+0.02*K,GG_y[0]+0.02*K,'$G_1$',fontsize=15)
57     plt.text(GG_x[1]+0.02*K,GG_y[1]+0.02*K,'$G_2$',fontsize=15)
58     plt.text(GG_x[2]+0.02*K,GG_y[2]+0.02*K,'$G_3$',fontsize=15)
59
60     # Zeichnen der Legende und schreibe die Parameter ueber die
61     # Grafik.
62     plt.legend(loc="upper right", fancybox=True,frameon=True,
63                shadow=True,bbox_to_anchor=(1.05, 1.0),fontsize=13)
64     plt.title(r"$a=% 3.1f, b=% 3.1f, c=% 3.1f, d=% 3.1f, K=%i$"%(a,b,c,d,K),fontsize=15)
65
66 def GG_points_vectors(a,b,c,d,K):
67     """
68     Zeichnet und speichert eine Grafik der Gleichgewichtsgeraden
69     des Logistischen-Modells.
70     Die durch die Gleichgewichtsgeraden entstehenden 4 Felder
71     enthalten zusaetglich je eine Gruppe von 2 Vektoren,

```

```

    welche die Wachstumsrichtung der Beute- bzw.
    Raeuberpopulationen anzeigt.

70   An den Gleichgewichtsgeraden selbst wird pro Feld je 1 Vektor
        gezeichnet, welcher die Wachstumsrichtung der Population
        zeigt, welche nicht im Gleichgewicht ist.

71   """
72
73   # Initialisiere numpy-Arrays fuer x- und y-Achse.
74   N_1 = pl.arange(0,K+0.1,K*0.05)
75   P_1 = pl.arange(0,K+2,K*0.05)

76
77   # Bestimme die Gleichgewichtsgerade fuer G2.
78   G_2 = a/b * (1- N_1 /K)

79
80   # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
81   GG_x = np.array([0,d/c,K])
82   GG_y_2 = a/b*(1-(d/c)/K)
83   GG_y = np.array([0,GG_y_2,0])

84
85   # Festlegen des Formats der Grafik.
86   fig = plt.figure()
87   rcParams['xtick.labelsize'] = 12
88   rcParams['ytick.labelsize'] = 12
89   rcParams.update({'font.size': 15})
90   plt.xlabel(r'$Beute$', fontsize=15)
91   plt.ylabel(r'$R\$auber$', fontsize=15)
92   plt.grid(True)
93   plt.ylim(-.5,K+1)
94   plt.xlim(-.5,K+1)

95
96   # Zeichne die 4 Gleichgewichtsgeraden und speichere Name fuer
        Legende.
97   plt.plot([d/c]*len(P_1),P_1,color='m', label=r'$\frac{d}{c}$')
98   plt.plot(N_1,G_2, color="c",label=r'$P_1(t) = \frac{a}{b(1-\frac{N(t)}{K})}$')
99   plt.axhline(color="b",label=r'$P(t) = 0$')
100  plt.axvline(color="r",label=r'$N(t) = 0$')

101
102  # Zeichne und beschrifte die Schnittpunkte.
103  plt.plot(GG_x,GG_y,marker="x",color="k",linestyle="",mec="k",
        ms=10,mew=1,label=r'$GG$-Punkt')
104  plt.text(GG_x[0]+0.02*K,GG_y[0]+0.02*K,'$G_1$',fontsize=15)
105  plt.text(GG_x[1]+0.02*K,GG_y[1]+0.02*K,'$G_2$',fontsize=15)
106  plt.text(GG_x[2]+0.02*K,GG_y[2]+0.02*K,'$G_3$',fontsize=15)
107
108

```

```

109 ## Zeichne die jeweiligen Vektoren in jedes der 4 Felder.
110
111 # Unteres rechtes Feld.
112 pl.arrow(15,2,1, 0, fc="k", ec="k",head_width=0.3,
113     head_length=0.4,color="k" )
114 pl.arrow(15,2,0, 1, fc="k", ec="k",head_width=0.3,
115     head_length=0.4,color="k" )
116
117 # Unteres linkes Feld.
118 pl.arrow(5,6.5,1, 0, fc="k", ec="k",head_width=0.3,
119     head_length=0.4,color="k" )
120 pl.arrow(5,6.5,0, -1, fc="k", ec="k",head_width=0.3,
121     head_length=0.4,color="k" )
122 pl.arrow(5,15,0, -1, fc="k", ec="k",head_width=0.3,
123     head_length=0.4,color="k" )
124
125 # Oberes linkes Feld.
126 pl.arrow(17,12,-1, 0, fc="k", ec="k",head_width=0.3,
127     head_length=0.4,color="k" )
128 pl.arrow(17,12,0, 1, fc="k", ec="k",head_width=0.3,
129     head_length=0.4,color="k" )
130 pl.arrow(17,a/b *(1-17/K),0, 1, fc="k", ec="k",head_width
131     =0.3, head_length=0.4,color="k" )
132
133 # Zeichnen der Legende und schreibe die Parameter ueber die
134 # Grafik.
135 lgd = plt.legend(loc="upper right", fancybox=True,frameon=
136     True,shadow=True,bbox_to_anchor=(1.3, 1.1),fontsize=13)
137 plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \;
138 \; K=%i$% (a,b,c,d,K), fontsize=15)
139
140 # Speichere die Grafik und zeige sie direkt an.
141 plt.savefig("GG_points_vectors.pdf", bbox_extra_artists=(lgd
142     ,), bbox_inches='tight')
143 plt.show()

```

```

140 def streamfield(a,b,c,d,K):
141     """
142         Erstellt ein Stromlinienfeld fuer die gegebenen Parameter
143             fuer das logistische Rauber Beute Modell.
144     """
145     # Initialisiere ein numpy-Array fuer die x- und y-Achse und
146         # baue aus letzterem ein Netzwerk aus Punkten zur Berechnung
147             # der Richtungsvektoren in jedem Punkt dieses Netzwerkes.
148     # Speichere dies zwei mal als je eine Koordinaten-Matrix.
149     N_1 = pl.arange(0, K+0.02*K, K*0.05)
150     P_1 = pl.arange(0, K+0.2*K, K*0.05)
151     N,P = pl.meshgrid(N_1, N_1)
152
153
154     # Berechne die Steigung der Beute als U und die der Rauber
155         # als V.
156     U = N*(a*(1-N /K) -b*P)
157     V = P*(c*N-d)
158
159     # Bestimme die Gleichgewichtsgerade fuer G2.
160     G_2 = a/b * (1- N_1/K)
161
162     # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
163     GG_x = np.array([0,d/c,K])
164     GG_y_2 = a/b*(1-(d/c)/K)
165     GG_y = np.array([0,GG_y_2,0])
166
167
168     # Festlegen des Formats der Grafik.
169     pl.figure()
170     plotstyleset()
171     plt.ylim(0,K+0.1)
172     plt.xlim(0,K+0.1)
173
174     # Zeichne die 2 Gleichgewichtsgeraden und halte die Option
175         # offen die letzten 2 hinzuzufuegen. Sie sind
176             # auskommentiert.
177     plt.plot([d/c]*len(P_1),P_1,color='m',zorder=1, label=r"\$\
178             frac{3.1f}{3.1f}\$%(d,c)")
179     plt.plot(N_1,G_2, color="c",zorder=2,label=r"\$frac{3.1f}{3.1f}\$\
180             cdot\left(\frac{N(t)}{3.1f}\cdot i\right)\$%(a,b,c,K)")
181     #plt.axhline(color="b",label=r'$P(t) = 0$',zorder=3)
182     #plt.axvline(color="r",label=r'$N(t) = 0$',zorder=4)
183
184     # Zeichne und beschrifte die Schnittpunkte.
185     plt.plot(GG_x,GG_y,marker="x",color="k",linestyle=" ",mec="k",
186             ms=10,mew=1,label=r"$GG$-Punkt",zorder=5)
187     plt.text(GG_x[0]+0.02*K,GG_y[0]+0.02*K,'$G_1$',fontsize=15,

```

```

        zorder=11)
177     plt.text(GG_x[1]+0.02*K,GG_y[1]+0.02*K,'$G_2$',fontsize=15,
        zorder=11)
178     plt.text(GG_x[2]+0.02*K,GG_y[2]+0.02*K,'$G_3$',fontsize=15,
        zorder=11)
179
180     # Definiere die Variable "norm", welche den maximalen und
        minimalen Wert fuer die Farbcodierung der
        Steigungsintensitaet im Stromlinienfeld festzulegen.
181     norm = mpl.colors.Normalize(vmin=np.min(U)-1/10*(np.max(U)-np
        .min(U)), vmax=np.max(U)+2/5*(np.max(U)-np.min(U)))
182
183     # Nutze die Funktion streamplot um das Stromlinienfeld zu
        zeichnen.
184     Q = pl.streamplot(N,P,U, V,zorder=10,color=U, linewidth=2,
        cmap=plt.cm.hot,norm=norm) #,angles='xy', scale_units='xy
        ', scale=1/0.1,width=0.0001*K) #,scale=1/0.002)
185
186     # Hinzufuegen der Farbcodierung rechts der Grafik
187     plt.colorbar()
188
189     # Verschiebe die ganze Grafik um 0.05 nach rechts oben fuer
        uebersichtlichere und ansprechendere Grafik.
190     l,r,b1,t = pl.axis()
191     dx, dy = r-l, t-b1
192     pl.axis([l-0.05*dx, r+0.05*dx, b1-0.05*dy, t+0.05*dy])
193
194     # Zeichnen der Legende und schreibe die Parameter ueber die
        Grafik.
195     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \;
        K=%i$%(a,b,c,d,K),fontsize=15)
196     lgd = plt.legend(loc="upper right", fancybox=True, frameon=
        True, shadow=True, bbox_to_anchor=(1.6, 1.1), fontsize=13)
197
198     # Behalte die Grafik in Speicher auch ausserhalb der Funktion
        und gebe den Aufruf zum Zeichnen der Legende zurueck.
199     plt.hold(True)
200     return(lgd)
201
202 def vectorfield(a,b,c,d,K,scalar):
203     """
204         Zeichnet ein Vektorfeld fuer die gegebenen Parameter und
        nutze wieder K als maximalen Wert.
205         "scalar" ist ein Skalar, mit welchem die Skalierung der
        Vektorlaengen festgelegt wird.
206         Die Funktion gibt den Aufruf zum Zeichnen der Legende zurueck
        und haelt die Grafik im Speicher ausserhalb des

```

```

    Funktionsspeichers.

207    """
208
209    # Initialisiere ein numpy-Array fuer die x- und y-Achse und
210    # baue aus letztem ein Netzwerk aus Punkten zur Berechnung
211    # der Richtungsvektoren in jedem Punkt dieses Netzwerkes.
212    # Speichere dies zwei mal als je eine Koordinaten-Matrix.
213    N_1 = pl.arange(0,K+0.02*K,K*0.05)
214    P_1 = pl.arange(0,K+0.2*K,K*0.05)
215    N,P = pl.meshgrid( N_1, N_1)
216    # Festlegen der Gleichgewichtsgeraden fuer G2.
217    G_2 = a/b *(1- N_1/K)
218
219
220
221    # Berechne die Steigung der Beute als U und die der Raeuber
222    # als V.
223    U = N*(a*(1-N /K)-b*P)
224    V = P*(c*N-d)
225
226    # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
227    GG_x = np.array([0,d/c,K])
228    GG_y_2 = a/b*(1-(d/c)/K)
229    GG_y = np.array([0,GG_y_2,0])
230
231
232    # Festlegen des Formats der Grafik.
233    pl.figure()
234    plotstyleset()
235    plt.ylim(0,K+0.1)
236    plt.xlim(0,K+0.1)
237
238    # Verschiebe die ganze Grafik um 0.05 nach rechts oben fuer
239    # uebersichtlichere und ansprechendere Grafik.
240    l,r,rb,t = pl.axis()
241    dx, dy = r-l, t-rb
242    pl.axis([l-0.05*dx, r+0.05*dx, rb-0.05*dy, t+0.05*dy])
243
244    # Zeichne die 4 Gleichgewichtsgeraden und speichere Name fuer
245    # Legende.
246    plt.plot([d/c]*len(P_1),P_1,color='m',zorder=1, label=r"\$"
247             frac{d}{c}\$")
248    plt.plot(N_1,G_2, color="c",zorder=2,label=r"\$P(t) = \frac{"
249             a}{b}\cdot\left(1-\frac{N(t)}{c}\right)\$")
250    plt.axhline(color="b",label=r"\$P(t) = 0\$,zorder=3)
251    plt.axvline(color="r",label=r"\$N(t) = 0\$,zorder=4)
252
253    # Zeichne die Schnittpunkte der Gleichgewichtsgeraden
254    plt.plot(GG_x,GG_y,marker="x",color="k",linestyle="",mec="k",
255             ms=10,mew=1,label=r"\$GG\text{-Punkt}\$",zorder=5)

```

```

245
246     # Die Beschriftung der Schnittpunkte wurde hier
247     # auskommentiert, aber nicht geloescht, um sie jederzeit
248     # wieder einfuegen zu koennen.
249     # plt.text(GG_x[0]+0.02*K,GG_y[0]+0.02*K,'$G_1$',fontsize=15)
250     # plt.text(GG_x[1]+0.02*K,GG_y[1]+0.02*K,'$G_2$',fontsize=15)
251     # plt.text(GG_x[2]+0.02*K,GG_y[2]+0.02*K,'$G_3$',fontsize=15)
252
253     # Benutze die quiver-Funktion zum Zeichnen der Vektoren in
254     # jedem Punkt des Netzwerks mithilfe der Matrizen U und V.
255     Q = plt.quiver(N,P,U, V, width=0.0001*K, zorder=10, angles='xy',
256                     scale_units='xy', scale=1/scalar)#,scale=1/0.002)
257
258     # Zeichnen und speichern der Legende in lgd und schreibe die
259     # Parameter ueber die Grafik
260     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \; "
261               K=%i$%(a,b,c,d,K), fontsize=15)
262     lgd = plt.legend(loc="upper right", fancybox=True, frameon=
263                       True, shadow=True, bbox_to_anchor=(1.35, 1.1), fontsize=13)
264
265     # Behalte die Grafik in Speicher auch ausserhalb der Funktion
266     # und gebe den Aufruf zum Zeichnen der Legende zurueck.
267     plt.hold(True)
268     return(lgd)
269
270
271 def solution_G3(a,b,c,d,K,Anf):
272     """
273     Berechne die Loesung zum Anfangswert "Anf" mit den gegebenen
274     Parametern des linearisierten Logistischen Modells im
275     Gleichgewichtspunkt G3.
276     """
277     # Festlegen des Gleichgewichtspunktes G3 als G
278     G = np.array([K,0])
279
280     # Festlegen der Eigenwerte zur Loesung der linearisierung um
281     # G3.
282     lambda_1 = -a
283     lambda_2 = c*K-d
284
285     # Festlegen der Eigenvektoren zur Loesung der linearisierung
286     # um G3.
287     v_11 = 1
288     v_12 = 0
289
290     v_21 = -1

```

```

280         v_22 = (a+c*K-d) / (K*b)
281
282         # Festlegen von S und berechne die Inverse von S und
283         # definiere als S_1.
284         S = np.array([[v_11,v_21],[v_12,v_22]])
285         S_1 = np.linalg.inv(S)
286
287         # Aufgrund der Verschiebung in den Ursprung fuer die
288         # Berechnung der Matrix bei der Linearisierung muss fuer
289         # die Berechnung der tatsaechliche Anfangswert um den
290         # Gleichgewichtspunkt verschoben werden.
291         Anf = np.array(Anf) - G
292
293         # Berechne c_1 = inv(S)*Anf
294         c_1 = S_1.dot(Anf)
295
296         # Festlegen des Arrays der Zeitpunkte zu denen die Loesung
297         # berechnet und gezeichnet werden soll.
298         time = [i * 0.025 for i in range(401)]
299
300         # Initialisiere Arrays fuer die zu berechnenden
301         # Populationsgroessen der Raeuber und Beute. Stelle sicher,
302         # sie haben dieselbe Laenge, wie der Zeit-Array.
303         N = [0]*len(time)
304         P = [0]*len(time)
305
306         # Berechne die Loesungen fuer die Beute N und die Raeuber P
307         # und ersetze eine weitere 0 pro Iteration in den Arrays N
308         # und P.
309         for t_0,t in enumerate(time):
310             N[t_0] = c_1[0]*v_11*cm.exp(t*lambda_1) + c_1[1]*v_21
311                 *cm.exp(t*lambda_2) + G[0]
312             P[t_0] = c_1[0]*v_12*cm.exp(t*lambda_1) + c_1[1]*v_22
313                 *cm.exp(t*lambda_2) + G[1]
314
315         # Gebe die Arrays der Beutepopulation, Raeuberpopulation und
316         # Zeit zurueck, sodass die Funktion zum erstellen der
317         # Grafik die Loesungsfunktion aufrufen kann.
318         return N,P,time
319
320
321 def solution_G2(a,b,c,d,K,Anf,scalar):
322     """
323     Berechnet die Loesung zum Anfangswert "Anf" mit den gegebenen
324     Parametern des linearisierten Logistischen Modells im
325     Gleichgewichtspunkt G3.
326     """
327     # Festlegeb des Gleichgewichtspunkts G2 als G

```

```

312         G = np.array([d/c, a/b*(1-d/(c*K))])
313
314         ## Fallunterscheidung ob lambda reell oder imaginaer.
315         # Berechnen der zwei Terme, welche verglichen werden sollen.
316         left = a*d + 4*d*c*K
317         right = 4 * c**2 * K**2
318
319         # Vorbereiten von Einzelberechnungen fuer die
320             # Eigenwertberechnung
321         ad2cK = (-1)* a*d / (2*c*K)
322
323         # Fallunterscheidung: Falls Eigenwerte komplex
324         if left < right:
325             # Berechne M
326             M = sqrt((-1)*(a*d*(a*d - 4*c*K*(c*K-d))) / (2*c*K))
327
328             # Festlegen der Eigenwerte zur Loesung der
329                 # linearisierung um G2.
330             lambda_1 = ad2cK + 1j * M
331             lambda_2 = ad2cK - 1j * M
332
333             # Eigenvektor v_1
334             v_11 = 1
335             v_12 = (-1)*a/(2*b*K) - (1j*M*c)/(b*d)
336
337             # Eigenvektor v_2
338             v_21 = 1
339             v_22 = (-1)*a/(b*K) + (1j*M*c)/(2*b*d)
340             print("komplexe Eigenwerte")
341
342             # Fallunterscheidung: Falls Eigenwerte reell.
343             else:
344                 # Berechne M
345                 M = sqrt(a*d*(a*d - 4*c*K*(c*K-d))) / (2*c*K)
346
347                 # Festlegen der Eigenwerte zur Loesung der
348                     # linearisierung um G2.
349                 lambda_1 = ad2cK + M
350                 lambda_2 = ad2cK - M
351
352                 # Eigenvektor v_1
353                 v_11 = -1
354                 v_12 = (a/(2*b*K) + (M*c)/(b*d))
355
356                 # Eigenvektor v_2
357                 v_21 = 1
358                 v_22 = (-1)*a/(b*K) + (M*c)/(2*b*d)
359                 print("reelle Eigenwerte")

```

```

356
357     # Festlegen von S und berechne die Inverse von S und
358     # definiere als S_1.
359     S = np.array([[v_11,v_21],[v_12,v_22]])
360     S_1 = np.linalg.inv(S)
361
362     # Aufgrund der Verschiebung in den Ursprung fuer die
363     # Berechnung der Matrix bei der Linearisierung muss fuer
364     # die Berechnung der tatsaechliche Anfangswert um den
365     # Gleichgewichtspunkt verschoben werden.
366     Anf = np.array(Anf) - np.array([d/c,a/b*(1-d/(c*K))])
367
368     # Berechne c_1 = inv(S)*Anf
369     c_1 = S_1.dot(Anf)
370
371     # Festlegen des Arrays der Zeitpunkte zu denen die Loesung
372     # berechnet und gezeichnet werden soll.
373     time = [i * scalar for i in range(401)]
374
375     # Initialisiere Arrays fuer die zu berechnenden
376     # Populationsgroessen der Raeuber und Beute. Stelle sicher,
377     # sie haben dieselbe Laenge, wie der Zeit-Array.
378     N = [0]*len(time)
379     P = [0]*len(time)
380
381     # Berechne die Loesungen fuer die Beute N und die Raeuber P
382     # und ersetze eine weitere 0 pro Iteration in den Arrays N
383     # und P.
384     for t_0,t in enumerate(time):
385         N[t_0] = c_1[0]*v_11*cm.exp(t*lambda_1) + c_1[1]*v_21
386             *cm.exp(t*lambda_2) + d/c
387         P[t_0] = c_1[0]*v_12*cm.exp(t*lambda_1) + c_1[1]*v_22
388             *cm.exp(t*lambda_2) + a/b*(1-d/(c*K))
389
390     # Gebe die Arrays der Beutepopulation, Raeuberpopulation und
391     # Zeit zurueck, sodass die Funktion zum erstellen der
392     # Grafik die Loesungsfunktion aufrufen kann.
393     return N,P,time
394
395 def plotstyleset():
396     """
397     Standardformatierung fuer die Graphiken der Phasenebene.
398     """
399     plt.xlabel(r'$Beute$')
400     plt.ylabel(r'$R\$auber$')
401     plt.axhline(color="0.75")

```



```

429     N_2,P_2,t_2 = solution_G2(a,b,c,d,K,Anf1,scalar)
430     N_3, P_3,t_3 = solution_G2(a,b,c,d,K,Anf2,scalar)
431
432     # Festlegen des Gleichgewichtspunkts G2 als G
433     G = np.array([d/c,a/b*(1-d/(c*K))])
434
435     # Lege Grundlage fuer 2 Untergrafiken innerhalb einer Grafik.
436     fig = plt.figure(figsize=(10,5))
437
438     # Schreibe die Parameter ueber die Grafik.
439     plt.suptitle(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \; K=%i$"%(a,b,c,d,K), fontsize=15)
440
441     # Untergrafik 1
442     ax1 = fig.add_subplot(121) #,aspect="equal")
443     plotstyleset()
444     # Finde das Maximum des Maximalen x-Werts und Maximalen y-
445     # Werts und speichere diesen als "maxxy"
446     maxxy = np.max([maxx,maxy])
447
448     # Zeichne die Loesung und den Gleichgewichtspunkt ein.
449     first, = pl.plot(N_2,P_2,color="g") # label = "b = (5,2)",
450     third, = pl.plot(G[0],G[1],"co",ms=3.5) #,label="Gleichgewichtspunkt"
451     pl.arrow( N_2[35].real, P_2[35].real, N_2[36].real-N_2[35].real,
452             P_2[36].real-P_2[35].real, fc="g", ec="g", overhang=0,
453             head_width=0.1/6*maxxy, head_length=0.1/3*maxxy,color="g",
454             head_starts_at_zero = True, length_includes_head=True)
455     pl.ylim([miny,maxy])
456     pl.xlim([minx,maxx])
457
458     # Untergrafik 2
459     fig.add_subplot(122) #,aspect="equal")
460     plotstyleset()
461     pl.ylim([miny,maxy])
462     pl.xlim([minx,maxx])
463
464     # Zeichne die Loesung ein und den Gleichgewichtspunkt ein.
465     second, = pl.plot(N_3,P_3,color="r") # label = "b = (3,2)",
466     pl.plot(G[0],G[1],"co",ms=3.5) #,label="Gleichgewichtspunkt"
467
468     # Einzeichnen des Pfeils in den Loesungengraphen zur
469     # Darstellung der Veraenderungsrichtung der
470     # Populationsgroessen ueber die Zeit.
471     pl.arrow( N_3[15].real, P_3[15].real , N_3[16].real-N_3[15].real,
472             P_3[16].real-P_3[15].real, fc="r", ec="r",

```

```

        length_includes_head=True,head_width=0.1/6*maxxy,
        head_length=0.1/3*maxxy,color="r" )

466
467     # Zeichnen und speichern der Legende in lgd2
468     lgd2 = plt.figlegend([first, second, third],[r"$b = (5,2)$",r
469                           "$b = (3,1)$",r"GG-Punkt"],loc="upper right",frameon=
470                           True,fancybox=True,shadow=True,fontsize=13,bbox_to_anchor
471                           =(1, 1) ,borderaxespad=1.1)
472
473     # Behalte die Grafik in Speicher auch ausserhalb der Funktion
474     # und gebe den Aufruf zum Zeichnen der Legende zurueck.
475     plt.hold(True)
476     return lgd2

477
478
479     # Initialisiere 3 Anfangswerte: b_1, b_2, b_3.
480     b_1 = np.array([15,3])
481     b_2 = np.array([5,2])
482     b_3 = np.array([3,1])
483
484     # Festlegen der Parameter und der Kapazitaet K.
485     a = 2
486     b = 1.2
487     c = 0.7
488     d = 9.1
489     K = 10
490
491     # Berechnung der Loesungsarrays fuer alle Anfangswerte.
492     N,P,t = solution_G3(a,b,c,d,K,b_1)
493     N_2,P_2,t_2 = solution_G3(a,b,c,d,K,b_2)
494     N_3, P_3,t_3 = solution_G3(a,b,c,d,K,b_3)
495
496     ######
497     # Grafik fuer Darstellung der Loesung entlang der Zeitachse
498     ######
499     # Festlegen eines weiteren Anfangswertes
500     b_1 = np.array([3,1])
501
502     # Berechnung des Loesungsarrays fuer den Anfangswert b_1.
503     N_1,P_1,t_1 = solution_G3(a,b,c,d,K,b_1)
504

```

```

505     # Festlegen des Formats der Grafik.
506     pl.plot(t_1,P_1,label=r"$R\$ auber",color="r")
507     pl.plot(t_1,N_1,label=r"$Beute\$",color = "b")
508     pl.xlabel(r"$Zeit$")
509     pl.ylabel(r"$Populationsgr\"o\ss{}e$")
510     plt.axhline(color="0.5")
511     plt.axvline(color="0.5")
512     pl.grid(True)
513
514     # Speichern des Aufrufkommandos fuer die Legende und
515     # darstellen der gewaehlten Parameter oberhalb der Grafik,
516     # sowie speichern und anzeigen der Grafik.
517     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \; "
518               K=%i$%(a,b,c,d,K), fontsize=15)
519     lgd = plt.legend(loc="upper right", fancybox=True, frameon=
520                      True, shadow=True, bbox_to_anchor=(1.1, 0.99), fontsize=13)
521     pl.savefig('g3_time_log.pdf', bbox_extra_artists=(lgd,),
522                bbox_inches='tight')
523     pl.show()
524
525     ##### Grafik in der Phasenebene #####
526
527     # Festlegen des Gleichgewichtspunkts G2 als G
528     G = np.array([K,0])
529
530     # Festlegen des Formats der Grafik.
531     fig = plt.figure(figsize=(9,5))
532     plotstyleset()
533     pl.ylim([0,3.5])
534
535     # Zeichne die 3 Loesungen ein, jeweil mit den Pfeilen fuer
536     # die Richtungsangabe.
537     first, = pl.plot(N_2,P_2,color="g") # label = "b = (5,2)",
538     pl.arrow( N_2[5].real, P_2[5].real, N_2[6].real-N_2[5].real,
539               P_2[6].real-P_2[5].real, fc="g", ec="g", overhang=0,
540               head_width=0.1, head_length=0.2,color="g" ,
541               head_starts_at_zero = True, length_includes_head=True)
542     second, = pl.plot(N_3,P_3,color="r") # label = "b = (3,2)",
543     pl.arrow( N_3[5].real, P_3[5].real , N_3[6].real-N_3[5].real,
544               P_3[6].real-P_3[5].real, fc="r", ec="r",
545               length_includes_head=True,head_width=0.1, head_length
546               =0.2,color="r" )
547     forth, = pl.plot(N,P,color="b") # label = "b = (3,2)",

```

```

540     pl.arrow( N[5].real, P[5].real , N[6].real-N[5].real, P[6].
541                 real-P[5].real, fc="b", ec="b", length_includes_head=True,
542                 head_width=0.1, head_length=0.2,color="r" )
543
544     # Gleichgewichtspunkt
545     third, = pl.plot(G[0],G[1],marker="x",color="k",linestyle="",
546                         mec="k",ms=10,mew=1) #,ms=3.5) #,label="
547                         Gleichgewichtspunkt"
548
549     # Verschiebe die ganze Grafik um 0.05 nach rechts oben fuer
550     # uebersichtlichere und ansprechendere Grafik.
551     l,r,b1,t = pl.axis()
552     dx, dy = r-l, t-b1
553     pl.axis([l-0.05*dx, r+0.05*dx, b1-0.05*dy, t+0.05*dy])
554
555     # Speichern des Aufrufkommandos fuer die Legende und
556     # darstellen der gewaehlten Parameter oberhalb der Grafik,
557     # sowie speichern und anzeigen der Grafik.
558     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \;
559                 K=%i$\%(%a,b,c,d,K), fontsize=15)
560     lgd = plt.legend([first, second, forth, third],[r"$b = (5,2)$"
561                 ,r"$b = (3,1)$",r"$b = (15,3)$",r"$GG\text{-Punkt}$"], loc="upper
562                 right", frameon=True, fancybox=True, shadow=True,
563                 bbox_to_anchor=(1.2, 1.1), fontsize=13) #,bbox_to_anchor
564                 =(0.9, 0.8), borderaxespad=-1.)
565     pl.savefig("g_3_log_subs.pdf", bbox_extra_artists=(lgd,), ,
566                 bbox_inches='tight')
567
568     plt.show()
569
570
571
572
573
574
575
576
577 def dy(y,t_0,a,b,c,d,K):
578     """
579     Diese Funktion wurde geschrieben um die Loesung des
580     Differenzialgleichungssystems mit odeint() berechnen zu
581     koennen.
582
583     Berechne das Differenzialgleichungssystem des logistischen
584     Raeuber Beute Modells bei gegebener Loesung y.
585
586     """
587
588     # Initialisiere dy = (dN/dt, dP/dt)^T
589     dy = [0,0]
590
591     # Berechne dy mithilfe der Loesung und ersetze die 2
592     # Initialwerte.
593
594     dy[0] = y[0]*(a*(1-y[0]/K)-b*y[1])
595     dy[1] = y[1]*(c*y[0]-d)
596
597     # Gebe dy zurueck
598
599     return(dy)
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
999

```

```

570 def solver_time(a,b,c,d,K,y,time):
571     """
572         Zeichne die Loesung y mit den gegebenen Parametern des
573             Logistischen Modells ueber die Zeit.
574     """
575     # Festlegen des Formats der Grafik.
576     pl.xlabel(r"$Zeit$")
577     pl.ylabel(r"$Populationsgr\"o\ss{}e$")
578     plt.axhline(color="0.5")
579     plt.axvline(color="0.5")
580
581     # Zeichne die Raeuber- und Beutepopulationen in Abhaengigkeit
582         von der Zeit.
583     pl.plot(time,y[:,0],label=r"$Beute$",color = "b")
584     pl.plot(time,y[:,1],label=r"$R\"auber$",color="r")
585
586     # Zeichnen der Legende und schreibe die Parameter ueber die
587         Grafik.
588     plt.legend(fancybox=True,shadow=True,frameon=True)
589     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \; "
590                 K=%i$%(a,b,c,d,K), fontsize=15)
591     # Behalte die Grafik in Speicher auch ausserhalb der Funktion
592
593     .
594     plt.hold(True)
595
596 def solver_phaseplane(a,b,c,d,K,time):
597     """
598         Zeichne die Loesung y mit den gegebenen Parametern des
599             Logistischen Modells in der Phasenebene.
600     """
601
602     # Initialisiere numpy-Arrays fuer x- und y-Achse.
603     N_1 = pl.arange(0, K+0.02*K, K*0.05)
604     P_1 = pl.arange(0,K+0.2*K,K*0.05)
605
606     # Bestimme die Gleichgewichtsgerade fuer G2.
607     G_2 = a/b *(1- N_1/K)
608
609     # Festlegen der Schnittpunkte der Gleichgewichtsgeraden.
610     GG_x = np.array([0,d/c,K])
611     GG_y_2 = a/b*(1-(d/c)/K)
612     GG_y = np.array([0,GG_y_2,0])
613
614     # Berechne die Loesung des Logistischen Raeuber Beute Modells
615         fuer 4 verscheidene Anfangswerte.
616     y = odeint(dy,[10,15],time,args=(a,b,c,d,K))

```

```

610     y_1 = odeint(dy,[2,2],time,args=(a,b,c,d,K))
611     y_2 = odeint(dy,[19,18],time,args=(a,b,c,d,K))
612     y_3 = odeint(dy,[4,19],time,args=(a,b,c,d,K))
613
614     # Festlegen des Formats der Grafik.
615     plt.figure()
616     plotstyleset()
617     pl.xlabel(r'$Beute$')
618     pl.ylabel(r'$R\"auber$')
619     plt.xticks(np.arange(0, K+0.1, 1))
620     plt.yticks(np.arange(0, K+0.1, 2))
621     plt.ylim(0,K+0.1)
622     plt.xlim(0,K+0.1)
623     pl.grid(True)
624
625
626     # Zeichne die 2 Gleichgewichtsgeraden und speichere Name fuer
627     # Legende.
628     plt.plot([d/c]*len(P_1),P_1,color='m',zorder=1, label=r'$\frac{3.1f}{3.1f} \cdot \frac{3.1f}{3.1f} \cdot (d, c)$')
629
630     # Zeichne und beschrifte die Schnittpunkte.
631     plt.plot(GG_x,GG_y,marker="x",color="k",linestyle="",mec="k",
632     ms=10,mew=1,label=r'$GG$-Punkt$',zorder=5)
633     plt.text(GG_x[0]+0.02*K,GG_y[0]+0.02*K,'$G_1$',fontsize=15,
634     zorder=11)
635     plt.text(GG_x[1]+0.02*K,GG_y[1]+0.02*K,'$G_2$',fontsize=15,
636     zorder=11)
637     plt.text(GG_x[2]+0.02*K,GG_y[2]+0.02*K,'$G_3$',fontsize=15,
638     zorder=11)
639
640     # Zeichne die Loesungskurven in die Phasenebene
641     plt.plot(y[:,0],y[:,1],label=r'$Anf = (10,15)$',color="b",
642     zorder=10)
643     plt.plot(y_1[:,0],y_1[:,1],label=r'$Anf = (2,2)$',color="g",
644     zorder=10)
645     plt.plot(y_2[:,0],y_2[:,1],label=r'$Anf = (19,18)$',color="r",
646     zorder=10)
647     plt.plot(y_3[:,0],y_3[:,1],label=r'$Anf = (4,19)$',color="k",
648     zorder=10)
649
650     # Verschiebe die ganze Grafik um 0.05 nach rechts oben fuer
651     # uebersichtlichere und ansprechendere Grafik.
652
653     l,r,b1,t = pl.axis()

```

```

644     dx, dz = r-l, t-b
645     pl.axis([l-0.05*dx, r+0.05*dx, b1-0.05*dz, t+0.05*dz])
646
647
648     # Zeichnen und speichern der Legende in lgd und schreibe die
649     # Parameter ueber die Grafik.
650     lgd = plt.legend(loc="upper right", fancybox=True, frameon=
651     True, shadow=True, bbox_to_anchor=(1.35, 1.1), fontsize=13)
652     plt.title(r"$a=% 3.1f, \; b=% 3.1f, \; c=% 3.1f, \; d=% 3.1f, \;
653     K=%i$% (a,b,c,d,K), fontsize=15")
654
655     # Speichere die Grafik und zeige sie direkt an.
656     plt.savefig("phaseplane_log.pdf", bbox_extra_artists=(lgd,), 
657     bbox_inches='tight')
658     pl.show()
659
660
661
662     # Gruppe von Parametern 1
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679

```



```

719     plt.savefig("solver_time_(2,2).pdf")
720     plt.show()
721
722     plt.figure()
723     solver_time(a,b,c,d,K,y_2,time)
724     plt.savefig("solver_time_(19,18).pdf")
725     plt.show()
726
727     plt.figure()
728     solver_time(a,b,c,d,K,y_3,time)
729     plt.savefig("solver_time_(4,19).pdf")
730     plt.show()
731
732     # Nutze die Funktion solver_phaseplane um einen Graph in der
733     # Phasenebene zu erstellen.
734     solver_phaseplane(a,b,c,d,K,time)
735
736     # Nutze die Funktion streamfield um eine Stromlinienfeld fuer
737     # die oberen Parameter zu estellen, speichere es und zeige
738     # es direkt nach Ausfuehrung des Programms.
739     lgd = streamfield(a,b,c,d,K)
740     plt.savefig("streamfield_log.pdf", bbox_extra_artists=(lgd,), ,
741     bbox_inches='tight')
742     plt.show()
743
744     #
745
746
747
748
749
750
751     # Bestimme die Parameter, welche fuer die zweite Gruppe an
752     # Parametern genutzt werden soll.
753     a = 2
754     b = 1.2
755     c = 0.5
756     d = 0.9
757
758     K = 10
759
760
761     # Nutze Funktion plot_G2 zum zeichnen der Grafik, speichere
762     # sie und zeige die Grafik direkt im Anschluss an.
763     lgd1 = plot_G2(a,b,c,d,K,Anf,0.07)
764     plt.savefig('g2_time_log_(%i,%i).pdf'% (Anf[0],Anf[1]), ,
765     bbox_extra_artists=(lgd1,), bbox_inches='tight')
766     plt.show()

```

```

755
756     # Two subplots, unpack the axes array immediately
757     b_2 = np.array([5,2])
758     b_3 = np.array([3,1])
759
760     # Nutze Funktion plot_G2 zum zeichnen der Grafik auf der
761     # Phasenebene, speichere sie und zeige die Grafik direkt im
762     # Anschluss an.
763     lgd2 = phaseplane_G2(a,b,c,d,K,0.1,b_2,b_3,(6.1,3.1,-2.1,0))
764     plt.savefig("g_2_log_subs.pdf")#, bbox_extra_artists=(lgd2,),
765     plt.show()
766
767     #
768     # Gruppe von Parametern 3
769     #
770     # Bestimme die Parameter, welche fuer die zweite Gruppe an
771     # Parametern genutzt werden soll.
772     a = 2
773     b = 1.2
774     c = 0.7
775     d = 1.4
776
777     # Nutze die Funktion streamfield um eine Stromlinienfeld fuer
778     # die oberen Parameter zu erstellen, speichere es und zeige
779     # es direkt nach Ausfuehrung des Programms.
780     lgd = streamfield(a,b,c,d,K)
781     plt.savefig("streamfield_log_smallcoords.pdf",
782                 bbox_extra_artists=(lgd,), bbox_inches='tight')
783     plt.show()

```

D.3 Grafiken für Finale Diskussion

```

1 #-----
2 # Erstellen der Grafiken fuer die Diskussion
3 #-----
4 # Importieren der noetigen python-libraries.
5 import numpy as np
6 import urllib.request
7 import pandas as pd
8 import matplotlib.pyplot as plt
9 from scipy.integrate import odeint

```

```

10 from scipy.optimize import minimize
11 import random
12 import seaborn
13 from decimal import *
14
15 def fileopener():
16     """
17         Oeffnet die Datei, welche aus der von der der Webseite runter
18             geladen wurde und speicher sie in einem Dataframe und
19             gibt diesen zurueck
20     """
21     with open("snowshoe_hare_data.txt", "r") as f:
22
23         # Einlesen der Daten und erste Trennung nach newline-
24             Charaktern.
25         read_data = f.read()
26
27         sep_data = read_data.rsplit("\n")
28
29
30         # Initialisieren der 3 Arrays fuer die 3 Datenpunkte
31             der Quelle.
32         years = []
33         snowshoe_hare = []
34         lynx = []
35
36         # Extrahieren der Daten und wiederholtes Anhaengen
37             der Datenpunkte an die jeweiligen Arrays.
38         for n,entry in enumerate(sep_data):
39             entry = entry.rsplit(" ")
40
41             if n > 5 and n <21:
42                 years.append(float(entry[0]))
43                 snowshoe_hare.append(float(entry[1]))
44                 lynx.append(float(entry[2]))
45
46             # Da die Datenpunkte nicht in Originalgroessen
47                 gespeichert wurden, sonden ihre tatsaechliche
48                 Anzahl im Titel angegeben wurde, wird dier hier
49                 manuell nachgeholt.
50         snowshoe_hare = np.array(snowshoe_hare)*1000
51         lynx = np.array(lynx) *100 #*1.8
52
53         # Erstellen des dictionaries zum Erstellen des
54             dataframes
55         pre_df = {"years elapsed":years, "snowshoe hare":-
56                 snowshoe_hare, "lynx":lynx}
57
58         # Erstelle den dataframe mit den Eintraegen und gebe

```

```

                ihn zurueck.

47             df = pd.DataFrame(pre_df)
48             return df
49
50 def solver(params, Anf):
51     """
52         Diese Funktion loest die Lotka-Volterra Differenzialgleichung
53             mithilfe der numerischen Annaeherung durch die Funktion
54                 odeint() an die Loesung.
55
56     """
57     # Setze eine Zeitspanne fuer eine glatte Zeichnung der
58     # numerischen Loesung der Funtkion
59     time = np.arange(0,29,0.01)
60     a,b,c,d = params
61
62     # Definiere die Differenzialgleichung als Funktion in der
63     # Funktion und rufe sie im Anschluss von odeint auf.
64     def dy_Lotka_volterra(y,t_0):
65         dy = [0,0]
66         dy[0] = y[0]*(a-b*y[1])
67         dy[1] = y[1]*(c*y[0]-d)
68         return(dy)
69
70     y = odeint(dy_Lotka_volterra,Anf,time)
71     return(y)
72
73
74 def solver_log(params, Anf):
75     """
76         Diese Funktion loest die logistische Lotka-Volterra
77             Differenzialgleichung mithilfe der numerischen
78                 Annaeherung durch die Funktion odeint() an die Loesung.
79
80     """
81     # Setze eine Zeitspanne fuer eine glatte Zeichnung der
82     # numerischen Loesung der Funtkion
83     time = np.arange(0,29,0.01)
84     a,b,c,d,K = params
85
86     # Definiere die Differenzialgleichung als Funktion in der
87     # Funktion und rufe sie im Anschluss von odeint auf.
88     def dy_log(y,t_0):
89         dy = [0,0]
90         dy[0] = y[0]*(a*(1-y[0]/K)-b*y[1])
91         dy[1] = y[1]*(c*y[0]-d)
92         return(dy)
93
94     y = odeint(dy_log,Anf,time) #, mxstep=50000, full_output=True)
95     return(y)
96
97
98 def getindex(time_measured):
99     """
100        Diese Funktion berechnet die Zeitpunkte in der Zeitspanne,

```

```

        welche zum Zeichnen der Loesung verwendet wurde und gibt
        einen Array zurueck der die Indexe der Zeitpunkte
        enthaelt zu denen die Daten gemessen wuden.

85     """
86     # Initialisiere den Index-Array
87     index = []
88     # Nutze eine for-Schleife ueber den time-Array (Genutzt zum
89     # Zeichnen der Loesungen der Differenzialgleichungen und
90     # definiert in den Funktionen, welche diese hier aufrufen)
91     # um den Index der Zeitpunkte der Datenaufnahme im time-Array
92     # zu erhalten, gebe den Index Array zurueck.
93     for n, step in enumerate(time):
94         for rstep in time_measured:
95             diff1 = ss(np.array(df[["snowshoe hare", "lynx"]]), YP, 1)
96             diff = diff0+diff1
97             return diff
98
99
100    def plotstyleset():
101        """
102            Standardformatierung fuer die Graphiken der Phasenebene.
103        """
104        plt.xlabel(r'$Zeit \; [Jahren]$')
105        plt.ylabel(r'$Populationsgr\"o\ss{}e$')
106        plt.axhline(color="0.75")
107        plt.axvline(color="0.75")
108        plt.grid(True)
109
110    def plotter(params, Anf, solver):
111        """
112            Diese Funktion erhaelt die Parameter, sowie den Anfangswert
113            der Differenzialgleichung und eine Funktion, hier kann
114            solver oder solver_log als Argumente uebergeben werden.
115            Es zeichnet dann die fuer die gegebenen Werte die
116            Loesungskurven der Differenzialgleichung gegen die Zeit,
117            sowie die Datenpunkte in einen Graph.
118        """
119
120        # Berechne die Zeitpunkte im Zeitstrahl, zu denen die
121        # Datenpunkte berechnet wurden und speichere den Index der
122        # jeweiligen Zeitpunkte in index. Speichere die
123        # tatsaechlichen Zeitpunkte in Tm.
124        time = np.arange(0, 29, 0.01)
125        time_measured = np.array(df["years elapsed"])
126        index = getindex(time_measured)
127        Tm = time[index]
128
129        # Berechne die Loesung der Differenzialgleichung mit den

```

```

                gegebenen Parametern und Anfangswerten,
119      # Berechne ausserdem die Werte der Loesung zu den Zeitpunkten
                  der Datenmessung.
120      y = solver(params,Anf)
121      YN = y[index,0]
122      YP = y[index,1]
123
124      # Lege das Format fuer die Graph fest
125      plt.figure()
126      plotstyleset()
127
128      # Zeichne die Modelle in p1,p2 und markiere die Zeipunkte zu
                  denen Daten erhoben worden sind mit Keisen
129      p1, = plt.plot(time,y[:,0],label=r"$Modell\$ $Beute$",color =
                  "b")
130      p2, = plt.plot(time,y[:,1],label=r"$Modell\$ $R\$\"auber$",color
                  ="r")
131      p3, = plt.plot(Tm,YN,"bo")#
132      p4, = plt.plot(Tm,YP,"ro")#
133
134      # Fuege die Datenpunkte zur Grafik hinzu und verbinde sie mit
                  Linien
135      p5, = plt.plot(df["years elapsed"],df["snowshoe hare"],"co-",
                  label=r"$Schneeschuhhase$")
136      p6, = plt.plot(df["years elapsed"],df["lynx"],"go-",label=r"
                  $Kanadischer Luchs$")
137
138      # Bestimme die Legende, behalte die Grafik und mache sie
                  zugaenglich ueber den Funktionsspeicher hinaus und gebe
                  die Legende zurueck.
139      lgd = plt.legend((p1,p3),(p2,p4),p5,p6), (r"$Modell\$ $Beute$",
                  r"$Modell\$ $R\$\"auber$",r"$Schneeschuhhase$",
                  r"$Kanadischer Luchs$"), loc="upper right", fancybox=True,
                  frameon=True, shadow=True, bbox_to_anchor=(1.35, 1.1),
                  fontsize=13)
140      plt.hold(True)
141      return(lgd)
142
143 if __name__ == '__main__':
144
145      # Herunterladen und speichern der Daten.
146      url = "http://www.biographics.co.uk/newgcse/baredata.txt"
147      file_name = "snowshoe_hare_data.txt"
148      urllib.request.urlretrieve(url, file_name)
149
150      # Nutze fileopener umd die Daten in einem dataframe zu
                  speichern und diesen Anschliessend in der Konsole

```

```

        anzeigen zu lassen.

151     df = fileopener()
152     print(df)
153
154     # Festlegen der Parameter fuer die beste Anpassung des Lotka-
155     # Volterra Modells an die Daten.
156     a = 0.11 *2.6
157     b = 1/10000 *2.6
158     c = 1/100000 * 5.7
159     d = 0.47 *5.7
160     Anf = [20000,1000]
161
162     # Erhalte index der Zeitspanne zum Loesen der
163     # Differenzialgleichung, an denen die Datenpunkte erhoben
164     # wurden.
165
166     # Speichere die Parameter in params und uebergebe sie der
167     # plotter Funktion, welche die Grafik erstellt.
168     params = (a,b,c,d)
169     # c=1/100000*7
170     # d=50/100*7
171     # K=200000
172     # Anf =[20000,1000]
173
174     # Festlegen der Parameter fuer die Anpassung des logistischen
175     # Lotka-Volterra-Modells an die Datenpunkte
176     # Parameter mit einer Guten Annaeherung an die Beute und so
177     # gut wie moeglich an die Raeuber.
178     a= 0.22 *1.34
179     b= 15/100000*1.34
180     c= 1/100000*7
181     d= 50/100*7
182     K= 200000
183
184     # Speichere die Parameter in params und uebergebe sie der
185     # plotter Funktion, welche die Grafik erstellt.
186     params = a,b,c,d,K

```

D.4 Grafiken für Appendix

```

1 #-----
2 # Zeichne Steigungsfelder fuer Appendix.
3 #-----
4
5 # Importieren der noetigen python-libraries.

```

```

6 import numpy as np
7 import pylab as pl
8 import matplotlib.pyplot as plt
9 import seaborn
10
11 def dy1(y,t):
12     """
13         Beispieldifferenzialgleichung Nummer eins: dy/dt = y+t.
14         Nimmt Skalare, Arrays und Matrizen und gibt jeweils in
15             derselben Dimension die berechneten Ableitungen in den
16             jeweiligen Punkten des gegebenen Objekts zurueck.
17     """
18     return y+t,'y+t'
19
20 def dy2(y,t):
21     """
22         Beispieldifferenzialgleichung Nummer zwei: dy/dt = y+1.
23         Nimmt Skalare, Arrays und Matrizen und gibt jeweils in
24             derselben Dimension die berechneten Ableitungen in den
25             jeweiligen Punkten des gegebenen Objekts zurueck.
26     """
27     return y+1,'y+1'
28
29 def plotstyleset():
30     """
31         Standardformatierung fuer die Grafiken.
32     """
33     plt.xlabel(r'$t$')
34     plt.ylabel(r'$y$')
35     plt.axhline(color="0.75")
36     plt.axvline(color="0.75")
37     pl.grid(True)
38
39 def slopefield(fun):
40     """
41         Zeichnet ein Steigungsfeld fuer die uebergebene Funktion.
42         Voraussetzung: Funktion gibt ihre Differenzialgleichung als
43             String mit zurueck.
44     """
45
46     # Initialisiere ein numpy-Array fuer die x- und y-Achse und
47     # baue aus letzterem ein Netzwerk aus Punkten zur Berechnung
48     # der Richtungsvektoren in jedem Punkt dieses Netzwerkes.
49     # Speichere dies zwei mal als je eine Koordinaten-Matrix.
50     y1 = pl.arange(-4,4.5,0.35)
51     t,y = pl.meshgrid(y1, y1)
52     # Festlegen der Laenge des Strichs fuer das Steigungsfeld.

```

```

46      r=0.35
47
48      # Berechne die Steigung v.
49      v,tito = fun(y,t)
50
51
52      # Normiere die Strichlaenge im Steigungsfeld (Entnommen aus [
53      # Sol].
53      v = np.sqrt((r**2) / (1 + 1/V**2))      # Laenge der Striche
54      # in y-Richtung
54      u = v/V      # Laenge der Striche in x-Richtung
55
56      # Festlegen des Formats der Grafik.
57      seaborn.set_style("whitegrid")
58      pl.figure()
59      plotstyleset()
60      pl.xlim(-4,4)
61      pl.ylim(-4,4)
62
63      # Benutze die quiver-Funktion zum Zeichnen des
64      # Steigungsfeldes in jedem Punkt des Netzwerks mithilfe der
65      # Matrizen u und v.
64      # Die quiver-Funktion zeichnet in der Standardeinstellung
65      # Pfeile, da hier aber Striche erwünscht sind, werden die
66      # Parameter fuer die Pfeilköpfe auf 0 gesetzt.
65      Q = pl.quiver(t,y,u, v,zorder=10,headwidth=0,headlength=0,
66                      headaxislength=0)
67
68      # Zeichnen und speichern der Grafik und schreibe die
69      # Differenzialgleichung über die Grafik.
70      plt.title(r"\frac{dy}{dt} = %s" %(tito), fontsize=15, y=1.02)
70      plt.savefig("Beispiel_%s.pdf" %(tito))
71
72 if __name__ == '__main__':
73     # Rufe die Funktion slopefield fuer die 2 Beispiel-
74     # Differenzialgleichungsfunktionen auf.
74     slopefield(dy1)
75     slopefield(dy2)

```

13

¹³[Sol]

Literatur

- [BMT96] M. Begon, M. Mortimer und D.J. Thompson. *Population Ecology: A Unified Study of Animals and Plants*. Wiley, 1996. ISBN: 9780632034789. URL: <https://books.google.com/books?id=YDAu4RuSf3sC>.
- [CL55] A. Coddington und N. Levinson. *Theory of Ordinary Differential Equations*. International series in pure and applied mathematics. Tata McGraw-Hill, 1955. ISBN: 9780070992566. URL: <https://books.google.com/books?id=8QjvnT2hmqwC>.
- [Dar09] C. Darwin. *The Origin of Species*. P. F. Collier & Son, 1909. URL: <https://books.google.com/books?id=TCwLAAAAIAAJ>.
- [DW58] Charles Darwin und Alfred Wallace. “On the Tendency of Species to form Varieties; and on the Perpetuation of Varieties and Species by Natural Means of Selection.” In: *Journal of the Proceedings of the Linnean Society of London. Zoology* 3.9 (1858), S. 45–62. ISSN: 1945-9416. DOI: [10.1111/j.1096-3642.1858.tb02500.x](https://doi.org/10.1111/j.1096-3642.1858.tb02500.x). URL: <http://dx.doi.org/10.1111/j.1096-3642.1858.tb02500.x>.
- [Edx] *Überisicht des Online Einführungskurses über Differenzialgleichungen auf der Webseite edx*. gehört vom 20.05.2015 bis 03.06.2015. URL: <https://www.edx.org/course/introduction-differential-equations-bux-math226-1x#!>.
- [Enc] *Qualitative theory of differential equations*. Datum: 17.08.2015, 14.27 Uhr PST. URL: https://www.encyclopediaofmath.org/index.php/Qualitative_theory_of_differential_equations.
- [GY] Extension Wildlife Specialist Greg Yarrow Professor of Wildlife Ecology. *The Basics of Population Dynamics*. Fact Sheet 29. Zugriffen auf: 09.09.2015, 17.19 Uhr. URL: http://www.clemson.edu/extension/natural_resources/wildlife/publications/pdfs/fs29_population_dynamics.pdf.
- [Hol59] C. S. Holling. “The Components of Predation as Revealed by a Study of Small-Mammal Predation of the European Pine Sawfly”. In: *The Canadian Entomologist* 91 (05 Mai 1959), S. 293–320. ISSN: 1918-3240. DOI: [10.4039/Ent91293-5](https://doi.org/10.4039/Ent91293-5). URL: http://journals.cambridge.org/article_S0008347X00072564.
- [HSD13] M.W. Hirsch, S. Smale und R.L. Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Academic Press. Academic Press, 2013. ISBN: 9780123820105. URL: <https://books.google.com/books?id=csYhsrOEh\MC>.
- [Hum] *Humangenomprojekt*. Datum: 14.09.2015, 22.05 Uhr PST. URL: <https://de.wikipedia.org/wiki/Humangenomprojekt>.

- [Jac] Klaus Jacob. "Die sechste Katastrophe". In: *Süddeutsche Zeitung* (). URL: <http://www.sueddeutsche.de/wissen/massenaussterben-die-sechste-katastrophe-1.2108160>.
- [JSB07] C.J. Jensen und State University of New York at Stony Brook. *Predation and Its Consequences: Insights Into the Modeling of Interference*. State University of New York at Stony Brook, 2007. ISBN: 9780549472292. URL: <https://books.google.com/books?id=2cp5pEsvvUsC>.
- [Kan] *Kanadischer Luchs - Wikipediaseite*. Datum: 10.09.2015, 15.51 Uhr PST. URL: https://en.wikipedia.org/wiki/Canada_lynx.
- [Lin] *Linearization*. Datum: 14.09.2015, 19.32 Uhr PST. URL: <https://en.wikipedia.org/wiki/Linearization>.
- [LM11] J. Liesen und V. Mehrmann. *Lineare Algebra*: Bachelorkurs Mathematik. Vieweg+Teubner Verlag, 2011. ISBN: 9783834882905. URL: <https://books.google.com/books?id=iZHwTIEa-psC>.
- [Log] *Logistic Function*. Datum: 17.08.2015, 12.42 Uhr PST. URL: https://en.wikipedia.org/wiki/Logistic_function#Applications.
- [Lot] *Lotka-Volterra-Gleichungen*. Datum: 22.06.2015, 16.08 Uhr PST. URL: <https://de.wikipedia.org/wiki/Lotka-Volterra-Gleichungen>.
- [Mal12] T.R. Malthus. *An Essay on the Principle of Population*. Dover Publications, 2012. ISBN: 9780486115771. URL: <https://books.google.com/books?id=0qYXcMM3hqYC>.
- [Mat] *Mathematical and theoretical biology*. Datum: 14.09.2015, 17.27 Uhr PST. URL: https://en.wikipedia.org/wiki/Mathematical_and_theoretical_biology.
- [MG87] W. Metzler und D. Gockert. *Dynamische Systeme in der Ökologie: Mathematische Modelle und Simulation*. Teubner Studienbücher Mathematik. Vieweg+Teubner Verlag, 1987. ISBN: 9783519020820. URL: <https://books.google.com/books?id=aSJvAAAACAAJ>.
- [Nag12] Rainer Nagel. *Lineare Dynamische Systeme*. Techn. Ber. LATEX- Edition von Markus Couturier und Tom-Niklas Faber. Universität Tübingen, 2012.
- [Pan10] Alexander Panfilov. *Qualitative Analysis of differential equations*. Techn. Ber. Utrecht University, Theoretical Biology, 2010. URL: <http://www-binf.bio.uu.nl/panfilov/bioinformatica/bioinf10.pdf>.
- [PEDP99] Ruth E Patterson, David L Eaton und JOHN D POTTER. "The genetic revolution: change and challenge for the dietetics profession". In: *Journal of the American Dietetic Association* 99.11 (1999), S. 1412–1420.
- [Pop] *Populationsökologie*. Datum: 23.06.2015, 11.03 Uhr PST. URL: <https://de.wikipedia.org/wiki/Populations%C3%B6kologie>.

- [Pyta] *Dokumentation matplotlib.pyplot - contour()*. Datum: 08.09.2015, 22.48 Uhr PST. URL: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.contour.
- [Pytb] *Dokumentation matplotlib.pyplot - quiver()*. Datum: 15.08.2015, 11.34 Uhr PST. URL: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.quiver.
- [Pytc] *Dokumentation matplotlib.pyplot - streamplot()*. Datum: 15.08.2015, 11.33 Uhr PST. URL: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.streamplot.
- [Pytd] *Dokumentation scipy.integrate - odeint()*. Datum: 15.08.2015, 11.31 Uhr PST. URL: <http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html>.
- [RM] M. L. Rosenzweig und R. H. MacArthur. “Graphical Representation and Stability Conditions of Predator-Prey Interactions”. In: *The American Naturalist* 97.895 (), pp. 209–223. ISSN: 00030147. URL: <http://www.jstor.org/stable/2458702>.
- [Sol] *Normalisierung der Vektorlänge des Steigungsfeldes*. Datum: 08.09.2015, 16.47 Uhr PST. URL: <http://matplotlib.1069221.n5.nabble.com/Need-help-with-direction-field-plot-td24886.html>.
- [Sto08] Franziska Stocker. *Phasenporträts linearer Systeme*. Techn. Ber. Ausarbeitung zum Vortrag im Seminar Mathematische Modellierung. Ruprecht-Karls-Universität Heidelberg, 2008/2009. URL: <https://www.mathi.uni-heidelberg.de/~thaeter/mathmod08/Stocker.pdf>.
- [Tay] *Linear Approximation*. Datum: 14.09.2015, 19.41 Uhr PST. URL: https://en.wikipedia.org/wiki/Linear_approximation.
- [Texa] *Datensatz des Schneeschuhhasen und Luchs*. Datum: 14.08.2015, 14.12 Uhr PST. URL: <http://www.biographics.co.uk/newgcse/baredata.txt>.
- [Texb] *Schneeschuhhase und Luchs Information zum Datensatz*. Datum: 14.08.2015, 14.12 Uhr PST. URL: <http://www.biographics.co.uk/newgcse/predatorprey.html>.
- [Thi12] Katja Thielbörger. *Skript zur Vorlesung Natur- und Artenschutz, Ökologie und Biodiversität II*. 2012.
- [TP63] M. Tenenbaum und H. Pollard. *Ordinary Differential Equations: An Elementary Textbook for Students of Mathematics, Engineering, and the Sciences*. A Harper international student reprint. Dover Publications, 1963. ISBN: 9780486649405. URL: <https://books.google.com/books?id=Feggn8pgDP4C>.

- [Tur13] P. Turchin. *Complex Population Dynamics: A Theoretical/Empirical Synthesis (MPB-35): A Theoretical/Empirical Synthesis (MPB-35)*. Monographs in Population Biology. Princeton University Press, 2013. ISBN: 9781400847280. URL: <https://books.google.com/books?id=DbLWAQAAQBAJ>.
- [Web] *Math 646 - Mathematical Modeling, Fall Semester 2010, Lotka-Volterra Models*. Datum: 15.08.2015, 11.46 Uhr PST. URL: <http://www-rohan.sdsu.edu/~jmahaffy/courses/f09/math636/lectures/lotka/qualde2.html#equilibria>.
- [Wer08] Bodo Werner. *Qualtitative Theorie bei gewöhnlichen Differentialgleichungen*. Techn. Ber. Universität Hamburg, 2008. URL: <http://www.math.uni-hamburg.de/home/werner/QuThODE.pdf>.

Erklärung

„Ich erkläre, dass ich die Arbeit selbstständig angefertigt und nur die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken, gegebenenfalls auch elektronischen Medien, entnommen sind, sind von mir durch Angabe der Quelle und des Zugriffsdatums sowie dem Ausdruck der ersten Seite belegt; sie liegen zudem für den Zeitraum von 2 Jahren entweder auf einem elektronischen Speichermedium im PDF-Format oder in gedruckter Form vor.“

Datum

Unterschrift: Sanja Stegerer