Medizinische Universität Wien

DIPLOMARBEIT

Toolbox zur Simulation inhomogener Populationen
Anwendung im Bereich Neuronenpopulationen

Thema

ausgeführt am Institut für

Medizinische Kybernetik und Artificial Intelligence

der Medizinischen Universität Wien

unter der Anleitung von ao. Univ.-Prof. Dipl.-Ing. Dr. Georg Dorffner

durch

Manfred Anzinger

Name

Säulengasse 17/27, 1090 Wien

Anschrift

Datum

Unterschrift

Danksagung

Ich möchte ganz besonders meiner Freundin Eva danken, dass sie mich treu durch mein Studium begleitet hat und all die Zeit meinen Unmut und meine Launen ertragen hat.

Weiters möchte ich Hr. Prof. Dorffner danken, der immer zur rechten Zeit den nötigen Hinweis gab um den nächsten Schritt durchzuführen.

Und natürlich meinen Studienkollegen Qian und Stefan, die immer mit Rat und Tat zur Seite standen und ohne die das Fertigstellen des Studiums in dieser Zeit nicht möglich gewesen wäre.

Zusammenfassung

Um ein Zusammenwirken von Populationen, wie es in Bereichen des Gehirns wie dem Hippocampus, Thalamus oder Olfactory Tract der Fall ist, zu simulieren, benötigt man eine geeignete Simulationsumgebung. Die Toolbox, die in dieser Arbeit beschrieben wird, wurde in MatLab® entwickelt. Sie stellt Modelle und Funktionen zur Verfügung, die einen Umgang mit homogenen Teilpopulationen erleichtern. Die implementierten Modelle werden zu Beginn vorgestellt und anschließend folgt eine detaillierte Beschreibung der Arbeitsweise der Toolbox. Am Schluss werden Beispiele behandelt, die sowohl die ausgesuchten Modelle näher bringen, wie auch Stärken und Schwächen der Toolbox aufzeigen.

Abstract

To simulate the behaviour of populations and there relationships together, like they occur for example in areas as the hippocampus or the olfactory tract, you need an appropriate simulation tool. In this work we will propose a toolbox, developed in MatLab®, which is able to solve such problems. Models and functions are implemented to make the interaction of populations like delays in signal transport easier. At begin, the implemented models will be described and then we give a detailed introduction to the toolbox and it work mode. The last part is describing different examples, which are selected to show the manner of work of the toolbox and how the different models can be used.

Inhaltsverzeichnis

1 Einführung und Motivation

Ursprünglicher Ausgangspunkt dieser Arbeit war die Suche nach einem Modell, mit dem es möglich ist, die verschiedenen Stadien des Schlafes, wie die REM–Phase (Rapid Eye Movement), die „Slow Wave"-Phase und die daraus entstehenden Spindeln, zu modellieren [1]. Die anfängliche Idee war, die Aufgabe mittels pulsierenden neuronalen Netzwerken [2] zu realisieren. Bei pulsierenden neuronalen Netzen bilden einzelne Neuronen die Basis, die in verschiedenen Detailierungsstufen eine Nervenzelle modellieren, die als Output einen Spike produzieren. Dies hatte zur Folge, dass bei größeren Netzen die Anzahl der Parameter unüberschaubar und eine Aussage über die Qualität des Netzes schwer machte.
Um die Anzahl der Parameter zu verringern und eine Simulation überschaubar zu machen, findet man in der Literatur die sogenannten Netzwerkmodelle [3], die als Grundlage ganze Neuronenpopulationen modellieren, von denen man annimmt, dass sie in sich homogen sind. Das heißt, eine Population besteht aus Zellen gleichen Zelltyps, mit identen Parametern. Diese haben weiters den Vorteil, dass sie Probleme wie die Interpretation von Feuerungsraten in ein durchschnittliches extrazelluläres Potential und die Filterung dieser zum Teil bereits mitmodellieren [3]. Dies erspart eine Menge an Modellierungsaufwand und bringt deutliche Vorteile in der Simulationsgeschwindigkeit.
Netzwerkmodelle können allerdings keine Aussage über die Phasenlage von einzelnen Spikes oder die Korrelation dieser treffen und haben so auch ihre Einschränkungen [4].

Um nun den Aufgaben gerecht werden zu können, und mit den in der Literatur gefundenen Modellen zur Beschreibung homogener Populationen arbeiten zu können, bestand die Notwendigkeit einer geeigneten Toolbox. Diese sollte Aufgaben, wie das Vernetzen von verschiedenen Populationen in verschiedenster Art und Weise, die Berücksichtigung von Verzögerungen in den einzelnen Verbindungen und das Verwenden von unterschiedlichen Modellen mit demselben Netz, lösen

helfen. Da eine solche zu dieser Zeit nicht zur Verfügung stand und das Simulieren unter MatLab® mit ODE Funktionen sich alsbald als zu mühselig herausstellte, wurde diese Toolbox geschrieben.

Diese Arbeit gliedert sich nun in drei unterschiedliche Bereiche. Der erste Teil bringt Allgemeines zu dynamischen Systemen, Modellbildung und Simulation. Der Beginn des zweiten Teils beschäftigt sich mit neurophysiologischen Aspekten des Olfactory Bulb, um die daraus gewonnen Modelle besser verstehen zu können. Daraufhin werden Modelle vorgestellt, die sowohl homogene Teilpopulationen abbilden können, als auch in der Toolbox implementiert sind.
Im dritten und letzten Teil wird schließlich die Toolbox im Detail erklärt und anhand ausgesuchter Beispiele die Arbeitsweise gezeigt. Dabei werden für jedes implementierte Modell Beispiele herangezogen, die sowohl das Modell selbst besser erklären, als auch Probleme die Toolbox betreffend aufzeigen. Diese Beispiele sollen letztendlich helfen, die Toolbox besser und effizienter einzusetzen.

2 Simulation dynamischer Systeme

Die folgenden Erläuterungen wurden vorwiegend aus Bossel 1992 entnommen [5].

2.1 System und Modellbildung

Wir wollen zuerst den Begriff System genauer erläutern, denn nicht alle Objekte in unserer Umwelt sind ein System. Wir nennen demnach ein Objekt ein System wenn es die folgenden Eigenschaften besitzt:

1. Das Objekt erfüllt einen bestimmten Zweck.
2. Das Objekt besteht aus einer bestimmten Anzahl von Elementen die zueinander in bestimmter Relation stehen und so die Funktion des Objekts bestimmen.
3. Das Objekt ist nicht teilbar, d. h. wird ein Element herausgelöst oder zerstört verliert das Objekt seinen Zweck. Die Systemidentität hat sich verändert.

Wir wollen diese Punkte für unser späteres Beispiel, den Olfactory Tract (das Riechsystem), an dieser Stelle prüfen:

Der Zweck: das Erkennen und Unterscheiden von Gerüchen, neue Gerüche mit entsprechenden Assoziationen speichern.

Die Anzahl der Elemente und deren Relationen: verschiedene Typen von Nervenzellen, deren Verbindungen zueinander.

Die Teilbarkeit: nimmt man einen kompletten Neuronentyp heraus wird das Erkennen etc. nicht mehr möglich sein.

Wir haben hier also eindeutig ein System vorliegen.
Systeme weisen weiters Eigenschaften auf anhand derer sie klassifiziert werden können:

offen/abgeschlossen

Bei einem offenen System bestehen Wechselwirkungen an den Systemgrenzen mit der Umwelt. Ein Beispiel wäre die Ausscheidung eines Medikamentes über die Zeit. Geschlossene Systeme dagegen zeigen keine Wechselwirkung mit der Umwelt, es wird zum Beispiel nur die kinetische Verteilung des Medikamentes im Körper ohne Ausscheidung berücksichtigt, und sind daher idealtypisch. Es ist daher notwendig, die Grenzen des Systems genau abzustecken, um es genauer untersuchen zu können. Grenzen sind dabei so zu wählen, dass nach innen ein starker Zusammenhalt und nach außen eine schwache Bindung besteht. Beim Olfactory Tract wären das z.B. die Rezeptorzellen zur Außenwelt.

dynamisch/statisch

Man spricht von dynamischen Systemen wenn sich Systeme über die Zeit ändern, wie zum Beispiel die Anzahl der Bevölkerung. Ein statisches System wäre zum Beispiel ein Haus, dessen Systemgrössen sich nicht ändern. Streng genommen sind in der Natur alle Systeme dynamisch, da auch ein Haus einem Alterungsprozess unterliegt.

kontinuierlich/diskret

Kontinuierliche Systeme ändern sich in beliebig kleinen Zeitabständen. Diskrete dagegen ändern sich Sprunghaft in bestimmten Zeitabständen, wie zum Beispiel die Verzinsung eines Kapitals. Das ist wichtig bei der Wahl des Integrationsverfahrens. Für diskrete Systeme dürfen nur Integratoren verwendet werden, die keine Stützstellen zwischen den Zeitschritten berechnen und die Schrittweite muss der des Systems entsprechen. Ein mögliches Verfahren für diskrete Systeme wäre das Euler-Cauchy Verfahren (siehe Anhang).

deterministisch/stochastisch

Unter identischen Bedingungen sind die Folgezustände in deterministischen Systemen reproduzierbar, in stochastischen sind diese nur durch Zufallsvariablen beschreibar.

Wir behandeln in unsere Arbeit demnach Systeme die die Eigenschaften *offen, dynamisch, kontinuierlich* und *deterministisch* besitzen.

Um den Zustand eines Systems genauer beschreiben zu können, müssen die Zustandsgrößen herausgearbeitet werden. Der Vektor der Zustandsgrößen bestimmt dann die Weiterentwicklung des Systems. Die Zustandsgrößen sind dabei voneinander unabhängig, d.h. sie lassen sich nicht von anderen Größen ableiten. Weiters geben sie die Dimension in einem System wieder. Das zeigt sich, indem für jede Zustandsgröße in dem späteren Modell eine Differentialgleichung existieren muss, die diese beschreibt. In unserem Beispiel des Olfactory Tract ist das die Erregtheit der einzelnen Neuronenpopulationen. Sie stellen gleichzeitig das Gedächtnis des Systems dar.

In unseren Simulationen, speziell auch bei Neuronenpopulationen haben wir das Problem, dass wir unter Umständen sehr viele Populationen, und damit auch eine hohe Dimensionalität, in unserem System vorfinden. Das System wird dann jedoch unüberschaubar und eine Vorhersage meist unmöglich. Daher müssen diese dann in Teilsysteme zerlegt werden, um sie getrennt analysieren zu können. Bei einer späteren Kopplung kann dann das gesamte System leichter verstanden werden. Wir werden später sehen, dass wir Neuronenpopulationen immer in Vierergruppen teilen und diese dann durch sog. laterale Gewichte koppeln.

Ist das System genau definiert, kann daraus ein Modell gebildet werden. Ein Modell stellt immer nur einen vereinfachten Ausschnitt der Realität dar. Es soll auch nur für diesen Ausschnitt eine gültige Aussage vermitteln. Es gibt dabei zwei Möglichkeiten, aus dem System ein Modell zu erstellen:

1. Man bildet das *Verhalten* des Systems nach. Dabei wird das System selbst als Blackbox betrachtet und die Reaktion auf einen Input bzw. die Änderung eines Parameters beobachtet und dieses Verhalten schließlich mit dem Modell nachgeahmt.
2. Man bildet die einzelnen Elemente des Systems und deren Relationen zueinander ab. Stimmen diese mit dem realen System überein, so ergibt sich gleiches Verhalten bei gleichen Größenänderungen.

Mischformen von 1. und 2. sind zulässig und meistens auch der Fall. In unseren später vorgestellten Modellen liegt auch meist eine Mischform vor.

2.2 Simulation des Systems

Wir haben nun am Schluss ein Modell über unser System erhalten, das normalerweise in mathematischer Form als Differentialgleichungen vorliegt. Um diese mathematischen Formalismen in ein Programm umsetzen zu können, sind folgende Schritte notwendig:

Auswahl einer geeigneten Entwicklungsumgebung:
Kriterien dazu sind die Art des Modells, Erweiterbarkeit des Systems, zu verwendende Programmiersprache, persönliche Präferenzen des Entwicklers, aber auch der Bekanntheitsgrad um Simulationen an Dritte weitergeben zu können.
MatLab® wird zur Zeit zu sehr vielen verschiedenen Simulationsaufgaben benutzt und hat daher einen weiten Anwenderkreis.

Auswahl eines geeigneten Integrationsverfahrens:
Man kennt zur numerischen Lösung von Differentialgleichungen das *Einschrittverfahren* und das *Mehrschrittverfahren*. Beim Einschrittverfahren wird immer zur Berechnung des Wertes x_{i+1} an der Stelle $t_{i+1}=t_i+\Delta t$ der vorher berechnete Wert x_i an der Stelle t_i verwendet. Beim Mehrschrittverfahren dagegen wird ein Mittelwert über die Funktionswerte über mehrere Zeitpunkte gebildet. Durch dieses Verfahren wird jedoch eine Glättung der Funktion erreicht, was insbesonders bei sprunghaften Änderungen der Eingangsgrößen das Ergebnis verfälscht. Mehrschrittverfahren sind daher für dynamische Systeme nicht geeignet [6].
Methoden für Einschrittverfahren sind das *Euler-Cauchy* und das *Runge-Kutta* Verfahren (siehe Anhang) [7]. Das Runge-Kutta Verfahren (4,5) z.B. berechnet innerhalb eines Zeitschrittes 4 Stützstellen und bestimmt somit die ersten 5 Elemente der Taylorreihe an einem Punkt. Das Euler-Cauchy Verfahren hingegen ist ein Verfahren 1. Ordnung und bestimmt die ersten 2 Elemente. Mit dem Runge-Kutta

Verfahren erzielt man daher eine wesentlich höhere Genauigkeit bei gleicher Schrittweite und wurde auch für diese Toolbox ausgewählt. Zur Lösung diskreter dynamischer Systeme ist, wie bereits erwähnt, allerdings nur das Euler-Cauchy Verfahren geeignet.

Laufzeitparameter:

Zur Simulation muss ein endlicher Zeitausschnitt gewählt werden, d.h. der Startzeitpunkt und Endzeitpunkt müssen bekannt gegeben werden. Besondere Bedeutung kommt hier der Wahl der Schrittweite zu. Sie bestimmt Dauer und Genauigkeit der Berechnung. Durch die endliche Schrittweite entsteht ein sog. Abbruchfehler (Diskretisierungsfehler). Doch eine möglichst kleine Schrittweite zu wählen ist nicht die Lösung des Problems. Neben diesem Abbruchfehler gibt es nämlich auch noch den Rundungsfehler der zum Gesamtfehler des Ergebnisses beiträgt. Dieser entsteht durch die endlich genaue Darstellung von Zahlen im Rechner und wächst mit der Zunahme der Einzelschritte der Berechnung. Es existiert daher ein Optimum der gewählten Schrittweite Δt (siehe Abbildung).

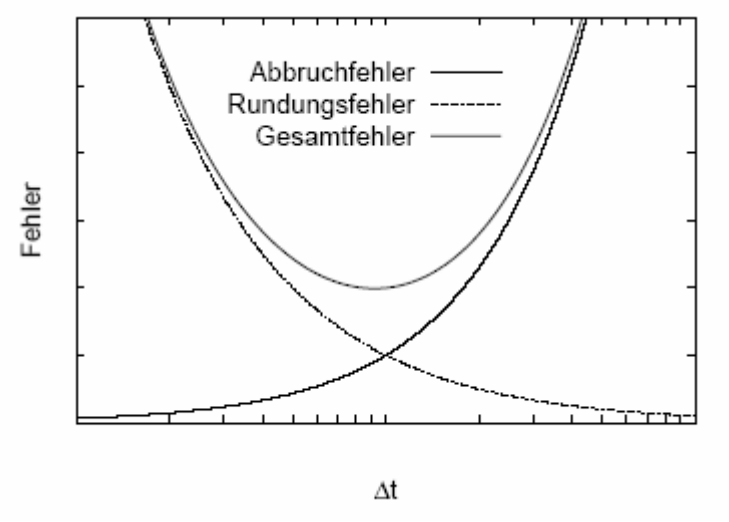


Abbildung 1: Die Darstellung zeigt das Verhalten des Gesamtfehlers in Abhängigkeit der Schrittweite.

Viele Integratoren berechnen die Schrittweite anhand der Steilheit der vorangegangenen Ergebnisse und erreichen so eine bessere Performanz bei gleicher Genauigkeit. Diese Methode ist in unserem Fall nicht möglich, da die Integration immer an bestimmten Zeitpunkten erfolgen muss, um eine Vorausberechnung der Verzögerungen zu ermöglichen.

Verzögerungen: Verzögerungen müssen Systemzustände speichern. Um das zu bewerkstelligen verwendet man meistens Haltespeicher. Diese haben die Aufgabe einen ihnen gemeldeten Zustand zum Zeitpunkt t zu einem späteren Zeitpunkt $t+T$ weiterzugeben, wobei T die Zeit der Verzögerung ist. Das heißt die Zielzustandsgröße erhält den Wert zum Zeitpunkt t vom Zeitpunkt $t-T$. Dazu sind bei der Initialisierung des Systems die vergangenen Werte von Bedeutung, die jedoch normalerweise nicht bekannt sind und deshalb mit *0* angenommen werden. In der Toolbox ist das Problem der Verzögerung gelöst, indem für jede Zustandsgröße ein Haltespeicher bereitgestellt wird, in den jede Quellzustandsgrösse den errechneten Wert zum Zeitpunkt $t+T$ der Zielzustandsgrösse ablegt.

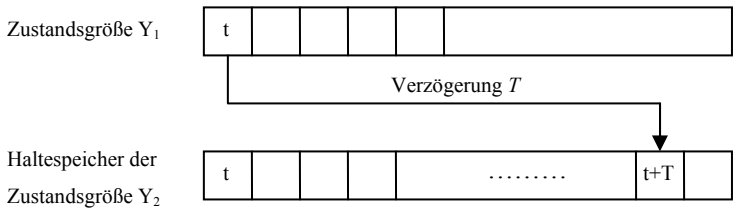


Abbildung 2: Realisierung des Haltespeichers für Verzögerungen.

Anfangswerte: Sie bestimmen den Wert der Zustandsgrößen zu Beginn der Simulation.

Systemparameter: müssen zu Beginn der Simulation bekannt sein. Die Reaktion des Systems (genau genommen des Modellsystems) bei Änderungen von Systemparametern bei verschiedenen Simulationsdurchläufen ist hier von Interesse.

Umwelteinwirkungen: Wieder ist die Reaktion des Systems auf äußere Einwirkungen von Interesse. Die Umwelteinwirkungen müssen vor Beginn der Simulation angegeben werden, wobei Form, Dauer und Zeitpunkt der Einwirkung bekannt sein muss. Im Weiteren werden diese als Stimulus bezeichnet.
In vielen Simulationsumgebungen ist eine gesonderte Angabe eines Stimulus nicht vorgesehen und muss als Funktion der Zeit dem Modell beigefügt werden. Die Toolbox sieht eine getrennte Betrachtung des Stimulus vor.

Szenarien: Bei komplexeren Systemen gibt es sehr schnell eine unüberschaubare Menge an Möglichkeiten der Kombination von Systemparametern und

verschiedenen Einwirkungen. Um diese zu reduzieren müssen sie in plausible und stimmige Szenarien zusammengefasst werden.

Ergebnisdarstellung: Es gibt meist verschiedene Arten das Ergebnis zu präsentieren. Diese sind dann je nach Art der Daten zu wählen. Tabellen, Grafiken, Phasenraum, etc.

2.3 Analyse des Systems

Besondere Bedeutung kommt der Analyse eines Systems dann zu, wenn dieses ein generisches Modell besitzt, das mit veränderten Parametern bei anderen Aufgaben Anwendung findet. Ein gutes Beispiel hierfür ist das Modell 3 (Kapitel 6) in dieser Arbeit, das zur Beschreibung der Infektionskrankheit Gonorrhöe entwickelt worden ist, aber generell zur Beschreibung inhomogener Populationen verwendet werden kann.
Bei der Analyse sind dann die Gleichgewichtspunkte und Attraktoren des Systems und deren Stabilität von Interesse.

Gleichgewichtspunkte: Ein Gleichgewicht liegt vor, wenn die Änderungen der Zustandsgrößen in einem Zeitschritt gegen Null gehen. Diese Gleichgewichtspunkte können stabil oder instabil sein. Eine empirische Aussage über die Stabilität erhält man, indem man die Anfangswerte in die Nähe des vermeintlichen Gleichgewichts setzt. Bewegen sich die Ergebnisse in Richtung dieses Punktes so ist er stabil. Bewegen sie sich weg von diesem Punkt so ist er instabil. Bei der empirischen Analyse ist jedoch größte Vorsicht geboten. Speziell bei nichtlinearen komplexen Systemen, können die Zustandsgrößen erst nach langerer Zeit von einem vermeintlichen Gleichgewicht abwandern.

Linearisierung: Besteht das Modell aus nicht linearen Gleichungen, was meist der Fall ist, so gestaltet sich eine Stabilitätsuntersuchung der Gleichgewichtspunkte schwer. Oft besteht hier allerdings die Möglichkeit, unter der Annahme, dass sich das System am Gleichgewichtspunkt nur geringfügig ändert, es am Gleichgewichtspunkt

zu Linearisieren. Dadurch stehen wieder die mathematischen Werkzeuge der linearen Systemanalyse, wie Eigenwertberechnung, Jakobimatrix etc., zur Verfügung.

Attraktoren und Phasenräume: Nichtlineare höherdimensionale Systeme besitzen anstatt von Gleichgewichtspunkten oft Attraktoren. Diese zeichnen sich dadurch aus, dass sie sich zu einem Zustandsraum hinbewegen. Innerhalb dieses Zustandsraumes tritt dann oft ein chaotisches Verhalten auf.
Die Anzahl der Attraktoren in einem System ändert sich mit der Größe der Systemparameter. Erreicht ein Systemparameter einen kritischen Punkt an dem sich die Anzahl oder die Stabilität ändert, dann nennt man diesen Bifurkationspunkt. Bifurkationspunkte stehen immer im Zusammenhang mit der Änderung der Realteile der Jacobimatrix des Systems [8]. Attraktoren lassen sich anhand ihrer Eigenschaften klassifzieren:

- Sattelknoten-Bifurkation: ein stabiler und ein instabiler Fixpunkt kollidieren und löschen sich aus.
- Transkritische Verzweigung: ein stabiler und ein instabiler Fixpunkt schneiden sich und wechseln ihre Stabilität.
- Heugabel-Bifurkation: aus einem stabilen Fixpunkt entstehen zwei stabile und ein instabiler Fixpunkt.
- Hopf-Bifurkation: ein stabiler Fixpunkt wird instabil. Endet er in einem stabilen Grenzzyklus so nennt man ihn *superkritisch* (Abbildung 3), ansonsten nennt man ihn *subkritisch*.

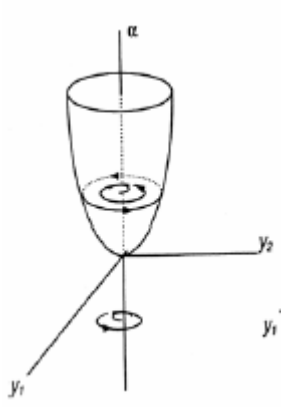


Abbildung 3: Darstellung des Verlaufs einer Hopf-Bifurkation mit stabilem Grenzzyklus.

Die Darstellung der Attraktoren erfolgt im sogenannten Phasenraum. Der Phasenraum gibt alle möglichen Zustände zweier Zustandsgrößen wieder. Die Darstellung bei höheren Dimensionen gestaltet sich allerdings wieder schwierig.

3 Der Olfactory Tract

Das Wort „olfactare“ kommt aus dem Lateinischen und bedeutet „ an etwas riechen“. Demnach umfasst der Olfactory Tract (OT) das gesamte Riechsystem. In diesem Kapitel werden die einzelnen Bereiche und die darin vorkommenden Nervenzellen des OT kurz erläutert, nicht zuletzt weil nachfolgende Modelle entweder aus Untersuchungen des Olfactory Tract motiviert und daraus entstanden sind, oder dort ihre Anwendung gefunden haben. Außerdem ist dieser gut untersucht, und das Zusammenspiel von Neuronenpopulationen, wie speziell im Olfactory Bulb von Mitralzellen und Granulzellen, ist gut erforscht. Nicht zuletzt ist es eines der einfachsten Sensorsysteme in Struktur und Funktion.
Es existieren eine Menge Modelle über den OT, welche sowohl detaillierte Nervenmodelle beinhalten [9, 10] als auch Netzwerkmodelle wie wir sie nachfolgend vorstellen werden [3, 11].

3.1 Morphologie und Verschaltung des Olfactory Tract

Der Mensch besitzt ca. 30 Millionen Riechzellen, die in die Riechschleimhaut, die Mucosa, eingebetet sind. Riechzellen besitzen eine durchschnittliche Lebensdauer von ca. einem Monat und werden dann von adulten Stammzellen wieder erneuert. Dies ist eines der seltenen Beispiele im adulten Nervensystem wo eine Erneuerung von Nervenzellen stattfindet. Mit weiteren Zellen wie den Stützzellen und den Basalzellen bilden sie gemeinsam das Riechepithel, das in der Nasenhöhle liegt. Die Axone der Riechzellen laufen zu Tausenden gebündelt durch die Siebbeinplatte zum Olfactory Bulb (OB) und bilden gemeinsam den *Nervus olfactorius*. Der Olfactory Bulb wird als vorgelagerte Hirnregion betrachtet [12].

Der OB bildet die Hauptschaltstelle des Geruchssystems. Hier bilden die Axone der Rezeptorzellen mit den Dendriten der Mitralzellen und Periglomerularzellen die

Glomeruli, das sind nervenfaserknäuelartige Gebilde. Das ist die erste und einzige Verschaltung der Axone der Riechzellen. Hier kommt es zu einer starken Reduktion der Information, tausende Axone treffen auf die Dendriten einer einzigen Mitralzelle. Die Zahl der Glomeruli wird durch die Zahl der Riechzellen bestimmt, Größe und Anordnung sind bei allen Vertebraten ähnlich.

Der OB ist in Schichten organisiert, auf die Schicht der Glomeruli und Periglomerularzellen folgt jene der Mitralzellen und darunter liegt die Schicht der Granulzellen.

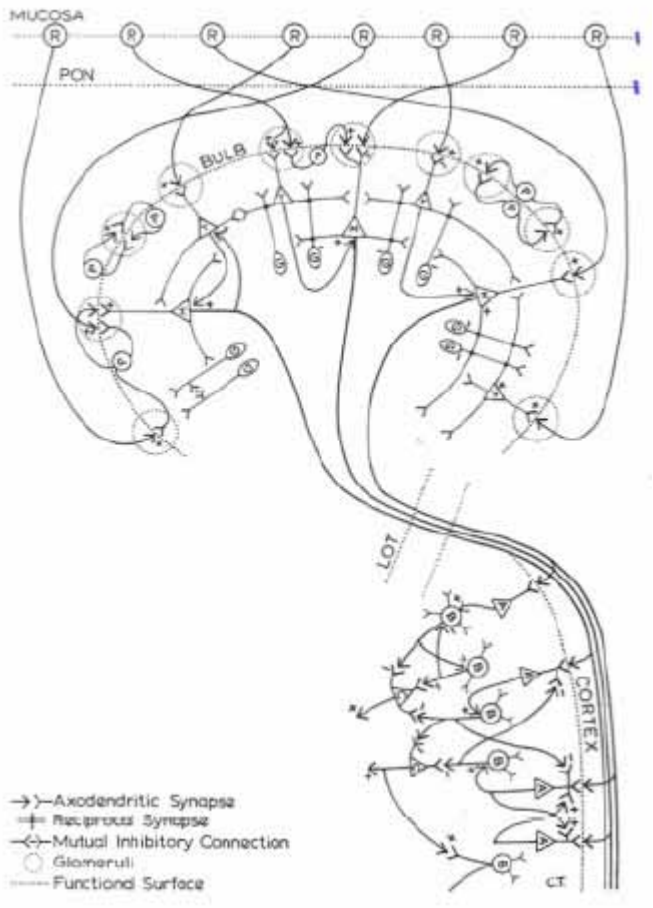


Abbildung 4: Schematische Darstellung der wichtigsten Zelltypen im Olfactory System und deren Verschaltung im Olfactory Bulb und Prepyriform Kortex. R, Rezeptorzellen; PON Primary Olfactory Nerve; LOT, Lateral Olfactory Tract; M, Mitralzellen; G, Granulzellen; P Periglomerularzellen; A, außen liegende Pyramidenzellen; B, Interneuronen des Kortex; C, innen liegende Pyramidenzellen. Aus Freeman [13].

Die etwa 30.000 Axone der Mitralzellen bilden schließlich den Lateral Olfactory Tract (LOT). Dieser ist der einzige Ausgang des Olfactory Bulb welcher die erhaltene Information in den Prepyriform Kortex (PC) weiterleitet. Dort wird die Information weiter im Kortex verteilt. Im PC bilden Pyramidenzellen wieder excitatorisch/inhibitorische Verbindungen mit Interneuronen welche wieder ein ähnliches Verhalten wie im Olfactory Bulb zeigen [12], siehe Abbildung 4.

3.2 Die Funktionsweise des Olfactory Tract

Der Olfactory Tract wird, hier stellvertretend für gesamten Kortex, als parallele Recheneinheit erfasst [13]. Trifft ein Geruch auf die Rezeptorzellen des Primary Olfactory Nerve (PON), so wird ein bestimmtes Muster auf die Glomeruli des Olfactory Bulb projiziert. Die Glomeruli sind horizontal durch die Periglomerularzellen verbunden. Periglomerularzellen sind inhibierende Interneuronen (Neurotransmitter GABA) und bewirken eine laterale Inhibierung an den umliegenden Glomeruli welches eine Schärfung des Geruchsmusters zur Folge hat.

Mitralzellen bilden wiederum mit den Granulzellen eine sogenannte dendrodendritische Verbindung. Dabei erhalten Granulzellen über erregende Synapsen (Neurotransmitter NMDA und AMPA) ein Aktionspotential und geben ihrerseits wiederum eine verlängerte Salve über inhibitorische Synapsen (Neurotransmitter GABA) an die angrenzenden Mitral- und Granulzellen ab.
Dadurch entsteht eine Hyperpolarisierung und weitere Spikes werden unterdrückt bis die Aktivität der Granulzellen nachlässt oder der Stimulus durch den Geruch an den Mitralzellen groß genug ist um ein erneutes Aktionspotential auszulösen. Durch die wechselseitige Beziehung geraten beide Zelltypen ins Schwingen, und bilden einen „limit cycle“. Ähnlich wie zuvor bei den Periglomerularzellen findet auch hier eine laterale Inhibierung statt, welche ein räumlich–zeitliches Muster im Bulb bildet.
Aufgrund der erhöhten Aktivität in stimulierten Zellassembly steigt die Kalziumkonzentration an der Innenseite der Zellmembran [14, 15], was wieder eine Adaptierung der Ausgangsfrequenz der Zelle zur Folge hat [16, 17].

Über den LOT wird das hervorgerufene Muster in den Kortex projiziert [18, 19]. Efferente Axone projizieren wiederum vom PC zurück in den Bulb.

Durch neuere Verfahren in denen Kalzium, das mit einem fluoreszierenden Marker versehen ist, in die zu untersuchende Region injiziert wird, kann die Aktivität und so ein räumliches Muster des Bulb mittels Laser Scanning Mikroskopie erstellt werden [20, 21].

4 Das K – Modell

Das K-Modell wurde von Walter J. Freeman aus Untersuchungen des Olfactory Tracts entwickelt [11, 13, 22]. Dabei wurden die einzelnen Bereiche, wie der Olfactory Bulb (OB), der Anterior Olfactory Nucleus (AON) und der Prepyriform Kortex (PC), stimuliert und die Antwort auf diesen Stimulus ausgewertet. Das Resultat ist ein Differentialgleichungssystem 2. Ordnung das aus einem linearen und einem nichtlinearen Teil besteht.

4.1 Gleichungssystem und Aufbau des Modells

Das Modell ist hierarchisch aufgebaut und besteht in seiner Grundstruktur aus dem K0 Objekt. Verbindet man zwei K0 Objekte mit positiven Gewichten erhält man ein KIe bzw. mit negativen Gewichten ein KIi Objekt. Die Verbindung von einem KIe und einem KIi bildet ein KII Objekt. Verschaltete KII-Objekte bilden ein KIII Objekt.

4.1.1 Das K0 Objekt

Das K0 Objekt bildet die Grundform des K Modells und besteht aus einem linearen und einem nichtlinearen Teil.

Der lineare Teil lautet:

$$\frac{1}{a*b}\frac{d^2}{dt^2}v_n(t)+\frac{a+b}{a*b}\frac{d}{dt}v_n(t)+v_n(t)=F(v_n) \qquad (1)$$

wobei v_n die Erregung der Population darstellt.
Der nichtlineare Teil lautet:

$$F(v_i) = \sum_{j \neq i}^{N} w_{ij} Q(v_j(t)) + r_i(t)$$

mit

$$\begin{aligned} Q &= Q_m \{1 - \exp[-e^{v} - 1) / Q_m]\} \quad mit \quad u_0 > 0 \\ Q &= -1 \quad mit \quad u_0 < 0 \\ u_0 &= -\ln[1 - Q_m \ln(1 + \frac{1}{Q_m})] \\ p &= u_0 (Q + 1) \end{aligned}$$

wobei Q eine sigmoide Funktion darstellt die aus den Hodgkin und Huxley Gleichungen hergeleitet wurde. r_i bestimmt den externen Stimulus. w_{ij} ist postiv bei excitatorischen und negativ bei inhibitorischen Verbindungen.
Der lineare Teil dieses Gleichungssystem ist an die Regelungstechnik angelehnt und ist dort als PT_2 Glied bekannt in der Form:

$$\frac{1}{\omega_0^2} \frac{d^2}{dt^2} x_a(t) + \frac{2D}{\omega_0} \frac{d}{dt} x_a(t) + x_a(t) = K_p x_e(t)$$

wobei ω_0 die Eigenkreisfrequenz und D die Dämpfung des Systems darstellt. Ein Vergleich der Koeffizienten ergibt, dass

$$\begin{aligned} \omega_0 &= \sqrt{ab} \\ D &= \frac{a+b}{2\sqrt{ab}} \end{aligned}$$

was zur Folge hat, dass D immer >1 ist. Dies wird als „Kriechfall“ bezeichnet und zeigt an, dass ein einzelnes K0 Objekt nicht schwingt. Erst durch das Zusammenschalten mehrer bekommt man ein schwingungsfähiges System.
Ergebnis der Simulation bei Stimulation durch einen Dirac Impuls mit den Parametern *a=0,22* und *b=0,72*:

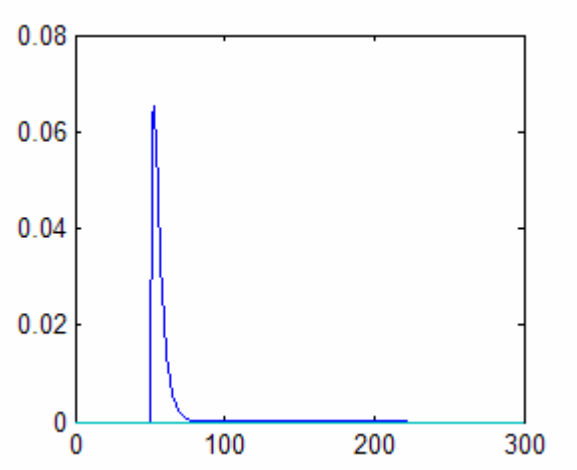


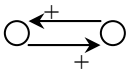
Abbildung 5: Antwort eines K0 Objekt auf einen Dirac Impuls zum Zeitpunkt 50 ms als Stimulus. Das System wird bis zu einem bestimmten Wert erregt und fällt wieder exponentiell ab.

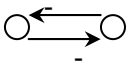
Eine Analyse des Modells zur Optimierung der Parameter findet man in [23].

4.1.2 Das KIe/KIi Objekt

Ein KIe bzw. KIi-Objekt besteht aus jeweils zwei K0 Objekten, wobei das KIe durch positive Gewichte ein sich gegenseitig erregendes, und das KIi durch zwei negative Gewichte ein sich gegenseitig hemmendes System darstellt.

Symbol:

KIe: 

KIi: 

Das KIe Objekt beschreibt die Selbsterregung einer Nervenassembly wie sie im Olfactory Bulb (OB) z.B. durch Mitralzellen stattfindet. Im Gegensatz dazu beschreibt das KIi Objekt die gegenseitige Inhibierung wie sie im OB durch Granulzellen passiert. Ein Kreis des Symbols stellt immer eine homogene Population dar und wird folgend als *Knoten* bezeichnet.
Ergebnis der Simulation des KIe bei Stimulation eines Punktes durch einen Dirac Impuls mit den Parametern $a=0,22$ und $b=0,72$.

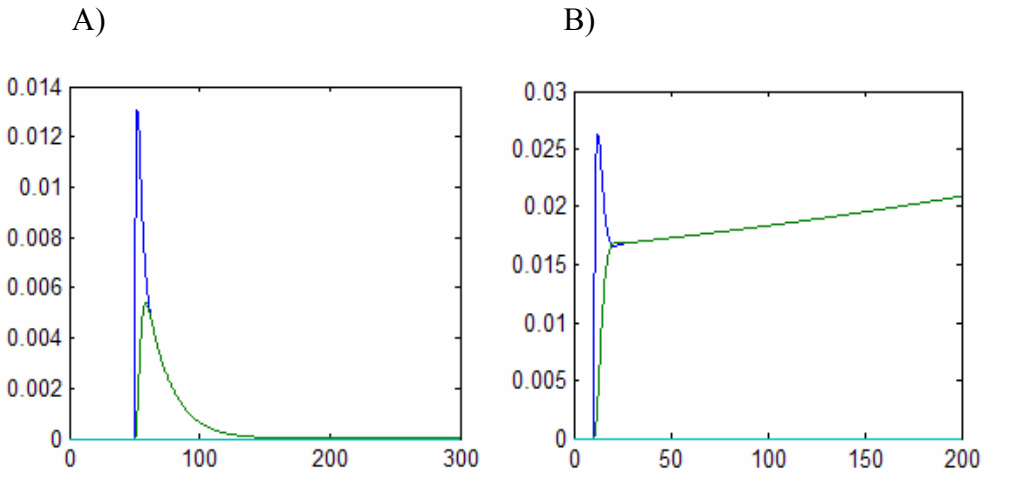


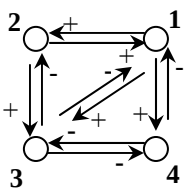
Abbildung 6: Antwort des KIe auf einen Dirac Impuls bei verschiedenen Gewichten. A) Das Gewicht zwischen beiden Knoten ist kleiner Eins. Die Erregung fällt exponentiell gegen Null. B) Das Gewicht zwischen beiden Knoten ist größer Eins. Die Knoten erregen sich gegenseitig immer mehr und steigen gegen unendlich.

Das Ergebnis des KIi Objekts ist dasselbe wie das des K0 Objekt, da der zweite Punkt durch das negative Gewicht nicht erregt wird und v aufgrund der sigmoiden Funktion auch nicht kleiner 0 werden kann.

4.1.3 Das KII Objekt

Das KII Objekt besteht aus einem KIe und einem KIi Objekt. Diese beiden sind mit einem Gewicht w_{ei} das die Erregung von inhibitorischen Neuronen durch excitatorische wieder gibt und einem Gewicht w_{ie} das die Inhibitierung der excitatorischen Neuronen durch inhibitorische wieder gibt. Ein Beispiel hierfür ist wieder der OB, wo die wechselseitige Erregung und Inhibierung durch Mitral- und Granulzellen passiert.

Symbol:



Das Vorzeichen „+“ bei einer Pfeilspitze beschreibt eine erregende, das Vorzeichen „-“eine inhibitorische Verbindung.
Durch diese Verschaltung ergeben sich insgesamt 3 Arten der Rückkopplung:

1. Gegenseitig inhibitorische Verbindung: Gewicht w_{ii} welches kleiner 0 ist.
2. Gegenseitig erregende Verbindung: Gewicht w_{ee} welches größer 0 ist.
3. Gegengekoppelte Verbindungen: Gewichte w_{ei} und w_{ie} wobei ersteres immer größer 0 zweites immer kleiner 0 ist.

Ergebnis der Simulation des KII bei Stimulation des Knoten 1 durch einen Dirac Impuls mit den Parametern $a=0,22$ und $b=0,72$:

(A) w=[2.0 1.5 -2.0 -1.0] (B) w=[0.25 1.5 -1.5 -1.8]

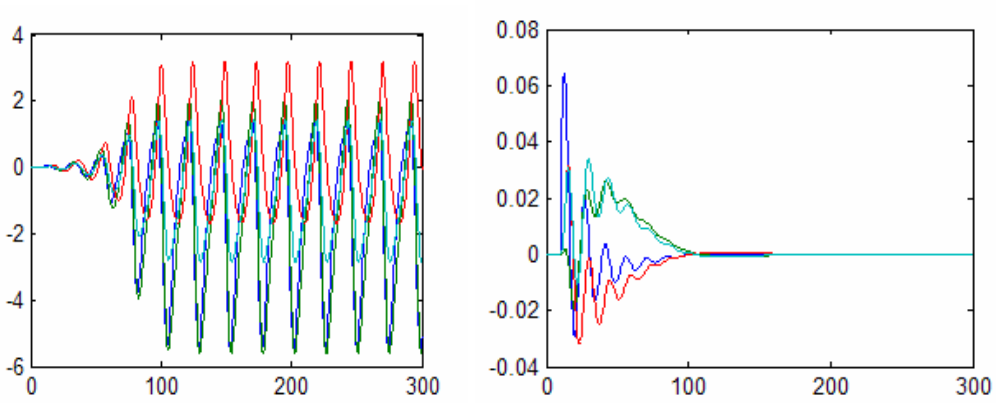


Abbildung 7: Verhalten des KII-Objektes bei verschiedenen Gewichten bei Stimulation durch einen Dirac Impuls. (A) Bei ausreichend hohen Gewichten schwingt das System auf. (B) Nach kurzem Schwingen fällt das System wieder in die Ruhelage zurück.

Je nach Wahl der Gewichte schwingt System auf oder fällt wieder zurück in einen stabilen Zustand der nicht unbedingt Null sein muss. Eine Analyse wann welcher Zustand erreicht wird ist noch nicht gelungen. Aufschluss über solches Verhalten geben Birfucationsdiagramme die hier allerdings nicht erstellt wurden.

An dieser Stelle muss erwähnt werden, dass als großer Nachteil dieses Modells die fehlende Möglichkeit einer unterschiedlichen Parametrisierung der einzelnen Populationen (Knoten) innerhalb eines KII-Objektes gesehen wird. So weiß man, dass z.B. Mitral- und Granulzellen unterschiedliche Aktivierungsfunktionen besitzen und andere Zeitkonstanten haben.

4.1.4 Das KIII Objekt

Das KIII Objekt stellt im speziellen Fall von Freeman [11] den gesamten Olfactory Tract dar mit den Bereichen Olfactory Bulb (OB), Anterior Olfactory Nucleus (AON) und Prepyriform Kortex (PC). Jeder dieser Bereiche besteht aus einem KII Objekt. Die einzelnen Bereiche werden mit lateralen Gewichten verbunden, wobei alle Gewichte positiv sind. Verzögerungen durch lange Axone werden im Modell zusätzlich durch die Totzeiten L1-L4 berücksichtigt (siehe Abbildung 8).

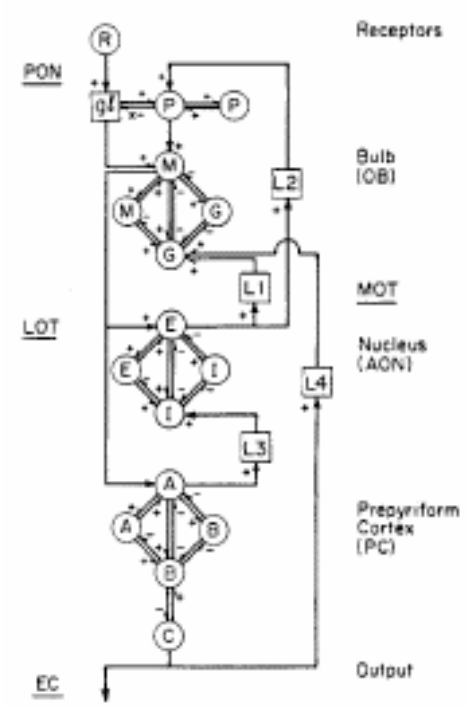


Abbildung 8: Das KIII Objekt nach Freeman [11, 22]. Stellt hier die Verschaltung des gesamten Olfactory Tract dar.

Von Kozma und Freeman wurde dieses KIII Objekt zu einem KVI Objekt erweitert [24]. Dieses soll das Zusammenspiel von verschiedenen Hirnregionen wie Hippocampus mit Olfactory System und Amygdala simulieren und besteht aus einer

Reihe von verschalteten KIII Objekten. Mit dieser Simulation wurde sogar die Steuerung eines simulierten Roboters durchgeführt.

4.2 Attraktoren und dynamisches Lernen im Modell

Schaltet man mehrere KII Objekte, wie im obigen Fall, zusammen, verhält sich das System aufgrund der hohen Dynamik chaotisch. Erhält es z.B. am Punkt *R* (siehe Abbildung 8) einen Stimulus, so werden die einzelnen Bereiche in einen Zustand höherer Schwingung versetzt – ein Attraktor bildet sich aus. Sind in den Layern mehrere Objekte vorhanden, bildet sich bei einem bestimmten Input in einem Objekt der Attraktor stärker aus. Durch Anpassen der lateralen Gewichte in diesem Layer durch einen Lernalgorithmus wird dieser Attraktor hervorgehoben.

Beispiel: folgendes Netz wird verwendet:

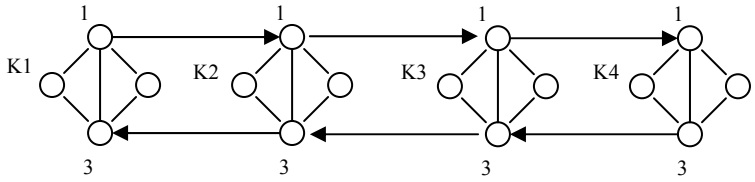


Abbildung 9: Verschaltung von vier KII-Objekten. Die unteren Verbindungen der Knoten 3 besitzen eine Verzögerung von 10 ms und sind negativ. Alle Gewichte haben den Wert 1. Verbindungen zwischen Objekten stellen im Weiteren immer „laterale“ Gewichte dar.

Vier KII Objekte, K1-K4, werden miteinander verschalten, wobei die Verbindungen der Knoten 1 jeweils ein Gewicht von +1 und die Verbindungen der Knoten 3 ein Gewicht von -1 mit einer Verzögerung von 10 ms besitzen. An jedes Objekt wird am Knoten 1 ein Rauschen moduliert. Bei t=1200 ms wird ein Stimulus mit der Länge von 200 ms und einer Amplitude von 5 angelegt. Jedes Objekt antwortet mit einem Aufschwingen um anschließend wieder in die Ruhelage zurückzufallen (siehe Abbildung 10).

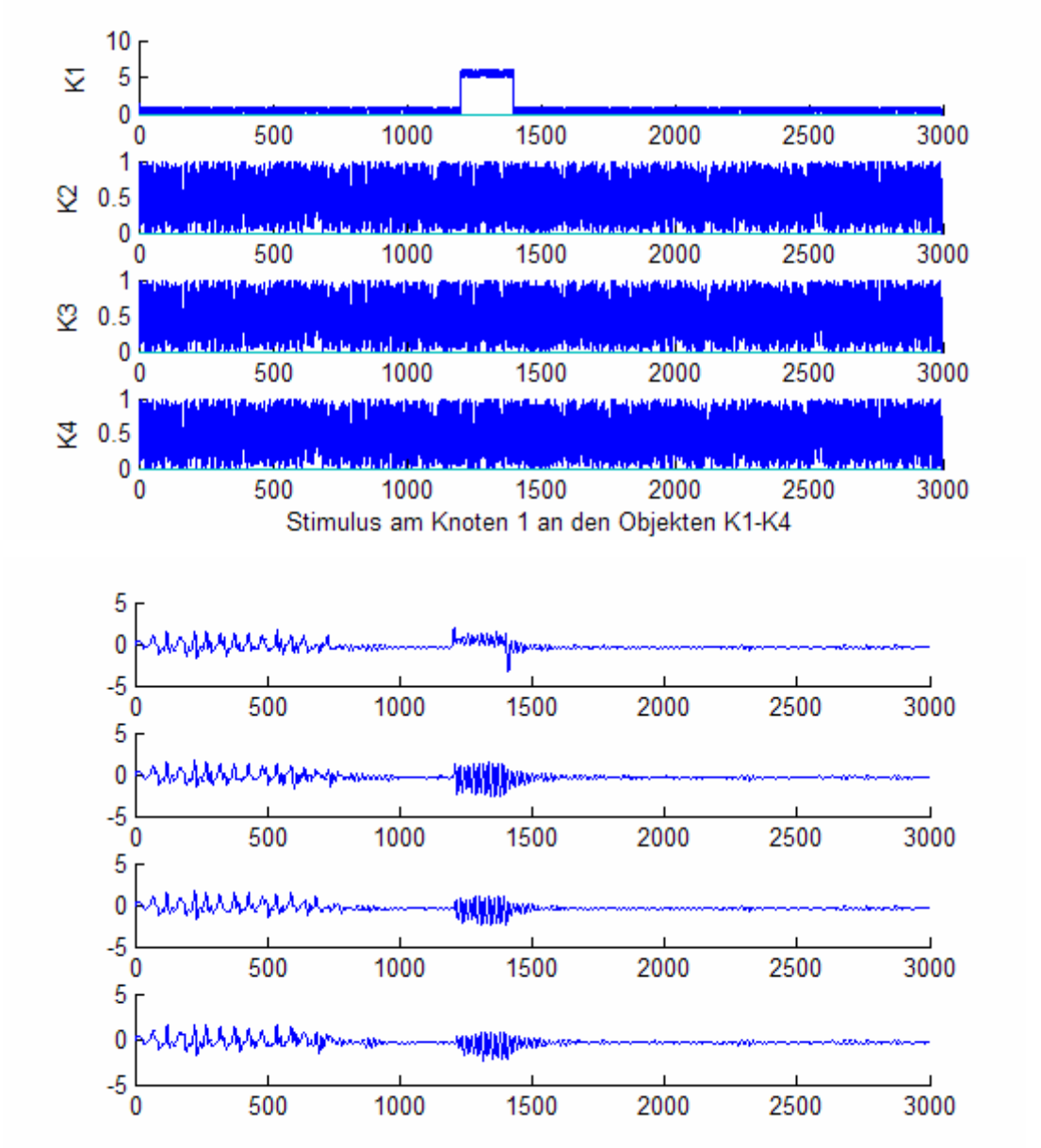


Abbildung 10: Oben: Angelegte Stimulus an den Objekten K1-K4. An die Objekte K1-K4 wurde an den Knoten 1 jeweils ein gleichverteiltes Rauschen mit der Höhe Eins angebracht, an Objekt K1 zusätzlich ein Rechteck mit der Amplitude 5 als Stimulus. Unten: Antwort der vier Objekte auf den Stimulus.

4.2.1 Lernen im K Modell

Von den im Rahmen dieser Arbeit behandelten Modellen ist das K Modell das einzige, das einen Lernalgorithmus besitzt. Dieser wurde von Kozma und Kollegen [25] übernommen und in die Toolbox integriert.

Zum Training wird am Eingangslayer (OB) ein Stimulus angelegt und das Modell für eine bestimmte Zeitspanne berechnet, dies nennt man die *aktive Phase*. Nach der aktiven Phase werden die lateralen Gewichte des 3. Layer (PC) neu berechnet. Anschließend läuft das Modell für eine bestimmte Zeitspanne ohne Stimulus, um es mit den neuen Gewichten wieder in die Ruhelage zu bringen, dies nennt man die *relax Phase*.

Sind die Gewichte festgelegt wird ein Referenzdatensatz zur weiteren Berechnung herangezogen. Mit diesem wird wieder für jeden Datensatz (aktive Phase) eine Standardabweichung σ für jeden Knoten1 jedes KII Objektes berechnet und die zugehörige Klasse zu jedem dieser σ-Vektoren gespeichert. Wieder muss nach jedem Datensatz eine relax Phase erfolgen um das System in die Ruhelage zu bringen. Nach diesem wird schließlich ein Testdatensatz herangezogen aus welchem ebenfalls aus jedem Datensatz wie beim Referenzdatensatz, die Standardabweichung berechnet wird.

Zur Klassifizierung wird schließlich aus den beiden Vektoren des Referenz- und des Testdatensatzes mit den zugehörigen σ die *k* nächsten Nachbarn berechnet und der Datensatz in diese Klasse übergeben.

4.2.2 Der Lernalgorithmus

Zuerst wird aus jedem ersten Knoten jedes KII Objekts im Layer die Standardabweichung σ berechnet. Anschließend wird aus allen erhaltenen σ die mittlere Standardabweichung *mS* berechnet. Dann wird für jedes laterale Gewicht in diesem Layer die Gewichtsänderung *dw* berechnet und zwar mit

$$dw_{ij} = \alpha(\sigma_i - mS)(\sigma_j - mS)$$

wobei α die Lernschrittweite darstellt.

5 Das Firing-Rate Modell

Im Gegensatz zum K-Modell wird in diesem Modell als Ausgabe die Frequenz einer Neuronenpopulation und nicht deren Erregtheit anhand der Membranspannung modelliert. Konnte man im vorigen Modell das Ergebnis als „durchschnittlich hervorgerufenes Potential“ (AEP – Averaged Evoked Potential) interpretieren wie dies durch ein EEG geschieht, so muss man in diesem Modell das Ergebnis als das Histogramm der Feuerungsrate der Populationen sehen. Die folgenden Erklärungen wurden zum größten Teil aus Dayan & Abbot [3] entnommen.

5.1 Herleitung des Modells

Wir nehmen ein Neuron an, das als Input den postsynaptischen Strom (PSC – Postsynaptic Current) I_s von den Inputneuronen $\boldsymbol{u}$ erhält und damit die Feuerungsrate $\boldsymbol{v}$ des Neuron steuert (Abbildung 11).

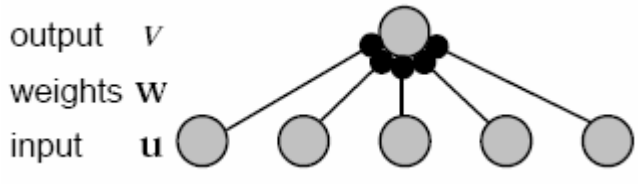


Abbildung 11: Ein vorwärts gerichtetes Netzwerk mit einem einzelnen Neuron. Durch den Input $\boldsymbol{u}$ durch die ankommenden Neuronen wird über das Gewicht $\boldsymbol{w}$, welches die synaptische Stärke der Verbindung wiedergibt, die Outputrate $\boldsymbol{v}$ als Frequenz gesteuert.

Dazu bestimmen wir als erstes I_s des Neurons:

$$\tau_s \frac{dI_s}{dt} = -I_s + \sum_{i=1}^{N} w_i u_i = -I_s + \mathrm{w} \cdot \mathrm{u}$$

wobei ***w*** und ***u*** die Matrix der Gewichte bzw. der Vektor der Inputs ist und ***w*** · ***u*** eine Matrixmultiplikation darstellt welche der Summierung über die Inputs entspricht.
Nun müssen wir aus dem bekannten Strom auf die Feuerungsrate ***v*** schließen. Dies passiert unter der Annahme, dass *Is* konstant ist und daher eine konstante Frequenz über die Funktion $F(I_s)$ ausgegeben wird. *F* wird hier die Aktivierungsfunktion genannt und ist unserem Fall wieder, wie auch schon im vorherigen Modell, eine sigmoide Funktion. Damit erhalten wir aus obiger Gleichung mit $Is = \mathbf{w} \cdot \mathbf{u}$

$$v_\infty = F(\mathrm{w} \cdot \mathrm{u})$$

und weiters

$$\tau_r \frac{dv}{dt} v = -v + F(\mathrm{w} \cdot \mathrm{u})$$

Man beachte, dass dieses Modell bereits einen Tiefpassfilter enthält, da die Änderung von *v* bei einer Änderung von I_s eine endliche Zeit benötigt.
Die Konstante τ_r gibt an, wie schnell *v* einer Änderung des Input folgen kann und den stationären Zustand erreicht, und nicht die Zeitkonstante der Membran wie man glauben könnte.
Wir erweitern das Modell nun mit der Annahme, dass das Modell aus sich gegenseitig beeinflussenden excitatorischen und inhibitorischen Populationen besteht und fassen die diese verbindenden Gewichte in der Matrix *M* zusammen. Der Input der „feed forward" Neuronen ***w*** · ***u*** ersetzen wir durch die Variable *h* und betrachten diese im Weiteren als externen Stimulus (Abbildung 12).

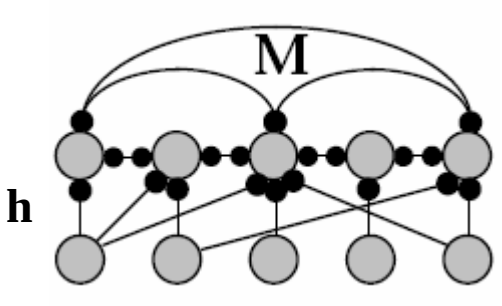


Abbildung 12: Ein rückgekoppeltes Netzwerk. Die unteren Neuronen können als externer Stimulus bzw. Rezeptorneuronen betrachtet werden. *M* beinhaltet die Gewichte der synaptischen Verbindungen.

Daraus ergibt sich ein Netzwerk, das in der Literatur als so genanntes „recurrent network“ zu finden ist und bezeichnen es hier im Weiteren als rückgekoppeltes Netzwerk. Die Modellgleichung lautet

$$\tau_r \frac{dv}{dt} = -v + F(Mv + h)$$

Aufgrund der leichteren Analyse wird für rückgekoppelte Netzwerke in der Regel angenommen, dass die Gewichtsmatrix *M* symmetrisch ist, was allerdings gegen das Gesetz von Dales verstößt. Dieses besagt, dass das postsynaptische Potential (PSP) eines Neurons entweder excitatorisch oder inhibitorisch ist, und zwar für alle Synapsen des jeweiligen Neurons. Es können die Synapsen eines Neuron also nicht einige Neuronen erregen und gleichzeitig andere inhibieren.

Obige Modellgleichung stellt die allgemeinste Form unseres Modells dar. Um dem Gesetz von Dales unter allen Umständen gerecht zu werden, werden excitatorische und inhibitorische Neuronen oft getrennt beschrieben. Dies hat zudem den Vorteil, dass die Zeitkonstante τ_r und die Aktivierungsfunktion *F* für beide Gruppen getrennt gewählt werden können.

$$\tau_E \frac{dv_E}{dt_E} = -v_E + F_E(M_{EE} \cdot v_E + M_{EI} \cdot v_I + h_E)$$

$$\tau_I \frac{dv_I}{dt_I} = -v_I + F_I(M_{IE} \cdot v_E + M_{II} \cdot v_I + h_E)$$

Wir besitzen nun vier Gewichtsmatrizen welche alle vier möglichen synaptischen Verbindungen beschreiben.
Daraus ergeben sich wieder die drei möglichen Rückkopplungen wie wir sie aus dem vorigen Modell bereits kennen:

1. Gegenseitig inhibitorische Verbindung: hier Matrix M_{II} - alle Gewichte sind kleiner 0.
2. Gegenseitig erregende Verbindung: Matrix M_{EE} - alle Gewichte sind größer 0.
3. Gegengekoppelte Verbindungen: Matrizen M_{EI} und M_{IE} wobei die Gewichte von Matrix M_{EI} immer größer 0 sind und jene der Matrix M_{IE} immer kleiner 0 sind.

Als Beispiel wurde dieses Modell von Li und Hopfield [26] ebenfalls zur Modellierung des Olfactory Bulb herangezogen. Dieses Beispiel wird im übernächsten Kapitel genauer erläutert.

5.2 Phasendiagramm und Stabilitätsanalyse

Aufgrund der Aktivierungsfunktion ist obiges Modell nichtlinear. Da nichtlineare Systeme schwer auf Ihre Stabilität zu untersuchen sind, nehmen wir an, dass sich unser Modell nur im linearen Bereich der sigmoiden Funktion bewegt. Wir schreiben daher

$$\tau_E \frac{dv_E}{dt_E} = -v_E + \lfloor M_{EE} \cdot v_E + M_{EI} \cdot v_I + h_E - \gamma_E \rfloor_+$$

$$\tau_I \frac{dv_I}{dt_I} = -v_I + \lfloor M_{IE} \cdot v_E + M_{II} \cdot v_I + h_E - \gamma_I \rfloor_+$$

γ_E stellt hier einen Schwellwert dar und ist als Hintergrundaktivität zu interpretieren. In unserem Beispiel zur Analyse bestehen die einzelnen Populationen aus einem einzigen Neuron und die Gewichtsmatrizen $M_{EE,}$ $M_{II,}$ $M_{EI,}$ M_{IE} werden daher zu einem Skalar. Wir wählen für M_{EE} = 1.25, M_{II} = 0, M_{EI} = -1, M_{IE} = 1, γ_E=-10 Hz, γ_I=10 Hz, τ_E = 10 ms und τ_I wählen wir variabel.

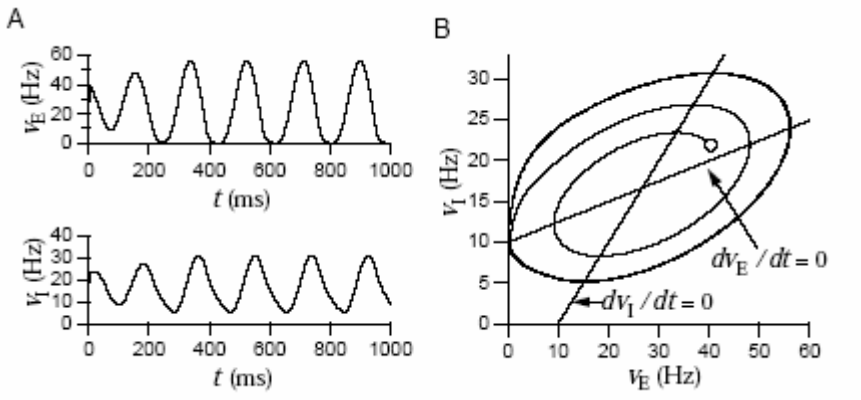


Abbildung 13: Ein Realteil der Eigenwerte ist größer Null. Beide Systeme schwingen auf und würden bei Fehlen einer Berichtigung positiv infinit. Der Fixpunkt ist somit unstabil. τ_I=50 ms.

Einen Fixpunkt erhalten wir in unserem System wenn $dv_E/dt = dv_I/dt = 0$. Zu diesem Zweck bilden wir die Jacobimatrix von vorherigem Gleichungssystem. Wird ein Realteil der Eigenwerte der resultierenden Matrix beim Variieren von τ_I größer Null, schwingt das System auf und würde bei Fehlen einer Berichtigung nach oben infinit positiv. Durch die Aktivierungsfunktion wird dies verhindert und wir erhalten einen „limit cycle“ (Abbildung 13).

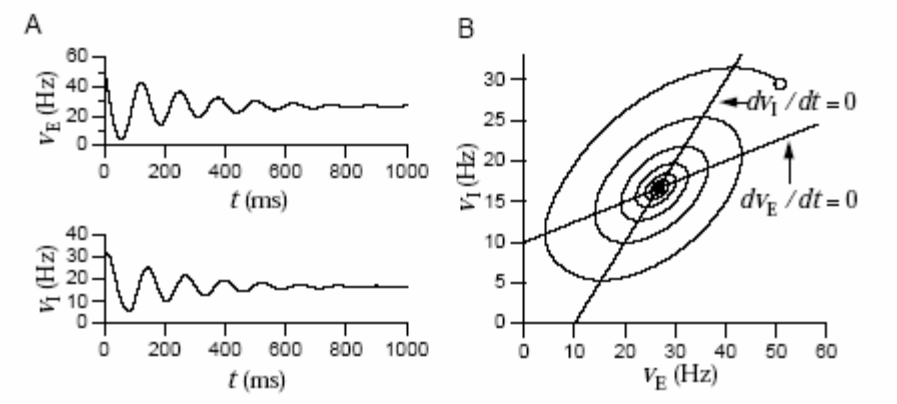


Abbildung 14: Verhalten des Systems bei stabilem Fixpunkt. (A) Einschwingvorgang jeder der beiden Teilpopulationen. (B) Phasendiagramm der beiden Populationen. Das System schwingt spiralförmig zum Fixpunkt. Der Anfang der Spirale stellt die Starwerte von v_E und v_I dar. τ_I=30 ms.

Sind dagegen alle Realteile der Eigenwerte kleiner Null besitzt das System eine stabilen Punkt (Abbildung 14).

In einem nichtlinearen System existieren normalerweise ein Menge von Fixpunkten die bei einer Veränderung in einer zyklischen Oszillation enden. Solche Stabilitätspunkte werden Hopfgabelungen (engl. „Hopf Bifurcation“) genannt [27]. Auf eine Analyse dieser Trajektoren in komplexeren Verschaltungsmustern wurde in dieser Arbeit verzichtet.

5.3 Modellierung des Olfactory Bulb

Li und Hopfield [26] verwendeten ein nichtlineares rückgekoppeltes Netzwerk zur Simulation des Olfactory Bulb. Da dieses Beispiel wiederum gut zu ersterem Modell passt, wollen wir dieses auch hier aufgreifen.

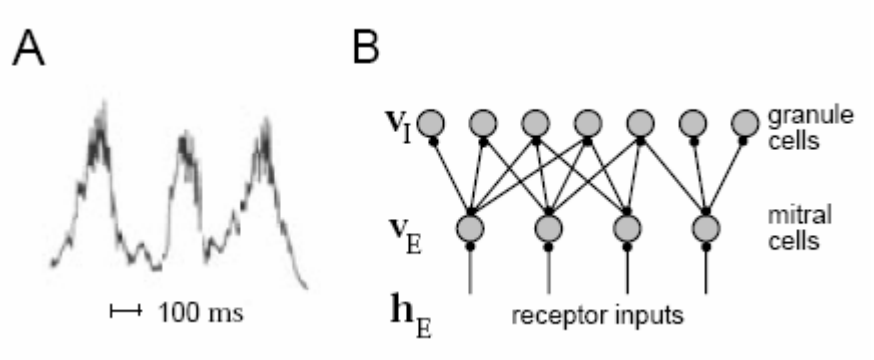


Abbildung 15: (A) Aufnahme des extrazellulären Potential im Olfactory Bulb. Die drei Wellen entstehen jeweils beim einatmen durch die Nase. (B) Verschaltungsmuster des Netzes zu Simulation.

Abbildung 15 (A) zeigt die Aufnahme des extrazellulären Potentials während der Inhalation durch die Nase. Oszillationen sind hierbei während jeder Inhalation zu sehen jedoch nicht dazwischen. Abbildung 15 (B) skizziert das Verschaltungsmuster der Simulation. Mitralzellen sind hierbei excitatorische Zellen welche Granulzellen erregen. Granulzellen dagegen sind inhibitorisch und wirken der Erregung der Mitralzellen entgegen. Mitralzellen die keinen Input erhalten werden durch die mit ihnen verbundenen Granulzellen hyperpolarisiert. Gegenseitig erregende Verbindungen bzw. gegenseitig inhibitorische Verbindungen gibt es in diesem Modell nicht.

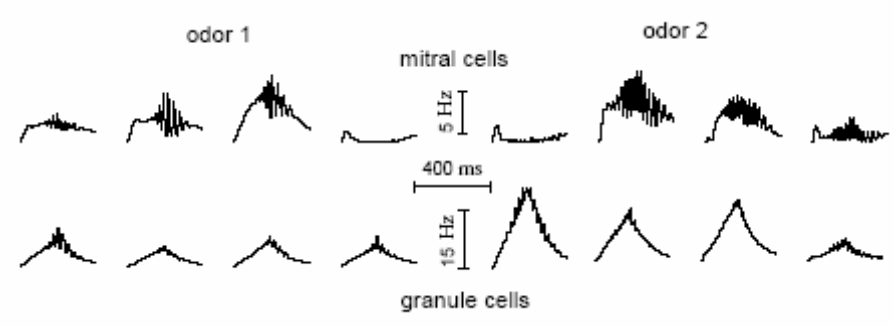


Abbildung 16: Aktivität von jeweils vier Granul- und vier Mitralzellen für zwei verschiedene Stimuli.

Abbildung 16 zeigt das Ergebnis von zwei verschiedenen Stimuli. Die obere Reihe zeigt das Verhalten der Mitralzellen, die untere die der Granulzellen. Amplitude und Phase der Zellen zueinander präsentieren jeweils einen anderen Stimuli.

6 Modell zur Beschreibung inhomogener Teilpopulationen

Dieses Modell hat ihren Ursprung in der Epidemiologie und wurde von Lajmanovich & Yorke entwickelt [28] um die Ausbreitung der Geschlechtskrankheit Gonorrhöe, im Volksmund Tripper, zu beschreiben. Im Weiteren wurde es zur Modellierung der Ausbreitung verschiedenster Infektionskrankheiten, wie z.B. Malaria, abgeändert.

Das Modell wurde ausgesucht, um damit die Regelung des Kalziumniveaus, wie in Kapitel 2 erläutert, zu beschreiben.

6.1 Gleichungssystem des Modells

Das Modell besteht aus einer Schar von linearen Differentialgleichungen erster Ordnung wobei jede Gleichung eine Population im System beschreibt.
Die Modellgleichung lautet:

$$C_i = \sum_{j \in G} w_{ji}(X_i - C_i)C_j - \alpha_i C_i$$

C_i … Kalziumniveau in Population i

X_i … Absolute Größe der Population i in unserem Fall immer 1

α_i … Abbaurate des Kalzium in Population i pro Zeiteinheit

w_{ji} … Stärke der synaptischen Verbindung von Population j zu Population i

Die mathematischen Vorraussetzungen für dieses Modell sind: $\alpha_i > 0$, $w_{ji} \geq 0$ für alle $j,i \in G$. Um das Modell leichter analysieren zu können, werden diese Bedingungen vorerst beibehalten.

Weiters wird angenommen, dass die Matrix w_{ji} irreduzibel ist. Ist dies der Fall nennt man das System *G* zusammenhängend. Ist dies nicht der Fall können die einzelnen Teilsysteme getrennt berechnet werden.

Gleichgewicht und Stabilität des Systems

Zur Bestimmung der Stabilität werden die in Kapitel 3 angeführten allgemeinen Bedingungen herangezogen.

Def.1: $B = \prod_{i \in G} [0, X_i]$

Def.2: A sei die Jacobi-Matrix der rechten Seite der Modellgleichung für $C = 0$

Damit gilt: $A=(a_{ij})$ sei die quadratische Matrix mit

$$a_{ij} = w_{ji} X_i \text{ für } i \neq j$$

$$a_{ii} = w_{ii} X_i \text{ für alle } i \in G$$

Def.3: s(A) sei dann der Stabilitätskoeffizient von *A*

Für das Verhalten des Modells gibt es dann genau folgende zwei Möglichkeiten:

1. $s(A) \leq 0 \Rightarrow$ Lösung $Y^*=0$ ist global asymptotisch stabil in B.
2. $s(A) > 0 \Rightarrow$ es existiert eine konstante Lösung C^*, mit $0<Ci^*<Xi$ für alle $i \in G$, die global asymptotisch stabil ist in $B\backslash 0$ und 0 ist instabil.

Die Ergebnisse der Simulation dieses Modell sind dann in Kapitel 8.3 zusammengefasst. Hier werden die Stabilitätsbedingungen auch leicht verständlich.

Da die Bedingung des Modells, $w_{ji} \geq 0$, in unserem Fall nicht erfüllt wird, da inhibitorische Synapsen durch ein negatives Gewicht dargestellt werden, wird im nächsten Kapitel das Modell etwas angepasst.

6.2 Modifiziertes Modell

Der erste Gedanke ist, dass durch die möglichen inhibitorischen Synapsen auch negative Gewichte im Modell vorkommen müssen. Weiters wird das Kalziumniveau bei Ausbleiben einer Aktivität als in Ruhelage und somit am Niveau 0 liegend angenommen. Wird die Population dann inhibiert, kann C_i auch einen negativen Wert annehmen. Hier zeigt sich das erste Problem, da bei zu großen negativen synaptischen Gewichten der Zustand des System instabil wird und C_i negativ infinit wird.

Abhilfe schafft die Annahme, dass

$$C_i = \sum_{j \in G} w_{ji}(X_i - C_i)C_j - \alpha_i C_i$$

$$C_i = -5 \text{ für alle } C_i \leq -5$$

Im Modell nach Traub [16, 17] spielt Kalzium eine weitere wichtige Rolle. Dieses Modell basiert auf den Hodgkin & Huxley Gleichungen und modelliert zusätzlich Kalziumkanäle. Dieses Modell zeigt, dass in Pyramidenzellen des Kortex nach einem Aktionspotential dieses ein Backpropagationspotential auslöst welches von der Soma rückwärts zu den Dendriten läuft. Durch dieses Aktionspotential wird die Zellmembran an den Dendriten stark depolarisiert, Kalzium strömt ein und wirkt der maximalen Frequenz der Zelle entgegen.
Um dieses Phänomen ins Modell aufzunehmen wäre es denkbar das oben beschriebene „Firing Rate"-Modell mit diesem Populationenmodell zu verbinden um zuerst die Frequenz zu modellieren und mit dieser dann das damit verbundene Kalzium zu regulieren, und umgekehrt.

7 Die Toolbox - Aufbau und Funktionsweise

Um die oben angeführten Modelle in vernünftiger Art und Weise simulieren zu können ohne für jede Fragestellung ein neues Programm implementieren zu müssen, war es nahe liegend, eine Toolbox zu entwerfen, mit der es möglich ist, beliebige Netze bzw. Verschaltungsmuster aus inhibitorischen und exzitatorischen Populationen zu gestalten und diese dann mit den verschiedenen Modellen simulieren zu können. Ziel war es dabei, das Netzwerk beibehalten zu können und durch Änderung von wenigen Parametern, die die Modelle unterscheiden, diese einfach untereinander austauschen zu können. Bei der Implementierung wurde stark auf das Modell von Freeman [11] Bezug genommen, deshalb sind auch viele Analogien, wie zum Beispiel gleiche Variablennamen und Funktionsgleichungen, im Programm zu finden.
Zur Implementierung wurde MatLab® gewählt, da dieses eine sehr mächtige Entwicklungsumgebung bereitstellt, und durch das Konzept der Matrizenverarbeitung für Simulationen sehr gut geeignet ist.

Der Aufbau einer Simulation gliedert sich im Wesentlichen in vier Teile. Das ist zum Ersten die Verschaltung des Netzes mit den einzelnen Objekten. Dazu ist die Bekanntgabe deren Gewichte zwischen den Knoten innerhalb eines Objektes und den lateralen Verbindungen mit Gewichten und Verzögerung zwischen den Objekten notwendig. Der zweite Teil ist die Erstellung von den nötigen Stimuli und deren Anbindung an die jeweiligen Knoten. Im dritten Teil wird die Simulation mit dem ausgewählten Modell durchgeführt, und im vierten Teil werden die Ergebnisse dargestellt.

7.1 Erstellen des Netzes

Motiviert durch das Verschaltungsmuster des K-Modells von Freeman [11, 13, 22] besteht in der Toolbox ein Objekt als kleinste Einheit aus vier Knoten, wobei die

interne Verschaltung dieser Knoten durch die mit ihnen verbundene Gewichtsmatrix bestimmt wird. Dabei werden vier Arten der Verbindung unter den Knoten unterschieden: zwischen zwei erregenden Knoten *wee,* zwischen zwei inhibierenden Knoten *wii,* von einem erregenden zu einem inhibierenden Knoten *wei,* und von einem inhibierenden zu einem erregenden Knoten *wie.* Die daraus resultierende Matrix besitzt die Form

```
0   wee wie wie
wee   0 wie   0
wei wei   0 wii
wei   0 wii   0
```

welches dem Verschaltungsmuster (Kapitel 4.1) gleichkommt:

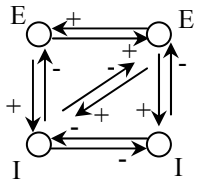


Abbildung 17: Verschaltung eines Objekts innerhalb der Toolbox

Zu lesen ist die Matrix dabei in folgender Weise: der Knoten mit dem Zeilenindex *i* bekommt einen Input vom Knoten mit dem Spaltenindex *j* mit dem Gewicht und Vorzeichen w_{ij}.

Wir erhalten aus obiger Matrix vier verschiedene Gewichte die als Vektor der Funktion *addKII.m* beim Aufruf übergeben werden.
Möchte man nun dieses Verschaltungsmuster ändern, hat dies in der Funktion *addKII.m* zu erfolgen wenn es von Dauer sein soll, oder man übergibt mittels des Parameter *'wMatrix'* eine neue 4x4 Matrix in welcher jedes Gewicht separat für jede Verbindung zweier Knoten gesetzt werden kann. Dabei wird die Variable die sonst die vier nötigen Gewichte enthält unwirksam.

Hinweis: Werden Gewichtsvariablen mit dem Gewicht 0 belegt, kommt dies einem Aufheben der entsprechenden Verbindungen gleich. Auf diese Weise kann auch nur ein einzelner Knoten verwendet werden.

Beispielcode:

```
net=createModel(); % eine struct Variable wird initialisiert
ptau=[1 50 1 50];     % vom Modelltyp abhängige Zeitkonstanten
% Gewichtsparameter wP = [wee, wie, wie, wii]. Erst durch Vorzeichen wird
Erregung / Inhibierung bestimmt.
wP=[0.4 0.8 -1.2 -0.9];
% Ein Objekt wird zum Netz hinzugefügt.
[net P1]=addKII(net, wP, ptau);
[net P2]=addKII(net, wP, ptau);
```

Bei vielen Anwendungen ist es zudem notwendig einen Startwert vorzugeben. Dies geschieht mit dem Parameter *'inity1'* welcher ebenfalls optional ist und bei Weglassen mit 0 initialisiert wird. Der Übergabewert ist entweder ein Skalar wenn für alle vier Knoten derselbe Wert gilt, oder ein 1x4 Vektor.

Über die Variable *P* wird das jeweilige Objekt referenziert. Auf einen bestimmten Knoten innerhalb des Objekt *P* wird mit *P.node{x}* zugegriffen wobei *x* einen Integerwert zwischen 1 und 4 darstellt.
Die lateralen Verbindungen zwischen den Objekten werden durch die Funktion *connectKIII.m* hergestellt. Diese Funktion benötigt als Parameter das Netz in dem die Verbindung erstellt werden soll, den Quellknoten, den Zielknoten, das Gewicht der Verbindung und eventuell eine Verzögerung des Signals.

Beispielcode:

```
net = connectKIII(net, 'src',P1.node{1},'dest',P2.node{1},'weight ', 0.9,'delay',0);
```

Hierbei wird wiederum in der Structvariablen *net* eine Liste mit den angegebenen Verbindungen aufgebaut. Wird dieselbe Verbindung zweimal angegeben, kommt dies einer Verdoppelung des Gewichts gleich sofern die Verzögerungszeit dieselbe ist, d.h. die Verbindung wird tatsächlich zweimal abgearbeitet. Es ist auch möglich, eine Verbindung zwischen zwei Knoten innerhalb desselben Objekts zu erstellen. Dies macht dann Sinn, wenn eine Verzögerung zwischen zwei Knoten innerhalb des Objekts bestehen soll.
Parameter mit Präfix *'src'* oder *'dest'* sind nicht an eine Reihenfolge wie sie in der Funktion bekannt gegeben werden gebunden. Die Parameter *'delay'* und *'weight'* sind optional und werden bei Weglassen mit '0' bzw. '1' initialisiert.

Architektur:

Die Architektur beschränkt sich in diesem Teil auf die Strukturvariable *net*. Hier werden alle notwendigen Daten mit den oben beschriebenen Funktionen gesammelt.

Die Strukturvariable *net*:

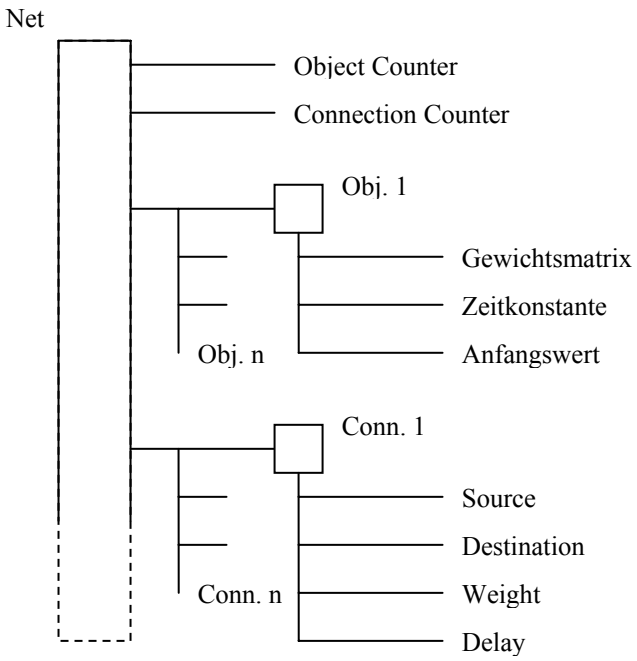


Abbildung 18: Aufbau der Strukturvariable *net*. Alle Information die zur Erstellung des Netzes und späteren Abarbeitung benötigt werden, werden hier gesammelt.

Abbildung 18 zeigt den Aufbau der Strukturvariablen *net*. Im Element *Object Counter* ist die Anzahl der in das Netz eingefügten Objekte enthalten, im Element *Connection Counter* die Anzahl der bestehenden lateralen Verbindungen. Bei jedem Aufruf der Funktion *addKII.m* bzw. *connectKIII.m* werden die beiden Elemente incrementiert und das Objekt bzw. die Verbindung eingefügt. Jedes Objekt besitzt weiters Kenntnis über die Zeitkonstante und den Anfangswert jedes Knoten und die Gewichte unter den Knoten. Jede Verbindung besitzt Kenntnis über die Quelle, das Ziel und dem Gewicht und der Verzögerung des transportierten Signals. Quelle und Ziel sind immer als Vektor in der Form *[idxObjekt, nrKnoten]* hinterlegt.

Implementierung:

Die Sicht der Implementierung ist stark datenorientiert und stehen für das Netz im Vordergrund. Der Aufbau ist entscheidend für eine spätere effiziente Abarbeitung und Bedienung.

Funktionen die das Netz bearbeiten, wird die Variable *net* übergeben, die Funktion bearbeitet diese und gibt sie wieder zurück. Das ist notwendig da MatLab Variablen nicht mit *„call by referenc“* übergibt, wenn das nicht explizit angegeben ist. Es wurde darauf geachtet, dass Parameter eine *Default*-Initialisierung erhalten und diese optional angegeben werden können. Die Variablen benötigen dabei immer ein „Präfix“ damit sie nicht von einer Reihenfolge abhängig sind. Das wurde auf folgende Art und Weise gelöst:

```
a = strmatch('wmatrix', lower(strvcat(varargin{1:2:end})), 'exact');
    if length(a) == 1
      a=(a-1)*2+1;
      wMat = varargin{a+1};
    else                          % Defaulteinstellung
      wMat =  [0   kee kie kie;
               kee 0   kie 0;
               kei kei 0   kii;
               kei 0   kii 0];
    end
```

Obiges Beispiel zeigt die Auswertung des optionalen Parameter *‘wMatrix’* der Funktion *addKII.m*. Der Befehl *varargin* enthält alle Übergabeparameter die nicht explizit im Funktionskopf angegeben sind. Mit *varargin{1:2:end}* werden alle Präffix gefiltert. Die Funktion *strvcat* erstellt eine Stringmatrix mit den erhaltenen Präfixen, *lower* konvertiert sie zu Kleinbuchstaben. Die Funktion *strmatch* sucht schließlich den angegebenen Präfix und gibt die zugehörige Position zurück. Wurde genau eine Position gefunden, wird die nachfolgende Variable als Gewichtsmatrix übernommen, ansonsten wird ein Defaultwert genommen.

Die Implementierung von optionalen Parametern wurde für die gesamte Toolbox so gewählt.

Wie bereits erwähnt, sind für die Indizierung eines Objektes immer der Index des Objektes und die Nummer des Knoten notwendig. Um eine spätere Indizierung möglichst einfach zu gestallten, wird von der Funktion *addKII.m* ein Strukturelement

zurückgegeben, in der Form, dass für jeden Knoten immer Index und Nummer des Knoten als Vektor gespeichert sind.

Also zum Bespiel:
```
KII.node{1}=[KII.idx, 1];
KII.node{2}=[KII.idx, 2];
```
usw.

7.2 Bestimmen der Stimulus

Die meisten dynamischen Systeme besitzen, je nach Ordnung und Komplexität eine Vielzahl von Attraktoren. Um diese ausfindig zu machen beziehungsweise deren Verhalten zu prüfen ist es oft notwendig die Erregung des Systems zu verändern. Dazu muss ein Stimuli generiert und dessen Ort und Zeitpunkt bekannt gegeben werden. Vorweg ist anzumerken, dass die meisten Systeme, und auch die hier verwendeten, Null als astabilen Gleichgewichtspunkt besitzen. Dies hat zur Folge, dass in jeder Simulation zu Beginn ein Puls gesetzt werden muss, der das System „startet“.

Ein Stimuli wird mit der Funktion *addStimulus.m* hinzugefügt. Die benötigten Parameter sind wiederum, in der angeführten Reihenfolge, das Netz, der stimulierte Knoten, der Stimuli selbst und der Zeitpunkt ab dem er hinzugefügt wird.

Beispielcode:

```
S1=ones(1,20/dt)*1;   % Rechteck
S2=(sin((0:dt:30)./0.5)+1)*1;        % Sinus mit Amplitude 0-2
net = addStimulus(net, P1.node{1}, S1, 10);
net = addStimulus(net, P1.node{1}, S2, 20);
```

In diesem Beispiel wird der Stimuli *S1* dem Knoten *P1.node{1}* zum Zeitpunkt 10 ms hinzugefügt. Der Stimuli *S1* ist in diesem Beispiel ein Rechteck mit 20 ms Dauer. Die Länge des Stimuli wird in diesem Beispiel durch die Schrittweite des Integrators berechnet. Dies ist generell ratsam, da die Dauer des Stimuli somit unabhängig von der Schrittweite der Berechnung konstant bleibt. In Zeile zwei wird ein sinusförmiger Stimuli zum selben Knoten hinzugefügt. Die beiden Stimuli überlagern sich 10 ms und addieren sich in dieser Zeit (Abbildung 19).

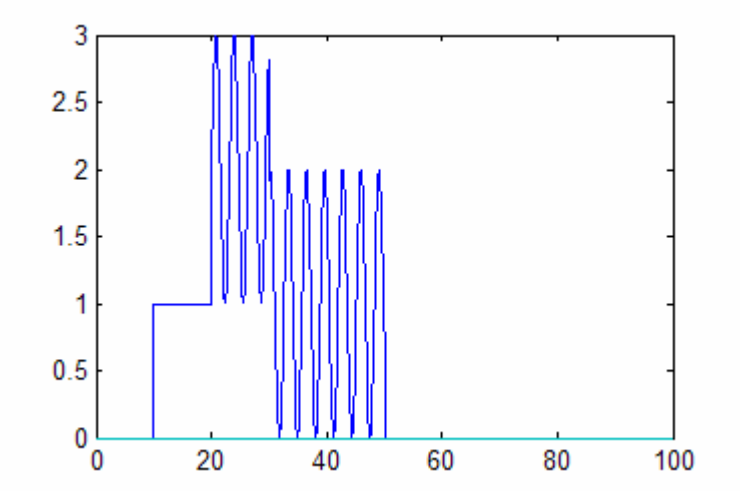


Abbildung 19: Stimuli am Knoten 1. Die ersten 10ms der Rechteck mit Amplitude 1, dann 10ms die Überlappung mit der Sinusfunktion und anschließend die verbleibenden 30 ms der Sinusfunktion.

Vorsicht ist geboten, dass die Dauer des Stimuli nicht die angegebene Integrationszeit des Modells übersteigt. Die Fehlermeldung „*??? Index exceeds matrix dimensions"* wäre die Folge.

Architektur:

Die Stimulus werden in derselben Form wie die Verbindungen des Netzes hinterlegt. Das Element *Stimuli Counter* gibt wieder die Anzahl der dem Netz hinzugefügten Stimuli an. Struktur:

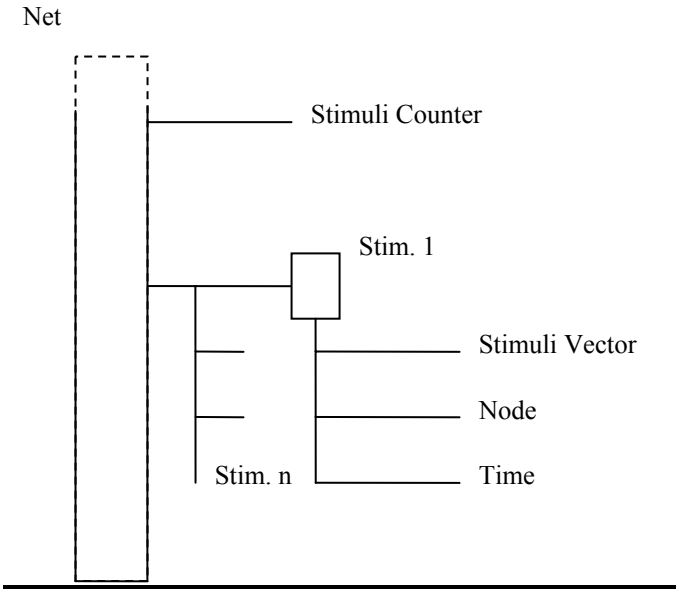


Abbildung 20: Die Struktur des Stimuli ist Teil der Strukturvariablen *net.*

7.3 Berechnen des Modells

Nun kommen wir zum „Herz“ der Toolbox, dem Berechnungsteil. Angestoßen wird dieser durch die Funktion *runModel.m.* Zwingende Parameter für diese Funktion sind wieder, genau in dieser Reihenfolge, das Netz *net*, die Schrittweite *dt* für den Integrator, und die gesamte Berechnungsdauer *tspan* über die integriert werden soll. Die Wahl des Modells mit der das Netz berechnet wird ist optional. Wird der Parameter *‘solver’* weggelassen, wird automatisch das erste Modell *S1* genommen, das in unserem Fall das K-Modell von Freeman ist (Kapitel 4).
Weitere optionale Parameter sind die sigmoide Funktion *‘sigm’* die als Zeiger auf die jeweilige Funktion übergeben wird. Da Steilheit, Höhe etc. dieser Funktion das Ergebnis der Modelle in der Regel stark beeinflussen, ist dem Benutzer die Möglichkeit gegeben, eigene Sigmoide zu generieren und diese möglichst einfach einzubinden. Hier ist jedoch zu beachten, dass es nicht für jedes Modell Sinn macht, in unserem Fall für Modell 3 das Populationenmodell, eine sigmoide Funktion vorzugeben. Durch die Struktur des Programms würde diese allerdings trotzdem in die Berechnung miteinbezogen werden.
Wird keine sigmoide Funktion übergeben wird modellabhängig die jeweils „typische“ verwendet.
Der Parameter *‘alpha’* ist nur bei der Verwendung von Modell 3 relevant und gibt die Verfallsrate im Populationenmodell bekannt. *‘alpha’* kann entweder ein Skalar, wenn für alle Knoten derselbe Wert gelten soll, oder ein Vektor mit 4 Elementen sein, wenn jeder Knoten eine verschiedene Verfallsrate besitzen soll.
In manchen Fällen kann es nötig sein die Ergebnisse einer Berechnung als Startwerte für einen nächsten Durchlauf zu übergeben. Hierfür ist der Parameter *‘hist’* zuständig. Dabei werden der letzte errechnete Wert als Startwert, und die Werte aus lateralen Verbindungen welche eine Verzögerung besitzen, für die neue Berechnung übernommen.

Beispielcode:

```
dt=0.1; % Schrittweite in ms
tspan=100; % gesamte Zeitspanne der Berechnung in ms
R = runModel(net, dt, tspan, 'solver', 'S2', 'sigm', @sigm2);
```

Anschließend wird die Datenlänge zur Berechnung des Modells aus *tspan* und *dt* ermittelt. Die Stimuli werden zu jedem Knoten an die angegebene Zeitstelle gespeichert und der Datenvektor wird um die Länge, die durch die Verzögerungszeiten der lateralen Verbindungen entstehen erweitert. Das Resultat ist eine Datenstruktur, die die Ergebnisse des Integrators, die Stimuli zu jedem Knoten und die Ergebnisse der lateralen Berechnungen jeweils für jeden Knoten für jeden Zeitpunkt in einem Vektor enthält.
Die Struktur der Ergebnisvariablen *R*:

R.KII(i).y1: Beim Modell 1 enthält diese Variable das Ergebnis nach dem zweiten Integrationsschritt, also das Endergebnis. Beim Modell 2 das Ergebnis der ersten Modellgleichung, also das Ergebnis des Firingrate-Modells. Beim Modell 3 das Ergebnis des Populationenmodells.

R.KII(i).y2: Beim Modell 1 enthält diese Variable das Ergebnis nach dem ersten Integrationsschritt, also die erste Ableitung des Endergebnisses. Beim Modell 2 steht hier das Ergebnis der zweiten Modellgleichung, also das des Populationenmodells. Für Modell 3 wird diese Variable nicht benötigt und enthält somit lauter Nullen.

R.KII(i).stim: Die errechneten Werte der Stimuli für jeden Zeitpunkt.

R.KII(i).latCa und *R.KII(i).latInp:* Die errechneten Werte der lateralen Verbindungen für jeden Zeitpunkt, wobei x.*latCa* nur für Modell 2 benötigt wird da hier die lateralen Gewichte für jede Modellgleichung separat berechnet werden müssen.

R.t: der Zeitvektor.

Der Integrator arbeitet in jedem Zeitschritt ein gesamtes Objekt in einem Rechengang ab und wird daher für jedes Objekt nur einmal aufgerufen. Als numerisches Integrationsverfahren wurde das Runge-Kutta Verfahren gewählt.

Es wurde bereits erwähnt, dass die Toolbox drei Modelle zur Berechnung besitzt, die durch den Parameter *'solver'* gewählt werden können. Diesem werden für die 3 Modelle die Optionen *S1 – S3* zugeteilt:

Modell 1 (Option *S1*): Stellt das oben beschriebene Modell 2. Ordnung nach Freeman aus Kapitel 4 dar.
Modell 2 (Option *S2*): Beinhaltet das Firingrate-Modell aus Kapitel 5
Modell 3 (Option *S3*): ist schließlich das Populationenmodell aus Kapitel 6.

Architektur:
Abbildung 21 zeigt die Abhängigkeiten der einzelnen Teile im Kern. Der Integrator ist mit dem ausgewählten Modell verknüpft und erhält die aufbereitete Netzstruktur. Nach jedem Zeitschritt werden die lateralen Verbindungen neu berechnet und die Ergebnisse in der Ergebnisstruktur abgelegt.

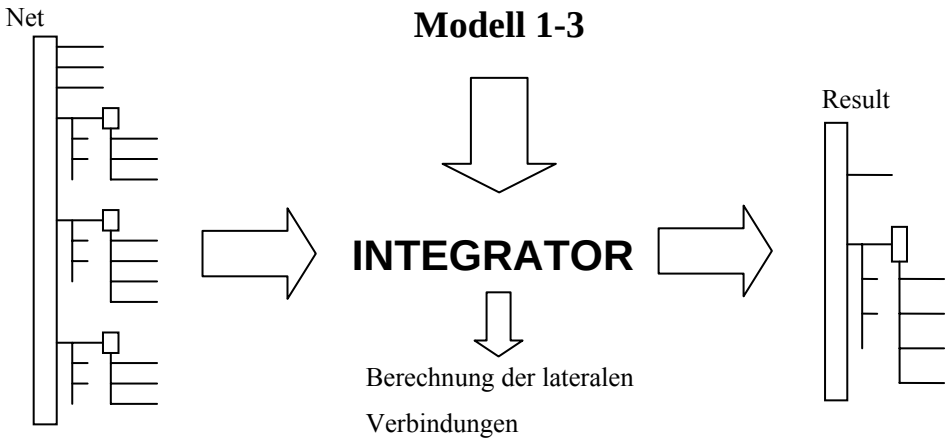


Abbildung 21: Aufbau des Kern. Der Integrator ist mit dem angegebenen Modell verknüpft, erhält die aufbereitete Netzstruktur und befüllt in jedem Zeitschritt die Ergebnisstruktur. Weiters werden nach jedem Zeitschritt die lateralen Verbindungen berechnet.

Als nächstes wollen wir die Berechnung in einem Zeitschritt genauer betrachten.

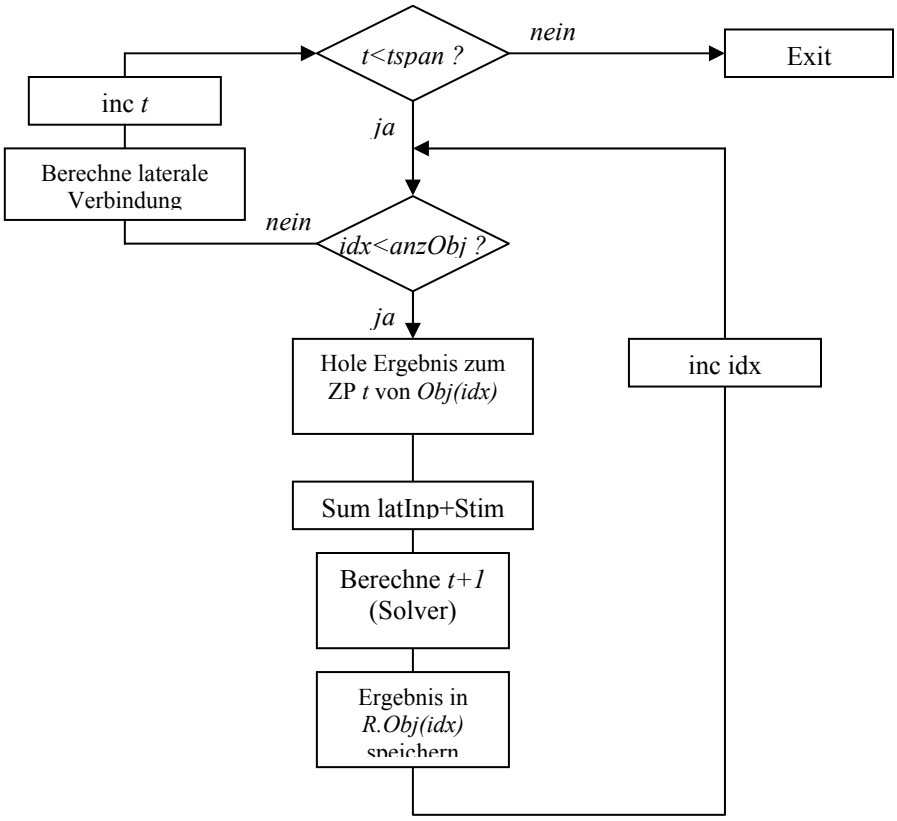


Abbildung 22: Ablauf der Berechnung in einem Zeitschritt.

Abbildung 22 zeigt den Ablauf der Berechnung. Solange *t<* als die gesamte Integrationszeit wird die Objektliste aus dem Netz abgearbeitet. Dabei wird das Ergebnis zum Zeitpunkt *t* für das jeweilige Objekt aus der Ergebnisvariablen geholt. Als nächstes werden alle Werte die von „Außen“ kommen, wie laterale Inputs und Stimulus, vorberechnet und dann dem Integrator übergeben. Dieser berechnet den Wert des Objekts für den Zeitpunkt *t+1*. Sind alle Objekte abgearbeitet werden *alle* Werte der lateralen Verbindungen für den Zeitpunkt berechnet und den Objekten zur Verfügung gestellt. Der nächste Zeitschritt kann berechnet werden.

Implementierung:

Bei der Implementierung ist hier der kritischste Punkt. Jeder Befehl wird je nach Integrationszeit zigtausend Mal aufgerufen. Es sind daher alle Berechnungen soweit wie möglich, außerhalb der Schleife durchzuführen.

```
for t=1:endT-1                  % für jeden Zeitpunkt
    for i=1:net.cntKII          % für jedes Objekt
```

```
            prevCa=R.KII(i).y1(:,t);
            Caj = net.KII(i).weight * R.KII(i).y1(:,t) + ...
                  R.KII(i).latInp(:,t) + R.KII(i).stim(:,t);
            [y1] = Solver3(net.KII(i), dt, prevCa, Caj, alpha);
            R.KII(i).y1(:,t+1)=y1;
        end;
        R = calcLatInput(net, R, dt, t, delay, sgm);
    end;
```

Die Berechnung der gewichteten Verbindungen erfolgt außerhalb des Solvers. Damit müssen weniger Parameter übergeben werden.

Wir wollen uns jetzt noch die Berechnung der lateralen Verbindungen ansehen.

```
for i=1:net.cntConnect
    idxSrc=net.connect(i).src(1); nodeSrc=net.connect(i).src(2);
    idxDest=net.connect(i).dest(1); nodeDest=net.connect(i).dest(2);
    weight=net.connect(i).weight;
    R.KII(idxDest).latInp(nodeDest, t+delay(i)) = ...
                        R.KII(idxDest).latInp(nodeDest, t+delay(i))...
                        + sgm(R.KII(idxSrc).y1(nodeSrc, t)).*weight;
end;
```

Für jede Verbindung die in der Liste im Netz angeführt ist, wird das Ergebnis vom Zeitpunkt *t* mit dem Gewicht der Verbindung multipliziert und in den Vektor *latInput* des Zielknotens zum Zeitpunkt *t+delay* dazu addiert.

Abschätzung der Rechenzeit:

Um zu ermitteln welche Teile in einem Modell welche Auswirkungen auf die Rechenzeit haben, wollen wir dazu eine Formel zur Abschätzung geben.

$$\frac{Z(2{,}5O + 0{,}6C) + 10}{R} = T$$

Z … Anzahl der Zeitschritte

O … Anzahl der Objekte im Netz

C … Anzahl der lateralen Verbindungen

R … Performanz des Rechners (z.B. für Pentium 1,5 GHz ca. 10.000)

T … gesamte Rechenzeit in sec

Anzahl der Objekte und Anzahl der lateralen Verbindungen sind linear zur berechnenden Zeitdauer und sind Hauptverantwortlich für die Berechnungsdauer.

7.4 Darstellen der Ergebnisse

Um die Ergebnisse der Variablen *R* (siehe oben) aus der Berechnung einigermaßen komfortabel darzustellen, kann man die Funktion *plotResult.m* verwenden. Diese gibt die Berechnung von jedem Knoten jedes Objektes aus, wobei für jedes Objekt ein Subplot generiert wird und alle 4 Knoten in einem Subplot dargestellt werden. Dargestellt werden dabei die Variablen *.y1*, *.y2* und *.stim* der Ergebnisvariable *R*. Dabei wird für jede dieser Variablen ein eigenes Fenster geöffnet. Hinweis: die Modelle 2 und 3 beinhalten in der Variable *y2* kein vernünftiges Ergebnis da diese nur für das erste Modell zur Speicherung der ersten Ableitung benötigt wird.

Die Parameter der Funktion :

Zwingende Parameter:
net: Das berechnete Netz. Hier wird nur die Anzahl der Objekte ausgelesen..

R: Das Ergebnis der Berechnung. Zwingend.

Optionale Parameter:
what: Mit diesem Parameter kann eine der Teilergbnissvariablen *y1* oder *y2* oder *stim* ausgewählt werden. Bei mehrmaligem Aufruf der Funktion mit diesem Parameter plotet die Funktion das Ergebnis immer in dasselbe Fenster.

merge: Dieser Parameter gibt an ob aus den Ergebnissen der 4 Knoten die Summe mit *sum* oder ein Durchschnitt mit Option *ave* gebildet werden soll. Diese Option kann nützlich sein um das Ergebnis z.B. als EEG zu interpretieren, welches ebenfalls den Durchschnitt mehrer Populationen darstellt.

Beispielcode:

```
% Plotet das Ergebnis der Variablen y1 der Summe der 4 Knoten in einem Objekt
aller Objekte
plotResult(net, R, 'what', 'y1', 'merge', 'sum');
```

An dieser Stelle noch ein Tipp: da es in der Toolbox verschiedene sigmoide Funktionen gibt, ist es oft hilfreich vor Gebrauch deren Verlauf darzustellen. Dies kann man leicht mit folgenden zwei Zeilen realisieren:

```
% plotet sigmoide Funktion
x=0:0.01:1;
plot(x, sigm2(x));
```

Man muss nur wissen in welchem Bereich sich der lineare Teil der Funktion bewegt, in diesem Fall zwischen Null und Eins.

7.5 Gegenüberstellung zu anderen Simulationsumgebungen

Simulink® [29] ist eine graphische Entwicklungsumgebung zur Simulation jeglicher Art von dynamischen Systemen. Es ist integriert in MatLab und alle Funktionen von MatLab stehen daher bei einer weiteren Verarbeitung von Daten zur Verfügung. Ein Vergleich mit einem derart großen und umfangreichen Softwarepaket ist nicht leicht. Anhand von einem einfachen Beispiel soll es trotzdem versucht werden.
Das einfachste Modell in der Toolbox ist Modell 3 da es linear ist. Wir wählen daher als Beispiel die Ausbreitung einer Infektion in einer Population da das Ergebnis evaluiert und das Modell in Simulink noch relativ einfach zu erstellen ist.

Modellgleichung:

$$\frac{dy(t)}{dt} = ky(t)[1 - y(t)]$$

Realisierung in Simulink:

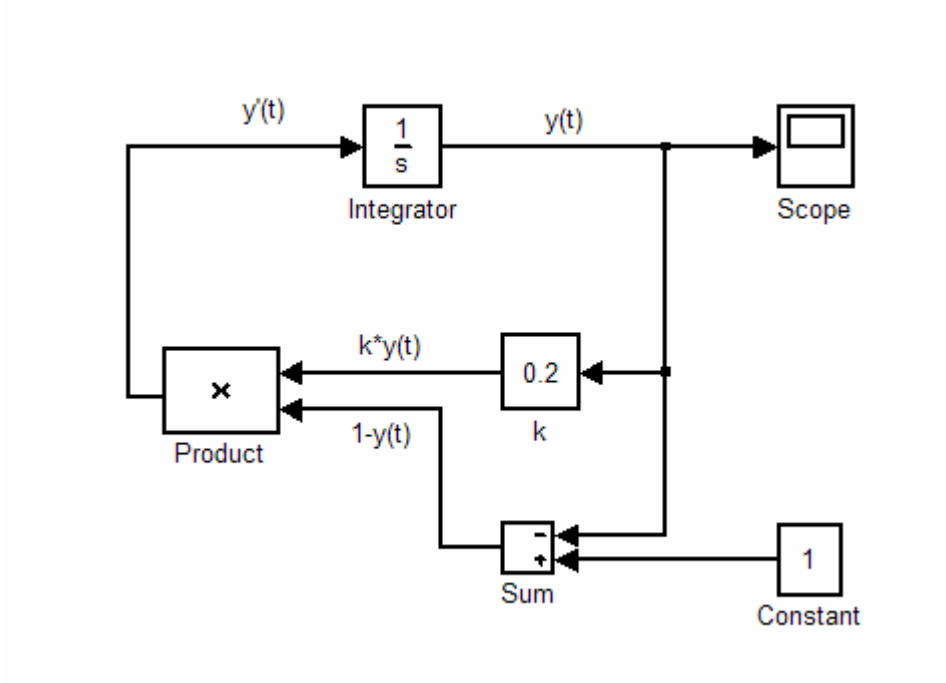


Abbildung 23: Realisierung der obigen Modellgleichung in Simulink

Ergebnis:

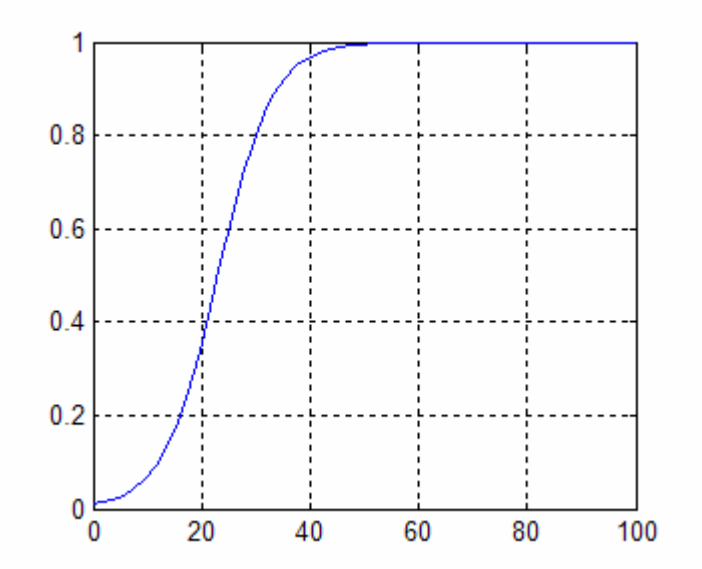


Abbildung 24: Ergebnis der Simulation mit Simulink. Startwert *y0=0.01* und *k=0.2*

Realisierung mit der Toolbox:

```
net=createModel();
ptau=[0 0 0 0];
wP=[0 0 0 0];
wM=[0.2 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0];
[net P]=addKII(net, wP, ptau, 'inity1', 0.01, 'wmatrix', wM);

dt=0.1; tspan=100;
R = runModel(net, dt, tspan, 'solver', 'S3', 'alpha', 0);
plotResult(net, R, 'what', 'y1');
```

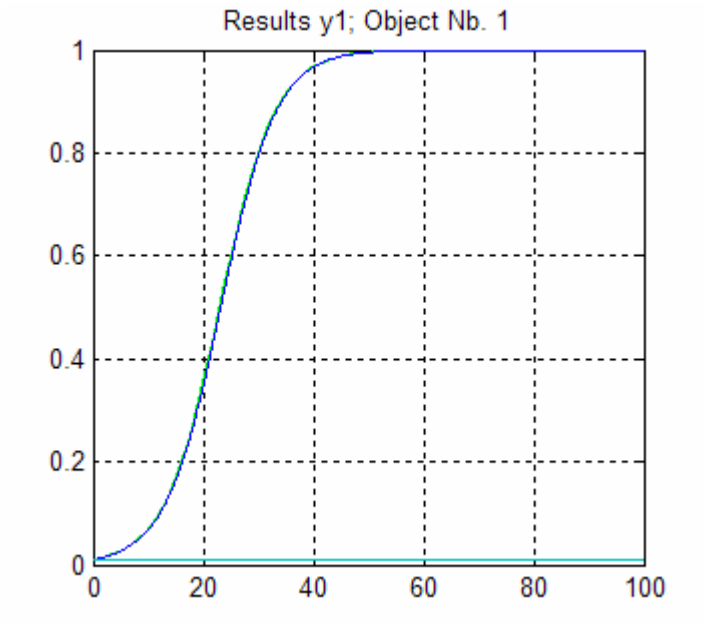


Abbildung 25: Die grüne Linie ist das Ergebnis der Simulation mit der Toolbox, die blaue Linie das Ergebnis mit Simulink von oben.

Wie wir gut sehen können liefern beide Simulationen dieselben Ergebnisse. Die grüne Linie zeigt das Ergebnis der Toolbox, die blaue Linie das Ergebnis mit Simulink. Simulink bittet zuerst den Vorteil einer graphischen Implementierung. Wird das Modell allerdings komplexer verliert man schnell den Überblick und es ist notwendig sich eigene Bibliotheken anzulegen. Eine Loslösung eines Netzes vom Modell ist grundsätzlich nicht vorgesehen und muss wieder über Bibliotheken realisiert werden. Mit der Toolbox können jedoch nur Modelle simuliert werden die auch implementiert sind.

Eine ähnliche Funktionalität bietet das Programm *Dynasys*, das ebenfalls eine graphische Oberfläche zur Modellierung besitzt, wenn auch nicht in dem Ausmaß wie Simulink. Die Vor- und Nachteile sind dieselben wie vorher schon erwähnt.

Eine weitere Alternative ist das Simulieren mit Solvern die von MatLab zur Verfügung gestellt werden. Beispiele sind die Funktionen *ode45.m* und *ode23.m* (ODE – Ordinary Differential Equations), die zur Lösung von Differentialgleichungssystemen 1.Ordnung verwendet werden können, sowie die Funktion *dde23.m,* die speziell die Lösung von Systemen mit Verzögerungen unterstützt. Für komplexere Modelle wird die Implementierung allerdings wieder unüberschaubar, eine getrennte Betrachtung von Netz und Modell, wie es die Toolbox bietet, ist zudem auch nicht möglich.
Die Beispiele in Kapitel 8.1.2 und 8.2.2 werden mit den Solvern *ode45* und *dde23* evaluiert, auf ein weiteres Beispiel an dieser Stelle wird daher verzichtet.

8 Beispiele zu den oben angeführten Modellen

In diesem Kapitel wurden Beispiele ausgewählt die zum einen die einzelnen Modelle besser erklären und verstehen helfen, aber auch zum anderen die Stärken und Schwächen der Toolbox aufzeigen. Der Beispielcode zeigt auch, wie man mit vermeintlichen Problemen umgehen und mit der Toolbox lösen kann. Einige Beispiele werden mit anderen Simulationen evaluiert.

8.1 Beispiele zum K-Model

Im Folgenden wollen wir nochmal das K-Modell aufschlüsseln und dabei zeigen, wie die einzelnen Teile mit der Toolbox zu simulieren sind. Im Weiteren wird nochmals ein Beispiel für den Olfactory Tract behandelt.

8.1.1 Simulation des K0 Objekt

Da die Toolbox immer in Viererknoten, wir nennen dies ein Objekt, organisiert ist und wir jedoch nur das Ergebnis und die Reaktion eines einzelnen Knoten sehen wollen, müssen wir die Verbindung zu den anderen Knoten lösen. Die Verbindung der Knoten untereinander passiert über die Gewichte, und um diese zu lösen müssen die Gewichte auf Null gesetzt werden.

Beispielcode:

```
net=createModel();
ptau=[0.22, 0.72];

wP=[0 0 0 0];
[net P]=addKII(net, wP, ptau);

% Modellparameter
dt=0.1; tspan=500;

% Stimulus kreieren
```

```
S1=ones(1,0.5/dt)*2;                % Dirac Impuls (mit 0.5 ms Länge)
S2=ones(1,20/dt)*1;                 % Rechteck
S3=(sin((0:dt:100)./1)+1)*1;        % Sinus

net = addStimulus(net, P.node{1}, S1, 50);
net = addStimulus(net, P.node{2}, S2, 150);
net = addStimulus(net, P.node{3}, S3, 250);

% Simulation starten
R = runModel(net, dt, tspan);

% Ergebnisse Ploten
plotResult(net, R);
```

Die Knoten werden damit einzeln berechnet und beeinflussen einander nicht.

Ergebnis:

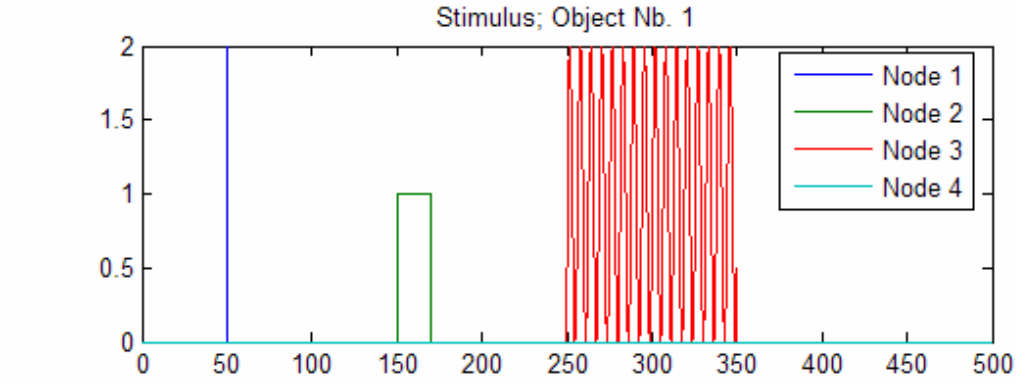


Abbildung 26. Stimuli an den verschieden Knoten. Blau ist der Stimulus am Knoten 1, ein Dirac Impuls mit der Fläche 1; grün der Stimulus am Knoten 2, ein Rechteck mit Amplitude 1 und der Dauer von 20ms. Am Knoten 4, türkise Linie, wurde kein Stimuli angelegt.

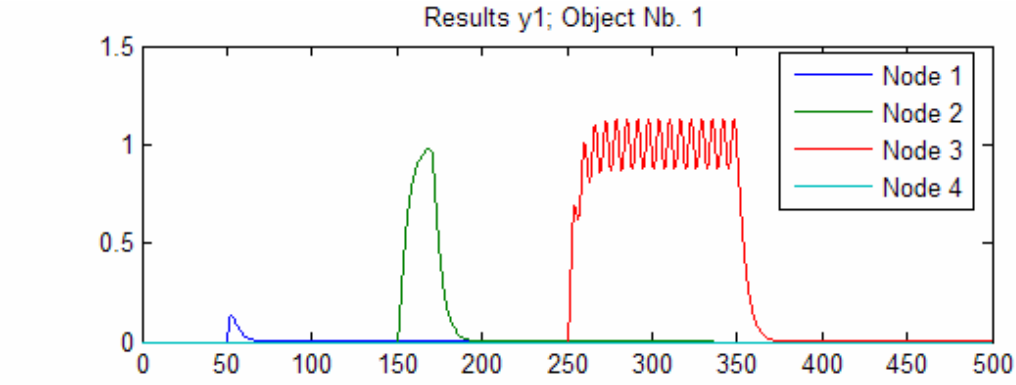


Abbildung 27: Antwort der Knoten auf jeden einzelnen Stimuli.

Beim Aufrufen der Funktion *plotResult.m* wird auch die erste Ableitung des Systems dargestellt, weil dieses eine Differentialgleichung 2. Ordnung ist und die erste Ableitung als Zwischenergebnis vorliegt.

8.1.2 Simulation des KI Objekt

Wir haben, wie oben schon beschrieben, drei Möglichkeiten der Verbindung (auch wenn es im ersten Moment so scheinen mag als wären es vier). Die ersten beiden, gegenseitig erregend und gegenseitig inhibierend, wurden bereits in Kapitel 4 gezeigt. Um zwei Knoten zu verbinden die gegengekoppelt sind, ohne dass diese von den restlichen beiden beeinflusst werden, gibt es zwei Möglichkeiten. Entweder man generiert zwei Objekte und setzt alle Gewichte auf Null und verbindet jeweils einen Knoten eines Objektes mit einem des anderen über laterale Gewichte, oder aber man ändert die Struktur der Gewichtsmatrix des Objektes, die grundsätzlich vorgegeben ist. Wir entscheiden uns für Möglichkeit Zwei, da diese bezüglich Rechenaufwand optimal ist.

Beispielcode:

```
% Struct für Netz
net=createModel();

% notwendige Zeitkonstanten (für Modell S3 nicht erforderlich)
ptau=[0.22, 0.72]; % für Modell S1

wP=[0 0 0 0];             % Gewichte des Objekts P
wei=1.4; wie=-1.9;
wMat=[0 0 0 wie; 0 0 0 0; 0 0 0 0; wei 0 0 0;];
[net P]=addKII(net, wP, ptau, 'wMatrix', wMat);

% Stimulus kreieren
dt=0.1;                           % Schrittweite des Integrators in ms
tspan=100;                        % Berechnete Zeitspanne in ms

S1=ones(1,1/dt);                  % Dirac Impuls (mit 1 ms Länge)
net = addStimulus(net, P.node{1}, S1, 10);

% Simulation starten
R = runModel(net, dt, tspan, 'solver', 'S1');

% Ergebnisse ploten
plotResult(net, R);
```

Oben wurde nun Knoten 1 mit Knoten 4 verbunden, wobei Knoten 1 Knoten 4 mit dem Gewicht 1.5 erregt und Knoten 4 Knoten 1 mit dem Gewicht -1.9 inhibiert. Parameter *wP* ist jetzt irrelevant muss allerdings übergeben werden da er nicht optional ist. Stimuliert wurde am Knoten 1 mit einem Impuls der Länge 1°ms zum Zeitpunkt 10°ms der Simulation.

Ergebnis:

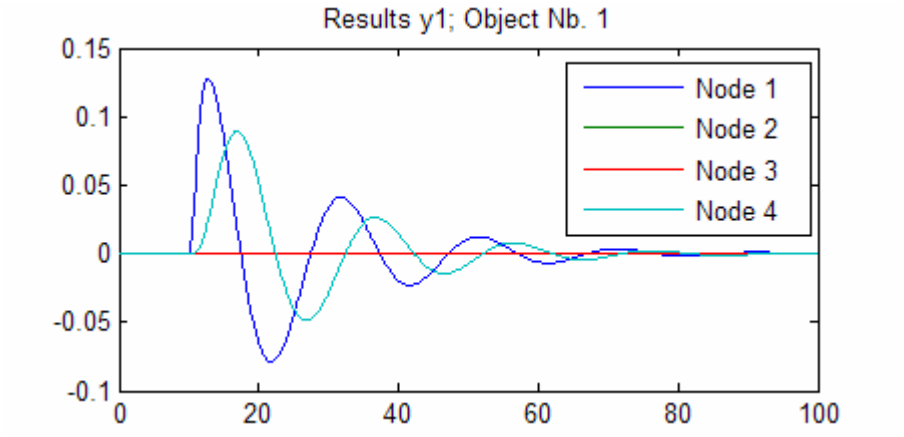


Abbildung 28: Knoten 1 und 4 sind mit den Gewichten 1.4 und -1.9 miteinander verbunden und wuden am Knoten 1 stimuiert. System schwingt sich mit *ptau* ein. Man beachte die 90° Phasenverschiebung der beiden Knoten.

Das System wird mit Stimuli erregt und schwingt sich wieder auf Null ein. Die Frequenz ist hierbei vom Parameter *ptau* im obigen Code abhängig, der mit 0.22 und 0.72 [23] gewählt wurde. Eine Erhöhung der Werte erzielt eine Erhöhung der Frequenz. Weiters ist die Phasenverschiebung zu beachten die sich durch das System ergibt. Es muss noch einmal darauf hingewiesen werden, dass allein von der Wahl der Gewichte abhängt, ob sich das Netz auf Null einschwingt, oder die Schwingung erhalten bleibt.

Validierung:

Dieses Beispiel wird „herkömmlich“ mit dem Solver *ode45.m* von MatLab simuliert und somit das Ergebnis überprüft. *ode45.m* verwendet ebenfalls das Runge-Kutta Verfahren 4. Ordnung [7].

MatLab Code:

```
function []=ExampleKI()
w=[1.4, -1.9];
K0=zeros(4,1);
tspan=[0:0.1:100];
col=['r', 'b', 'g', 'c','m'];
hold on;
options=odeset('maxStep', 0.1);
[t, K]=ode45(@calcK, tspan, K0, options, w);
for i=1:2:3
    plot(t, K(:,i), col(i));
end;
function dKdt=calcK(t,K,w)
    a=0.22; b=0.72;
    dKdt=zeros(4, 1);
    dKdt=[K(2);
          -(a+b)*K(2) - a*b*K(1) + a*b*(w(2)*F(K(3))+S(t));
          K(4);
          -(a+b)*K(4) - a*b*K(3) + a*b*w(1)*F(K(1))];
%-------------------------------------------------------------------------
function [x]=S(t);
    if t>=10 & t<=11
        x=1;
    else
        x=0;
    end
%-------------------------------------------------------------------------;
```

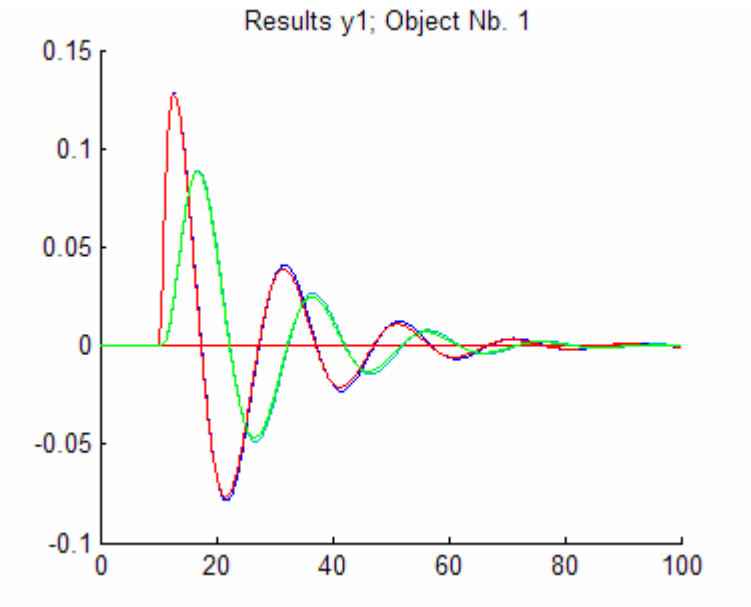


Abbildung 29: Das Ergebnis mit dem Solver *ode45.m*, rote und grüne Linie, wurde über das Ergebnis aus Abbildung 28 geplotet.

Die Ergebnisse aus beiden Simulationen sind identisch. Die Abweichungen ergeben sich aus den gewählten Schrittweiten. Die Funktion *ode45.m* wählt die Schrittweite je nach den Änderungen der Funktion zuvor, die Toolbox hat eine konstante Schrittweite. Möchte man ein genaueres Ergebnis erreichen, muss man die Schrittweite der Toolbox auf zum Beispiel *dt=0.05* verkleinern. Man kann gut sehen, dass bei einem komplexeren Netz die Simulation mit dem Solver *ode45* fast unmöglich wird, da für jeden Knoten die Modellgleichung angegeben werden muss.

8.1.3 Simulation des gesamten Olfactory Tract

Um mit dieser Toolbox auch aufwendigere Netze wie den gesamten Olfactory Tract mit mehreren Objekten pro Layer einigermaßen vernünftig und überschaubar simulieren zu können, werden für diesen Zweck in der Toolbox Funktionen zur Verfügung gestellt die dieses erleichtern sollten. Diese Funktionen *addKIII.m* und *addKIII_2.m* sollen als Anregung aufgefasst werden, größere Netze in Teilnetze zu zerlegen um sie in separaten Funktionen wieder zu verwenden.

Nochmals das Modell des Olfactory Tract:

Parameter:

Gewichte:

```
P: [0.5, 0, 0, 0]
OB: [0.25, 1.5, -1.5, -1.8]
AON: [1.5, 1.5, -1.5, -1.8]
PC: [0.25, 1.4, -1.4, -1.8]
C: [0, 0, 0, 0]
```

Verzögerungen [ms]:

```
L1=5.5; L2=6.5; L3=5.5;
L4=15
```

Laterale Gewichte:

```
MA=1.0; ME=1.5; PM=0.1; BC=;
AI=1.0; EG=0.67; EP=1.0;
CG=1.5;
```

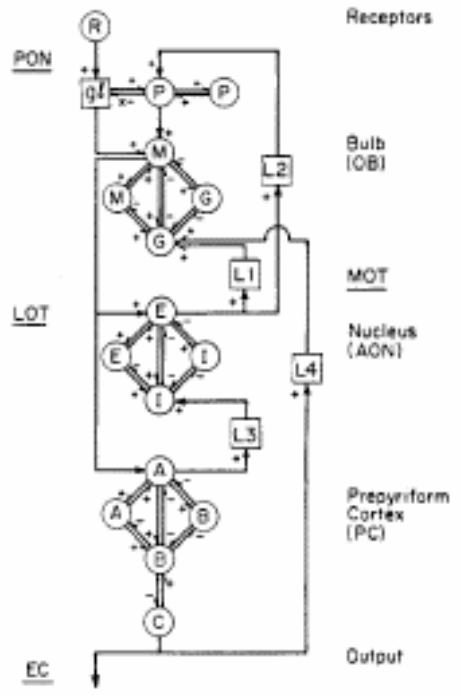


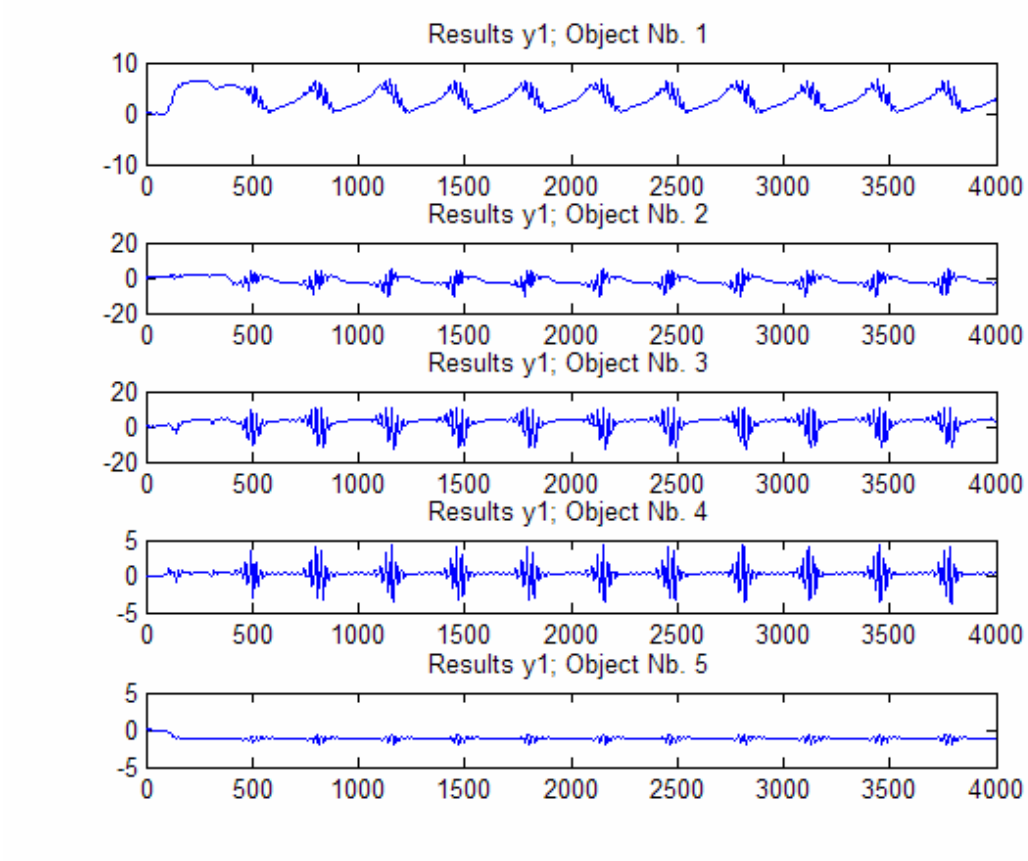
Abbildung 30: Verschaltung des OT mittels den K-Objekten (siehe oben)

Beispielcode:

```
tic;
net=createModel();
ptau=[0.220, 0.720];

wP=[0.5, 0, 0, 0];
wOB=[0.25, 1.5, -1.5, -1.8];
wAON=[1.5, 1.5, -1.5, -1.8];
wPC=[0.25, 1.4, -1.4, -1.8];
wC=[0, 0, 0, 0];

% Netze hinzufügen
[net P]  = addKII(net, wP, ptau);
[net OB] = addKII(net, wOB, ptau);
[net AON]= addKII(net, wAON, ptau);
[net PC] = addKII(net, wPC, ptau);
[net C]  = addKII(net, wPC, ptau);

% Verbindungen erstellen
net = connectKIII(net, 'src',P.node{1},'dest',OB.node{1},'weight
',0.3,'delay',0);

net = connectKIII(net, 'src',OB.node{1},'dest',AON.node{1},'weight
',1.5,'delay',0);
net = connectKIII(net, 'src',OB.node{1},'dest',PC.node{1},'weight
',1.0,'delay',0);

net = connectKIII(net, 'src',AON.node{1},'dest',P.node{1},'weight
',1.0,'delay',6.5); %L2
net = connectKIII(net, 'src',AON.node{1},'dest',OB.node{3},'weight
',1.0,'delay',5.5); %L1

net = connectKIII(net, 'src',PC.node{1},'dest',AON.node{3},'weight
',1.0,'delay',5.5); %L3
net = connectKIII(net, 'src',PC.node{3},'dest',C.node{1},'weight
',1.0,'delay',0);

net = connectKIII(net, 'src',C.node{1},'dest',PC.node{3},'weight
',1.0,'delay',0);
net = connectKIII(net, 'src',C.node{1},'dest',OB.node{3},'weight
',1.0,'delay',15); %L4

% Stimulus kreieren
duration=4000; dt=0.5;
noise=rand(1,duration/dt).*15;
S1=1*ones(1,200/dt);
S2=repmat(sin(0:0.05:2*pi), 1, 30)*5;

net = addStimulus(net, P.node{1}, 1, 0);
```

```
net = addStimulus(net, OB.node{1}, 1, 0);
net = addStimulus(net, P.node{1}, S1, 100);
net = addStimulus(net, OB.node{1}, S1, 100);

% Simulation starten
R = runModel(net, dt, duration,'solver','S1');

plotResult(net, R);
toc;
```

Ergebnis:



Das Ergebnis wurde als Summe über die vier Knoten eines Objektes geplotet. Die Resultate wie sie in [11] präsentiert wurden, konnten mit diesen Parametern nicht nachvollzogen werden.

8.2 Beispiele zum Firing-Rate Modell

8.2.1 Beispiel Kapitel 5

Um das Beispiel aus Kapitel 5 nachzusimulieren, müssen ein paar Besonderheiten berücksichtigt werden. Die erste ist, dass eine Hintergrundaktivität γ (Schwelle) angenommen wird. Diese ist in unserem Modell als solche nicht enthalten und muss daher durch einen Trick eingebracht werden. Das geschieht, indem die Hintergrundaktivität als Stimulus interpretiert wird, der über die gesamte Laufzeit geht. Als nächstes besitzt die exzitatorische Population (Knoten 1) eine Selbsterregung. Zur Simulation werden wieder die Knoten 1 und 4 herangezogen. Da dieses Modell für jede Population ein eigenes τ besitzt, müssen auch jene angegeben werden, deren Knoten nicht in Verwendung sind. Dabei muss dieser Wert ungleich Null sein, da ansonst eine Division durch Null entsteht.
Weiters verlangt dieses Beispiel, um es genau so wie in der Literatur vorgestellt zeigen zu können, dass der Startwert bei 30 und 50 Hz liegt. Diese können wir beim Erstellen des Objekts mit dem Parameter *'inity1'* für jeden Knoten übergeben.

Parameter:

M_{EE}=1.25; M_{II}=0; M_{EI}=-1; M_{IE}=1; τ_E=10ms; τ_I=30ms; γ_E=-10 γ_I=10 Hz.

Beispielcode:

```
net=createModel();
ptau=[10 1 1 30];
wP=[0 0 0 0];             % Gewichte des Objekts P
wei=1; wie=-1; wee=1.25;
wMat=[wee 0 0 wie; 0 0 0 0; 0 0 0 0; wei 0 0 0;];
y0=[30, 0, 0, 50];
[net P]=addKII(net, wP, ptau, 'wMatrix', wMat, 'inity1', y0);

% Stimulus kreieren
dt=0.1;                           % Schrittweite des Integrators in ms
tspan=1000;                       % Berechnete Zeitspanne in ms

S2=ones(1,tspan/dt)*10;           % Rechteck
```

```
net = addStimulus(net, P.node{1}, S2, 0);
net = addStimulus(net, P.node{4}, -S2, 0);

% Simulation starten
R = runModel(net, dt, tspan, 'solver', 'S2', 'sigm', @sigm0);

% Ergebnisse ploten
plotResult(net, R);
```

Das Phasendiagramm erhält man, indem man die Ergebnisse der beiden Knoten gegeneinander plotet:

```
plot(R.KII(1).y1(1,1:end),R.KII(1).y1(4,1:end));
```

Ergebnis:

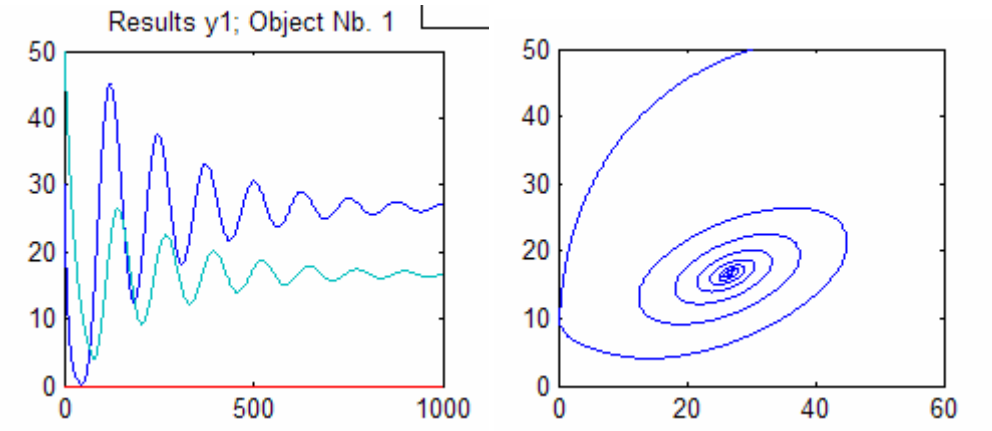


Abbildung 31: Links: Einschwingvorgang der beiden Popualtionen. Rechts: zugehöriges Phasendiagramm. Fixpunkte bei 18 und 28.

Im Phasendiagramm sieht man nun gut, dass das System im Punkt 18 und 28 stabil ist.

8.2.2 Populationen mit Verzögerung

In diesem Beispiel wollen wir zwei sich erregende Populationen simulieren, wobei die Signale eine Verzögerung von 10°ms und 20°ms besitzen. Dieses Beispiel wird anschließend mit dem MatLab-Solver *dde23.m* simuliert und somit das Ergebnis der Toolbox evaluiert. Die Funktion *dde23.m* unterstützt Modelle mit Verzögerungen.

Beispielcode:

```
net=createModel();
ptau=[10 10 1 1];
wP=[0 0 0 0];              % Gewichte des Objekts P
[net P]=addKII(net, wP, ptau);
% lateralen Verbindungen bestimmen
net = connectKIII(net,'src',P.node{1},'dest',P.node{2},'weight',1,'delay', 10);
net = connectKIII(net,'src',P.node{2},'dest',P.node{1},'weight',1,'delay', 20);
% Stimulus kreieren
dt=0.1;                              % Schrittweite des Integrators
tspan=250;                           % Berechnete Zeitspanne
S2=ones(1,1/dt);                     % Rechteck
net = addStimulus(net, P.node{1}, S2, 10);
% Simulation starten
R = runModel(net, dt, tspan, 'solver', 'S2', 'sigm', @sigm0);
% Ergebnisse ploten
plotResult(net, R, 'what', 'y1');
```

Ergebnis:

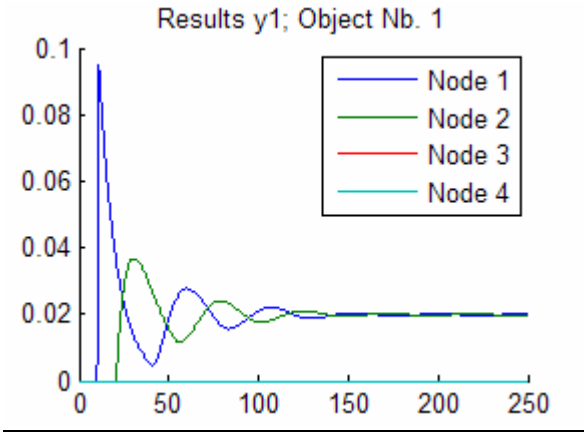


Abbildung 32: Die Ausbreitung der Erregung zweier verzögerter Populationen. Knoten 1 wurde zum Zeitpunkt 10°ms mit einem Dirac Impuls stimuliert. Knoten 2 wird 10°ms später von Knoten 1 erregt und Knoten 1 wiederum 20°ms später von Knoten 2.

Validierung:

Nochmal dasselbe Beispiel mit dem MatLab-Solver *dde23.m*. Die Funktion *dde23.m* verwendet ein Runge-Kutta Verfahren 2. Ordnung [7].

MatLab Code:

```
options=ddeset('maxStep', 0.1);
tspan=[0:0.1:250];
sol = dde23(@ddex1de,[10, 20],zeros(2,1), tspan, options);
hold on;
plot(sol.x,sol.y(1,:),'r');
plot(sol.x,sol.y(2,:),'g');
% --------------------------------------------------------------------------
function dydt = ddex1de(t,y,Z)
ylag1 = Z(:,1);
ylag2 = Z(:,2);
tau=[10, 10];
w=[1, 1];
dydt = [ (-y(1)+ylag2(2)*w(1)+S(t))./tau(1);
         (-y(2)+ylag1(1)*w(2))./tau(2)];
% --------------------------------------------------------------------------
    function x=S(t)
        if t>=10 & t<=11
            x=1;
        else
            x=0;
        end;
% --------------------------------------------------------------------------
```

Ergebnis:

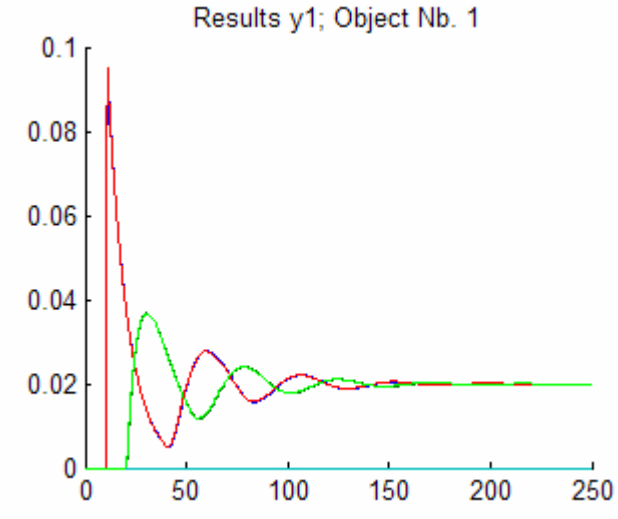


Abbildung 33: Berechnung mit dem Solver *dde23.m*. Das Ergbnis, rote und grüne Linie, wurden wieder über das Ergebnis der Toolbox geplotet.

Die Funktion *dde23.m* liefert dasselbe Resultat wie die Toolbox. Die Linien sind praktisch deckungsgleich. Größere Modelle werden jedoch wieder unübersichtlich, da für jede Population die Modellgleichung angegeben werden muss.

8.2.3 Simulation des Olfactory Bulb

Da in dieser Arbeit der Olfactory Bulb als Hauptregion zur Betrachtung des Verhaltens von Neuronenpopulationen genommen wurde, wollen wir noch ein Beispiel dazu zeigen. Wir verwenden das Modell von Li und Hopfield [26] aus Kaptel 5, dies soll das Verhalten der Mitral- und Granulzellen zueinander bei Stimulierung simulieren.

Parameter:

M_{EE}=0; M_{II}=0; M_{EI}=-1.5; M_{IE}=0.9; τ_E=6,7; τ_I=6,7ms; latGew von Mital zu Granul = 1; von Granul zu Mitral = -1.5

Netz:

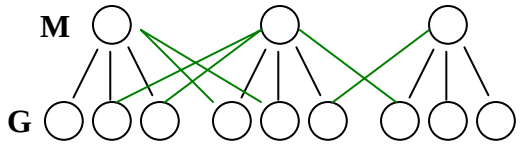


Abbildung 34: M sind Mitralzellen G sind Granulzellen. Grüne Linien sind laterale Verbindungen, alle Verbindungen sind „bidirektional“.

Das Ergebnis zeigt, dass Mitralzellen und Granulzellen bei Stimulation mit verschiedenen Gerüchen mit verschiedenen Amplituden aufschwingen. Es ist dabei zu beachten, dass die Schwingung nur Zustande kommt wenn eine bestimmte Schwelle überschritten wird, und zwar wird dieser von Realteil und Imaginärteil der Stabilitätsmatrix bestimmt. [26].

Ergebnis:

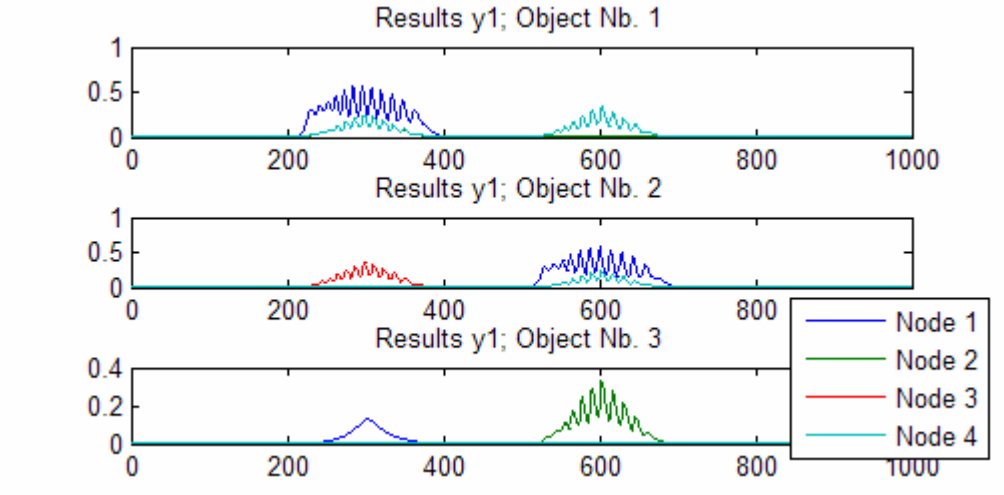


Abbildung 35: Mitralzellen (Knoten 1) wurden zum Zeitpunkt 200°ms mit einem Dreiecksimpuls mit verschiedenen Amplituden stimuliert und zum Zeitpunkt 500°ms mit einem zweiten Stimulus die jeweils zwei verschiedene Gerüche darstellen.

Als Schwäche stellt sich heraus, dass für verschiedene Populationen nicht verschiedene Aktivierungsfunktionen in der Simulation angegeben werden können. So besitzen z.B. Granulzellen eine wesentlich kleinere maximale Frequenz als Mitralzellen. Dies verzerrt die Simulation und kann nur zum Teil über die Veränderung der Gewichte wettgemacht werden.

8.3 Beispiele zum Populationenmodell

8.3.1 Ausbreitung von Infektionen

Dieses Beispiel ist jetzt ausnahmsweise nicht unserem Problem, der Simulation von Neuronenpopulationen, entrissen, sondern stammt aus dem Fachgebiet der Epidemiologie, und soll zeigen, dass auch andere Probleme mit der Toolbox simulierbar sind.

Die Ausbreitung von Infektionen kommt in verschiedenen Formen, wie z.B. mit/ohne Immunisierung und mit/ohne Latenzzeit, vor und wird in der Literatur mit den verschiedensten Modellen beschrieben. Keine Immunisierung bedeutet, dass Infektiöse nach ihrer Genesung wieder infizierbar sind. Damit bleibt die Größe der

Population konstant. Keine Latenzzeit bedeutet, dass Individuen nach ihrer Infizierung sofort Infektiös sind. Das würde in unserer Simulation einer Verzögerung gleichkommen.
In unserem Beispiel hier haben wir vier homogene Teilpopulationen ohne Immunisierung und ohne Latenzzeit und die Gewichte stellen die Kontaktrate von Infektiösen mit Suszeptiblen pro Zeiteinheit dar. Der Parameter *'alpha'* stellt die Genesungsrate in einer Population dar.

Was uns nun interessiert, sind die Stabilitätspunkte bei verschiedener Anzahl von Infektiösen zum Zeitpunkt Null. Dazu müssen wir die Simulation für jeden Startwert extra laufen lassen und das Ergebnis im selben Fenster ploten.

Beispielcode:

```
% Struct für Netz
net=createModel();
ptau=[0 0 0 0];
wP=[0.3 0.05 0.1 0.1];            % Gewichte des Objekts P
[net P]=addKII(net, wP, ptau);

% Stimulus kreieren
dt=0.1;                            % Schrittweite des Integrators
tspan=100;                         % Berechnete Zeitspanne

% Simulation starten
for i=1:20
    y0=ones(1,4)*i/20;
    net.KII(P.idx).inity1=[y0];
    R = runModel(net, dt, tspan, 'solver', 'S3', 'alpha', 0.7);
    plotResult(net, R, 'what', 'y1');
end
```

Wir verwenden hier 20 Startwerte von 0.1 bis 1 im Abstand von jeweils 0.05 für alle Teilpopulationen und alle Teilpopulationen starten mit demselben Wert. Um dies zu realisieren bedienen wir uns des Tricks, dass wir den Startwert nicht bereits beim Kreieren des Objekts *P* übergeben, sondern erst vor dem Start der Simulation dem Netz direkt. Die *plotResult.m* Funktion hat die Eigenschaft, dass sie beim Verwenden des Parameters *'what'* bei mehrmaligem Aufruf immer in dasselbe Fenster plotet.

Weiters wurde *'alpha'* für alle Teilpopulationen gleich groß gewählt, was nicht unbedingt notwendig ist.

Ergebnis:

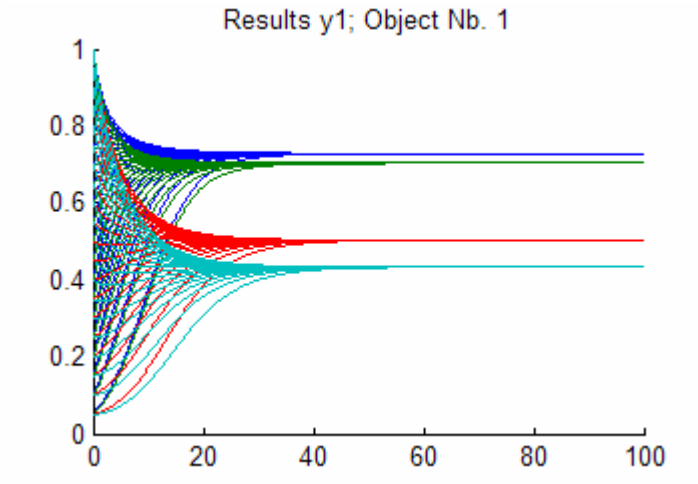


Abbildung 36: Stabilitätspunkte der vier homogenen Teilpopulationen. Man sieht hier, dass die Fixpunkte nicht von der Anzahl der Infektiösen zum Zeitpunkt Null abhängen.

Wie wir gut sehen können ist unabhängig vom Startwert, ausgenommen null Infektiöse, der Fixpunkt immer derselbe in jeder Teilpopulation. Daraus dürfen wir schließen, dass dieser alleinig von der Kontaktrate (Gewichtsmatrix) und der Genesungsrate *'alpha'* abhängt.
Die Startwerte sollten sinnvollerweise zwischen 0 und 1 liegen, da die Größe der Teilpopulationen im Modell auf 1 normiert ist, auch wenn es das Ergebnis in diesem Fall nicht ändern würde.

8.3.2 Homogene Populationen mit negativen Gewichten

Wir gehen nun wieder zurück und wollen die einzelnen Teilpopulationen wieder zum einen als inhibierende Neuronen, und zum anderen als erregende Neuronen sehen. Dadurch erhalten wir in unserem Modell wieder negative und positive Gewichte, die wiederum die Stärke der synaptischen Verbindung der einzelnen Neuronenverbände darstellen.

Dieses Beispiel wurde ausgewählt um zu zeigen, wie und wo man in der Simulation eingreifen muss, um die Gewichte des Systems innerhalb einer bestimmten Zeit zu verändern und wie man das gesamte Ergebnis darstellen kann. Außerdem wird das Verhalten der Fixpunkte genauer untersucht.

Durch geeignete Wahl der Parameter erreichen wir nun, dass abhängig vom Startwert das System in zwei verschiedene Fixpunkte für jeden Knoten fällt. Das Programm ist ansonsten vom vorherigen Beispiel unverändert.

Parameter:

w_{EE}=0.1; w_{IE}=-0.9; w_{EI}=0.1; w_{II}=-0.9; alpha=0.45; dt=0.1;

Ergebnis:

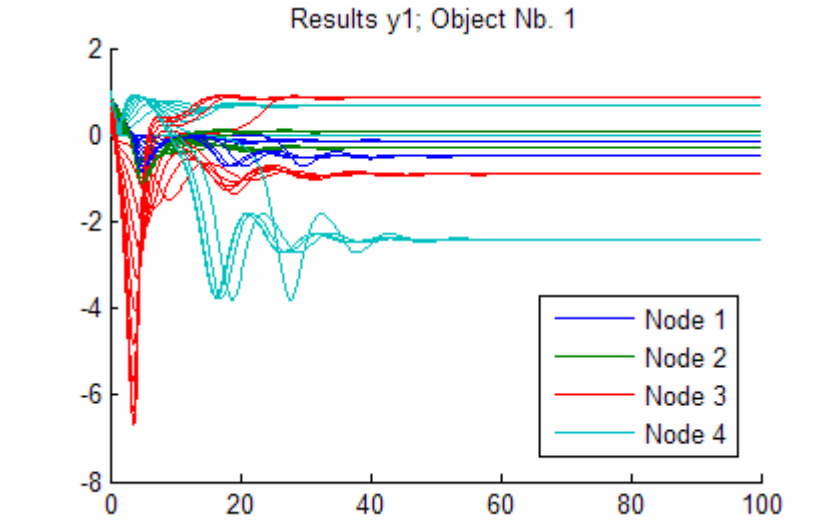


Abbildung 37: Simulation mit negativen Gewichten. Die Knoten zeigen abhängig vom Startwert jeweils einen verschiedenen Fixpunkt.

Wir wollen nun dieses Phänomen noch weiter durchleuchten in dem wir während der Simulation die synaptischen Verbindungen lösen, das heißt die Gewichte auf Null setzen. Dadurch erhalten die Populationen keine gegenseitige Erregung mehr und ihre eigene Erregung geht Richtung Null.
Um jedoch die Gewichte während der Simulation beeinflussen zu können, müssen wir diese in Teilschritten durchführen. Dazu simulieren wir zuerst mit den obigen

Parametern bis Zeit 40°ms, setzen dann die Gewichte auf Null, simulieren wieder 20°ms, die Erregung der Populationen fällt dabei gegen Null und koppeln dann die Populationen wieder mit den ursprünglichen Gewichten zusammen und simulieren weitere 60°ms.

Beispielcode:

```
% Struct für Netz
net=createModel();
ptau=[0 0 0 0];
wP=[0.1 -0.9 0.1 -0.9];             % Gewichte des Objekts P
[net P]=addKII(net, wP, ptau);

% Stimulus kreieren
dt=0.1;                             % Schrittweite des Integrators in ms
tspan=40;                           % Berechnete Zeitspanne in ms

% Simulation starten
res=struct();               % Variable zum sammeln der Zwischenergebnisse
oldW=net.KII(P.idx).weight; % alte Gewichte merken
c=10;
for i=1:c
    y0=ones(1,4)*i/c;
    res(i).y1=zeros(4,1);
    net.KII(P.idx).inity1=[y0];
    tspan=40;                           % Berechnete Zeitspanne in ms
    R = runModel(net, dt, tspan, 'solver', 'S3', 'alpha', 0.45);
    res(i).y1=[res(i).y1, R.KII(P.idx).y1];
    net.KII(P.idx).weight(:,:)=0;       % Gewichte auf null setzen
    tspan=20;                           % Berechnete Zeitspanne in ms
    R = runModel(net, dt, tspan, 'solver', 'S3', 'alpha', 0.45, 'hist', R);
    res(i).y1=[res(i).y1, R.KII(P.idx).y1];
    net.KII(P.idx).weight(:,:)=oldW;
    tspan=60;                           % Berechnete Zeitspanne in ms
    R = runModel(net, dt, tspan, 'solver', 'S3', 'alpha', 0.45, 'hist', R);
    res(i).y1=[res(i).y1, R.KII(P.idx).y1];
end
t=[0:length(res(1).y1)-1].*dt;
figure; hold on;
for i=1:c;
    plot(t, res(i).y1);
end;
```

Wie wir sehen können haben wir nun drei getrennte Simulationen, die Gewichte ändern wir im Netz direkt. Um das Ergebnis in einem Bild darstellen zu können

müssen wir die Teilergebnisse in einer externen Variablen sammeln und diese dann separat ploten. Hier gibt es Verbesserungspotential.

Ergebnis:

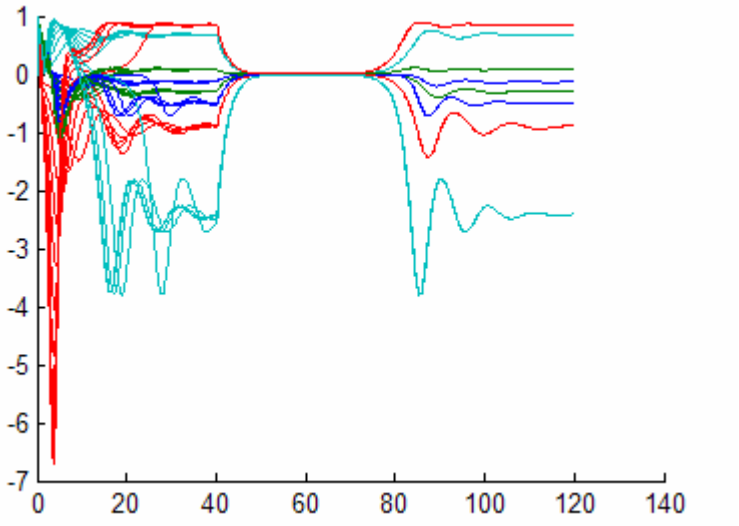


Abbildung 38: Bis zum Zeitpunkt (ZP) 40°ms Simulation mit den Gewichten [0.1, -0.9, 0.1, -0.9], von 40°ms bis ZP 60°ms werden alle Gewichte auf Null gesetzt, die Erregung fällt daher exponentiell gegen Null, von ZP 60°ms bis 120°ms werden vorige Gewichte wieder eingekoppelt und alle Populationen erreichen wieder den ihrigen Fixpunkt.

Nun dürften die verschiedenen Fixpunkte nicht direkt vom Startwert sondern von der daraus resultierenden Phasenlage der Populationen zueinander entstehen wie uns das Ergebnis der Simulation zeigt. Dauert diese Relaxphase allerdings zulange, fallen alle Werte wirklich auf Null und verbleiben dort.
Diese Art und Weise der Gewichtsänderung während der Simulation ist auch dafür gedacht diverse Lernalgorithmen, wie bereits im K-Modell vorgestellt, zu implementieren.

Am Schluss muss noch angemerkt werden, dass bei diesem Beispiel das Gesetz von Dales verletzt wurde welches besagt, dass exzitatorische Neuronen nur erregende, und inhibitorische Neuronen nur inhibierende, postsynaptische Verbindungen besitzen.

9 Aussichten und weiterführende Arbeiten

9.1 Die Arbeiten mit homogenen Teilpopulationen

Die Toolbox besitzt zur Zeit alle nötigen Funktionen um inhomogene Populationen, wie sie im Nervensystem vorkommen, zu simulieren. Eines der wichtigsten Vorhaben in nächster Zeit wird sein, die bekannten Modelle dahin weiterzuentwickeln, um die bekannten Mechanismen, wie sie während des Schlafes passieren, modellieren zu können. Das große Interesse darin entsteht, weil vermutet wird, dass während des Schlafes eine Umspeicherung des gewonnen Wissens in den Kortex passiert. Kann man nun diesen Vorgang mit einem Modell simulieren, das sich an physiologische Vorgänge hält, so kann man sich auch Erkenntnisse über den Vorgang dessen erhoffen, was beim Schlaf passiert.

9.2 Die Toolbox betreffend

Ein wesentlicher Faktor bei solchen Modellen ist die benötigte Zeit für einen Simulationsdurchlauf. Um diese zu verkürzen wären kompilierte Funktionen, zumindest jene die den Kern betreffen, von Vorteil und würden die Rechenzeit erheblich verkürzen. Darum wird in der nächsten Zeit angestrebt, die vorgestellten Modelle samt Integrator herauszulösen und als C-Module zu implementieren, die in kompilierter Form eingebunden werden können.

Weiters werden verbesserte Plotfunktionen angestrebt, besonders solche, mit denen es möglich ist, das erstellte Netzwerk mit Verbindungen und Verzögerungen graphisch darzustellen. Bei komplexeren Netzen verliert man schnell den Überblick im Code, eine graphische Darstellung zur Kontrolle wäre hier sehr hilfreich.

Zur Diskussion stehen auch Funktionen, mit denen es möglich ist, die enthaltene Gewichtsmatrix eines Modells genauer auf Stabilitätsverhalten zu prüfen. Dazu sind

allerdings in den meisten Fällen die mathematischen Vorraussetzungen, bis auf ein paar Ausnahmen, noch nicht geschaffen.

Angeregt wurde auch bereits die Möglichkeit, die Modelle auf ihre Bifurcationen hin zu prüfen. Die Schwierigkeit liegt hier darin, dass die Systeme bei Verwendung von nur wenigen Populationen schnell eine sehr hohe Komplexität erreichen. Da zur Analyse der Bifurcationen immer der stabile eingeschwungene Zustand bei einer bestimmten Gewichtsmatrix dargestellt wird, ergeben sich bereits bei nur zwei Gewichten theoretisch unendlich viele Diagramme, da ein Gewicht immer fixiert werden muss und das zweite variiert wird.

Anhang:

A.1 Allgemeine Aussagen zu Stabilität und Gleichgewicht

Hier werden mathematische Hilfsmittel, die zur qualitativen Analyse der Modelle hilfreich sind in einigen wenigen Sätzen erklärt bzw. definiert.

A.1.1 Irreduzible Matrizen

Definition 1: Eine $n \times n$-Matrix $A = (a_{ij})$ heiße irreduzibel, falls es nicht möglich ist, die Indexmenge $G = \{1,...,n\}$ derart in zwei disjunkte nichtleere Teilmengen $G1, G2$ zu unterteilen $(G1, G2 \neq 0; G1 \cap G2 = 0; G1 \cup G2 = G)$, dass $a_{ij} = 0$ *für alle* $i \epsilon G1$; $j \epsilon G2$.

Definition 2: Eine Matrix $A = (a_{ij})$ heiße nichtnegativ, falls $a_{ij} \geq 0$ *für alle i,j*.
A heiße positiv falls $a_{ij} > 0$ *für alle i, j*.

Bezeichnung: Ist $A = (a_{ij})$ eine $n \times n$-Matrix, so bezeichne $a_{ij}^{(k)}$ das a_{ij}-Element der Matrix A^k.

Lemma 3: Eine nichtnegative $n \times n$-Matrix A ist irreduzibel genau dann, wenn es für jedes Paar *(i,j)* ihrer Indexmenge eine positive ganze Zahl $m = m(i,j)$ gibt, sodass $a_{ij}^{(m)} > 0$

Definition 4: $\lambda_1,...,\lambda_n$ seien die Eigenwerte einer $n \times n$-Matrix A. Man definiert den Stabilitätskoeffizient (stability modulus) als $\max_{1<i<n} \mathrm{Re}\,\lambda_i$. Der Spektralradius ist definiert als $\max_{1<i<n} |\lambda_i|$.

Die Aussagen des folgenden Satzes haben ihren Ursprung in von Perron & Frobenius verfassten Arbeiten (um 1900).

Satz 5: Sei $A = (a_{ij})$ eine irreduzible, nichtnegative $n \times n$-Matrix. Dann besitzt A einen Eigenwert s mit folgenden Eigenschaften:

(a) s ist reell und positiv.

(b) s ist gleichzeitig auch der Stabilitätskoeffizient und der Spektralradius von A.

(c) s ist einfache Lösung der charakteristischen Gleichung von A.

(d) Wenn irgendein Element von A vergrößert wird, vergrößert sich auch s.

(e) Zu s gibt es rechte und linke Eigenvektoren von A, die positiv in allen Koordinaten sind.

(f) Es gibt nur jeweils einen linear unabhängigen Rechts- bzw. Linkseigenvektor zu s.

(g)

$$\min_i \sum_j a_{ij} \le s \le \max_i \sum_j a_{ij}$$

wobei aus Gleichheit in einer Ungleichung auch Gleichheit in der anderen Ungleichung folgt.

$$\min_j \sum_i a_{ij} \le s \le \max_j \sum_i a_{ij}$$

wobei aus Gleichheit in einer Ungleichung auch Gleichheit in der anderen Ungleichung von folgt.

Der Eigenwert s wird Perron-Frobenius-Eigenwert, die ihm entsprechenden positiven Eigenvektoren Perron-Frobenius-Eigenvektoren genannt.

Satz 6: Sei $A = (aij)$ eine irreduzible $n £ n$-Matrix und $aij ¸ 0$ $8i 6= j$. Dann besitzt A einen Eigenwert s mit folgenden Eigenschaften:

(a) s ist reell.

(b) s ist gleichzeitig auch der Stabilitätskoeffizient.

Die Eigenschaften (c)-(g) aus Satz 5 treffen unverändert auch auf dieses s zu.

A.1.2 Stabilität von Systemen

Auch für ein allgemeines Differentialgleichungssystem kann man eine Aussage über die Stabilität treffen, diese ist jedoch deutlich ungenauer:

$\frac{dy}{dt} = f(y), \quad y \in \Re^n, \quad f : D \to \Re^m, \quad D \subseteq \Re^n$, D sei ein Gebiet

f(y) sei zwei mal stetig partiell differenzierbar;

B bezeichne eine zusammenhängende Teilmenge von D;

$y_0 \in D$ sei ein Gleichgewichtspunkt des Differentialgleichungssystems (DGLS);

A sei wiederum die Jacobi-Matrix der rechten Seite des DGLS für $y=y_0$.

Dann gilt:

1) $s(A) \leq 0 \Rightarrow$ Lösung y_0 ist *lokal* asymptotisch stabil.
2) $s(A) > 0 \Rightarrow$ Lösung y_0 ist instabil.

A.2 Das Euler-Cauchy Verfahren

Hier teilt man das Integrationsintervall in **äquidistante** Teilintervalle mit den Stützstellen

$$t_n = t_0 + n \cdot \Delta t \qquad n = 0, 1, 2, \ldots \qquad (1)$$

und einer konstanten Schrittweite (Zeitschritt) Δt. Weiters verwendet man zur Approximation der ersten Ableitung von y eine Vorwärtsdifferenz erster Ordnung. Damit wird aus der gegebenen Differentialgleichung eine **Differenzengleichung**:

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} = f(t, y(t)) \qquad (2)$$

Diese läßt sich nach $y(t + \Delta t)$ auflösen,

$$y(t+\Delta t) = y(t) + \Delta t \cdot f(t, y(t)) \qquad (3)$$

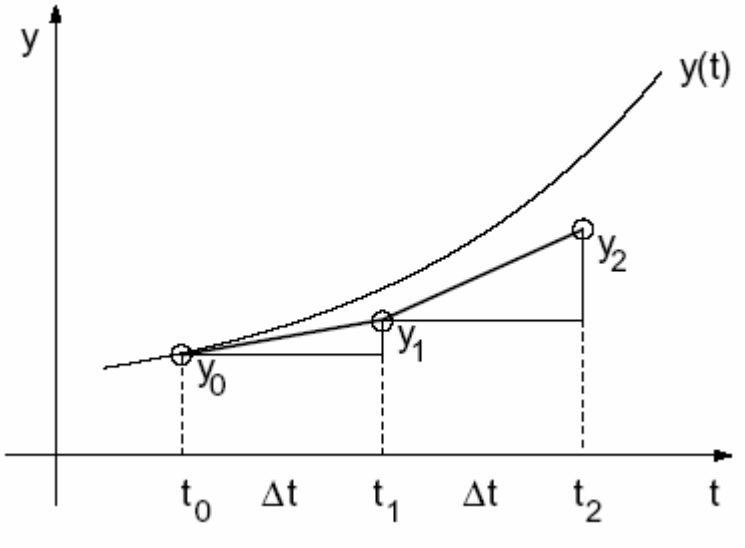
und man erhält schließlich das **Euler-Verfahren**:

$$\begin{aligned} y_{n+1} &= y_n + \Delta t \cdot f(t_n, y_n) \\ t_{n+1} &= t_n + \Delta t \end{aligned} \qquad (4)$$

Beginnend mit einem Startwert $y_0 = y(t_0)$ (aus der Anfangsbedingung) liefert dieses Verfahren für $n = 0,1,2,\ldots$ eine Folge von Näherungslösungen $y_n \approx y(t_n)$ des Anfangswertproblems. Das Euler-Verfahren ist ein **explizites Einschrittverfahren**,

da (a) y_{n+1} nur auf der linken Seite des numerischen Schemas (4) auftritt und (b) zur Berechnung von y_{n+1} nur der Näherungswert y_n aus dem unmittelbar vorhergehenden Zeitschritt verwendet wird. Man kann zeigen, dass der Fehler, den die Näherungslösung y_n auf Grund der Approximation (3) nach einer bestimmten Zahl von Zeitschritten gegenüber der exakten Lösung $y(t_n)$ aufweist (kumulativer *Diskretisierungsfehler* bzw. kumulativer *Abbruchfehler*), von der Größenordnung $O(\Delta t)$ ist. Man sagt, das Euler-Verfahren ist ein Verfahren **erster Ordnung**.

Geometrisch bedeutet das Euler-Verfahren, dass zur Berechnung eines neuen Näherungswertes y_{n+1} die Änderung y_{n+1}-y_n aus der Steigung $k:=f(t_n;y_n)$ der Tangente an y im Punkt $(t_n;y_n)$ abgeschätzt wird. Das Euler-Verfahren nennt man daher auch **Polygonzugverfahren**.



A.3 Das Runge-Kutta-Verfahren 4.Ordnung

Das Runge-Kutta-Verfahren erfordert *4 Funktionsauswertungen* pro Zeitschritt. Wegen der höheren Fehlerordnung kann aber bei gleicher Genauigkeit die Schrittweite Δt um ein Vielfaches größer gewählt werden als beim Euler-Verfahren [6] bzw. bei gleicher Schrittweite eine höhere Genauigkeit erzielt werden.

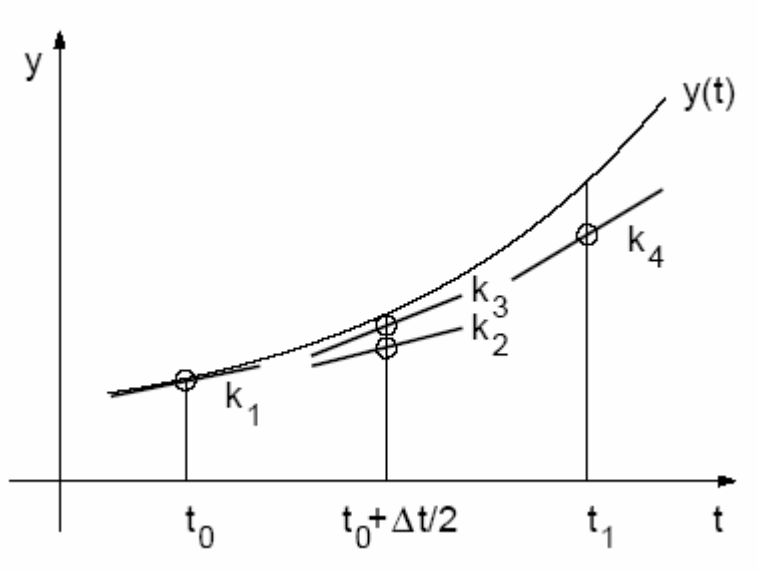
$$k_1 := f(t_n, y_n)$$

$$k_2 := f(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} k_1)$$

$$k_3 := f(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} k_2)$$

$$k_4 := f(t_n + \Delta t, y_n + \Delta t k_3)$$

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$



Zur Berechnung des neuen Näherungswertes y_{n+1} werden vier Steigungen verwendet, eine am Intervallanfang (k_1), zwei in der Intervallmitte (k_2 und k_3) und eine am Intervallende (k_4). Dadurch werden Näherungen für die Taylorreihe erzeugt [7].

Literaturverzeichnis:

1. Sejnowski, T.J. and A. Destexhe, *Why do we sleep?* Brain Res, 2000. **886**(1-2): p. 208-223.
2. Bishop, W.M.C.M., *Pulsed Neural Networks*. 1998.
3. Dayan, P. and L. Abbott, *Network Models*, in *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. 2001, MIT Press.
4. Koppl, C., *Phase locking to high frequencies in the auditory nerve and cochlear nucleus magnocellularis of the barn owl, Tyto alba.* J Neurosci, 1997. **17**(9): p. 3312-21.
5. Bossel, H., *Modellbildung und Simulation*. 1992, Wiesbaden.
6. Ludyk, G., *Simulation dynamischer Systeme*, in *CAE von dynamischen Systemen*. 1990, Springer: Bremen.
7. Dormand, J.R. and P.J. Prince, *A family of embedded Runge-Kutta formulae.* J. Comp. Appl. Math., 1980. **6**: p. 19-26.
8. Heermann, P., *Physik im Phasenraum: Dynamische Systeme und Chaos*. 2003: Univ. Heidelberg.
9. Linster, C. and M. Hasselmo, *Modulation of inhibition in a model of olfactory bulb reduces overlap in the neural representation of olfactory stimuli.* Behav Brain Res, 1997. **84**(1-2): p. 117-27.
10. Davison, A.P., J. Feng, and D. Brown, *Dendrodendritic Inhibition and Simulated Odor Responses in a Detailed Olfactory Bulb Network Model.* J Neurophysiol, 2003. **90**(3): p. 1921-1935.
11. Freeman, W.J., *Petit mal seizure spikes in olfactory bulb and cortex caused by runaway inhibition after exhaustion of excitation.* Brain Res, 1986. **396**(3): p. 259-84.
12. Michael T. Shipley, M.E., *Functional organization of olfactory system.* Journal of Neurobiology, 1996. **30**(1): p. 123-176.
13. Skarda, C.A. and W.J. Freeman, *How brains make chaos in order to make sense of the world. Behavioral and Brain Sciences.* 1987(10): p. 161-173.
14. Kerr, J.N. and D. Plenz, *Action potential timing determines dendritic calcium during striatal up-states.* J Neurosci, 2004. **24**(4): p. 877-85.
15. Kerr, J.N. and D. Plenz, *Dendritic calcium encodes striatal neuron output during up-states.* J Neurosci, 2002. **22**(5): p. 1499-512.
16. Traub, R.D., et al., *A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances.* J Neurophysiol, 1991. **66**(2): p. 635-50.
17. Ermentrout, B., *Linearization of F-I curves by adaptation.* Neural Comput, 1998. **10**(7): p. 1721-9.
18. Freeman, W.J. and W. Schneider, *Changes in spatial patterns of rabbit olfactory EEG with conditioning to odors.* Psychophysiology, 1982. **19**(1): p. 44-56.
19. Freeman, W.J. and C.A. Skarda, *Spatial EEG patterns, non-linear dynamics and perception: the neo-Sherringtonian view.* Brain Res, 1985. **357**(3): p. 147-75.
20. Charpak, S., et al., *Odor-evoked calcium signals in dendrites of rat mitral cells.* Proc Natl Acad Sci U S A, 2001. **98**(3): p. 1230-4.
21. Sachse, S. and C.G. Galizia, *Role of inhibition for temporal and spatial odor representation in olfactory output neurons: a calcium imaging study.* J Neurophysiol, 2002. **87**(2): p. 1106-17.
22. Freeman, W.J., *Simulation of chaotic EEG patterns with a dynamic model of the olfactory system.* Biol Cybern, 1987. **56**(2-3): p. 139-50.
23. Chang, H.-J. and W.J. Freeman, *Parameter optimization in models of the olfactory neural system* Neural Netw. , 1996 **9** (1): p. 1-14
24. Kozma, R. and W.J. Freeman, *Basic principles of the KIV model and its application to the navigation problem.* J Integr Neurosci, 2003. **2**(1): p. 125-45.
25. Li, H. and R. Kozma, *A Dynamic Neural Network Method for Time Series Prediction Using the KIII Model.* IEEE, 2003(03).
26. Li, Z. and J.J. Hopfield, *Modeling the olfactory bulb and its neural oscillatory processings.* Biol Cybern, 1989. **61**(5): p. 379-92.

27. Sandefur, J.T., *Discrete Dynamical Systems - Theory and Applications*. 1 ed. 1990.
28. Lajmanovich, A. and J. A.Yorke, *A deterministic model for gonorrhea in a nonhomogeneous popul.* Mathematical Biosciences, 1976. **28**(3-4): p. 221-236.
29. MathWorks, *Simulink*: http://www.mathworks.com/products/simulink/.