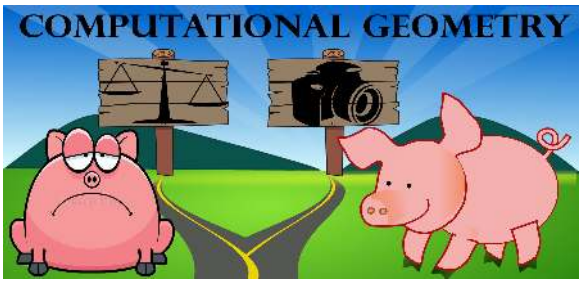
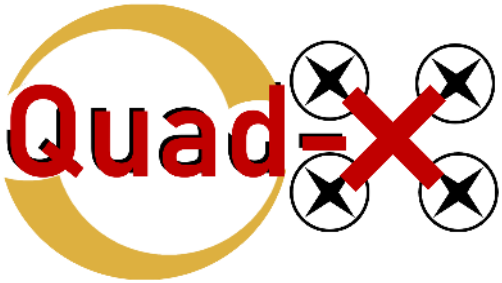
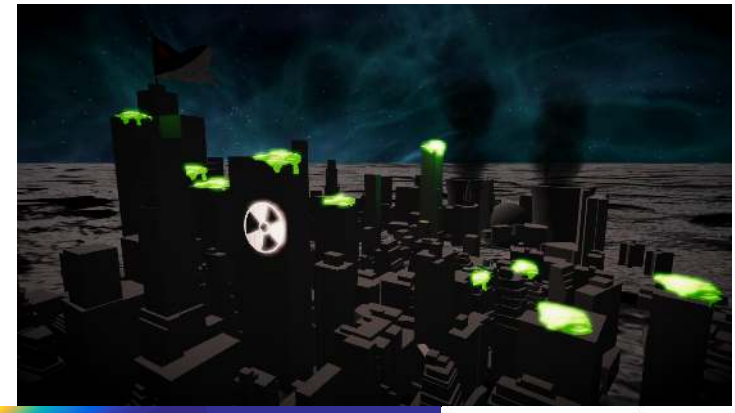
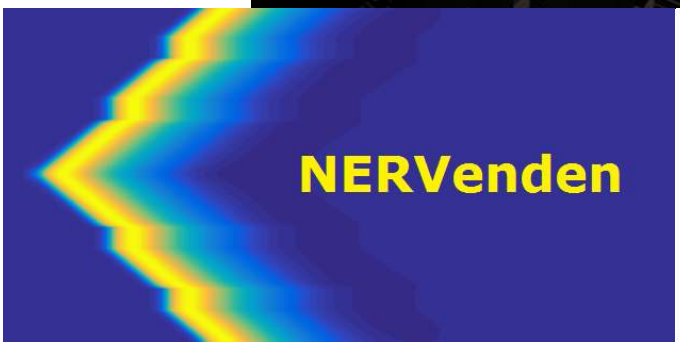
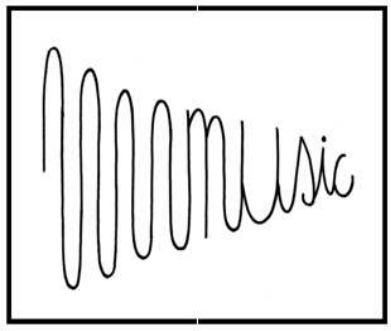
14. WochE der MODELLIERUNG mit Mathematik











Dokumentationsbroschüre 10.2. – 16.2.2018

WOCHE DER MODELLIERUNG MIT MATHEMATIK

PÖLLAU, 10.2.–16.2.2018

WEITERE INFORMATIONEN:

https://imsc.uni-graz.at/modellwoche/2018/

ORGANISATOREN UND SPONSOREN

KOORDINATION

Mag. DDr. Patrick-Michel Frühmann

Mag. Vanessa Peinhart

Alexander Sekkas

Vorwort

Viele Wissenschaften erleben zurzeit einen ungeheuren Schub der Mathematisierung. Mathematische Modelle, die vor wenigen Jahrzehnten noch rein akademischen Wert hatten, können heute mit Hilfe von Computern vollständig durchgerechnet werden und liefern praktische Vorhersagen, die helfen, Phänomene zu verstehen, Vorgänge zu planen, Kosten einzusparen. Damit unsere Gesellschaft auch in Zukunft mit der technologischen Entwicklung schritthält, ist es wichtig, bereits junge Leute für diese Art mathematischen Denkens zu begeistern und in der Gesellschaft das Bewusstsein für den Nutzen angewandter Mathematik zu heben. Dies war für uns einer der Gründe, die Woche der Modellierung mit Mathematik zu veranstalten.

Nun ist leider für viele Menschen Mathematik ein Schulfach, mit dem sie eher unangenehme Erinnerungen verbinden. Umso erstaunlicher erscheint es, dass Schülerinnen und Schüler sich freiwillig melden, um eine ganze Woche lang mathematische Probleme zu wälzen - und dabei auch noch Spaß haben. Sie erleben hier offensichtlich die Mathematik auf eine Art und Weise, wie sie der Schulunterricht nicht vermitteln kann. Die jungen Leute arbeiten und forschen in kleinen Gruppen mit Wissenschaftler/innen an realen Problemen aus den verschiedensten Bereichen und versuchen, mit Hilfe mathematischer Modelle neue Erkenntnisse zu gewinnen. Sie arbeiten ohne Leistungsdruck, dafür mit Eifer und Enthusiasmus, rechnen, diskutieren, recherchieren, oft auch noch am späten Abend, in einer entspannten und kreativen Umgebung, die den Schüler/innen und betreuenden Wissenschaftler/innen gleichermaßen Spaß macht. Die Projektbetreuer konnten auch in diesem Jahr wieder erleben, wie eigenes Entdecken und Selbstmotivation das Verhalten der Schüler/innen während der ganzen Modellierungswoche bestimmen. Sie lernen eine Arbeitsmethode kennen, die in beinahe allen Details den Arbeitsmethoden einer Forschergruppe entspricht. Bei keiner anderen Gelegenheit erfahren Schüler/innen so viel über Forschung wie bei so einer Veranstaltung.

Modellierungswochen gab bzw. gibt es zum Beispiel auch in den USA, in Deutschland oder in Italien. Wir verdanken Herrn Prof. Dr. Stephen Keeling den Vorschlag, auch durch die Universität Graz so eine Woche zu veranstalten, und seiner unermüdlichen Organisationsarbeit das tatsächliche Zustandekommen. Er leitet nun bereits zum 14. Mal diese inzwischen zur Institution gewordene Veranstaltung. Ihm sei an dieser Stelle noch einmal ausdrücklich und herzlich gedankt. Besonders wichtig war in den vergangenen Jahren auch die Unterstützung durch den langjährigen Mentor der Modellierungswoche, Herrn o.Univ.-Prof. Dr. Franz Kappel, der oft auch eine eigene Gruppe mit interessanten Problemstellungen betreut hat.

Wir danken dem Landesschulrat für Steiermark, und hier insbesondere Frau Landesschulinspektorin HR Mag. Christa Horn und Frau Fachinspektorin Mag. Michaela Kraker, für die Hilfe bei der Organisation und die kontinuierliche Unterstützung der Idee einer Modellierungswoche.

Finanzielle Unterstützung erhielten wir von der Karl-Franzens-Universität Graz durch Vizerektor Prof. Dr. Martin Polaschek und Dekan Prof. Dr. Christof Gattringer, vom regionalen Fachdidaktikzentrum für Mathematik und von Comfortplan.

Ohne den idealistischen, unentgeltlichen und engagierten Einsatz der direkten Projektbetreuer Florian Thaler, BSc, Richard Huber, BSc, Raphael Watschinger, BSc und Dipl.-Math. Dr. Carl Philip Trautmann – Institut für Mathematik und Wissenschaftliches Rechnen – hätte diese Modellierungswoche nicht stattfinden können.

Besonderer Dank gebührt ferner Herrn Mag. DDr. Patrick-Michel Frühmann, der die ganze Veranstaltung betreut und auch die Gestaltung dieses Berichtes übernommen hat, Frau Mag. Vanessa Peinhart für die tatkräftige Hilfe bei der organisatorischen Vorbereitung und Herrn Alexander Sekkas für die Hilfe bei der Betreuung der Hard- und Software.

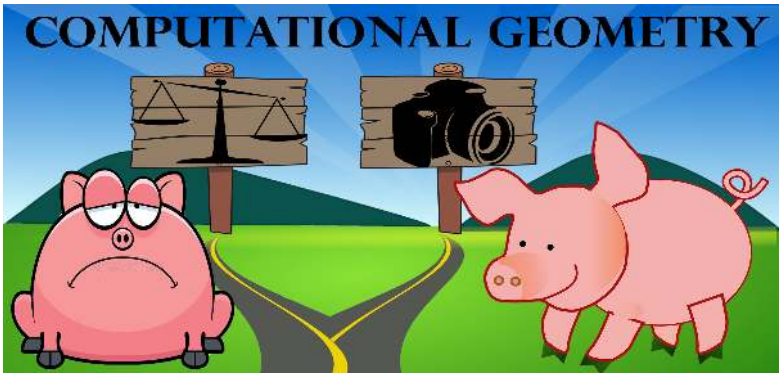
Pöllau, am 16. Februar 2018

Bernd Thaller
Institut für Mathematik und Wissenschaftliches Rechnen
Karl-Franzens-Universität Graz

WOCHE DER MODELLIERUNG

Projekt: COMPUTATIONAL GEOMETRY

Titel: Bilddatenanalyse



TeilnehmerInnen: Birchbauer Matteo, Ekam Felizia, Hernler Jakob, Plösch Simon, Wagner Phillip, Wurzinger Lena

Betreuer: Florian Thaler B.Sc

Inhaltsverzeichnis

1 Einleitung

Bei unserem Projekt beschäftigten wir uns mit der Frage, wie das Gewicht eines Schweines allein anhand von Fotos ermittelt werden kann. Die Inspiration dafür lieferte uns "Wuggl": eine Software, die mit optischer Körpermessung arbeitet und das Wiegen von Schweinen einfacher gestalten soll.

Wir setzten es uns zunächst zur Aufgabe, durch Computational Geometry die Fläche und den Umfang eines zweidimensionalen Objekts (unter anderem mithilfe eines rekursiven Verfahrens und eines Box-Counting Verfahrens) zu approximieren mit dem Ziel mithilfe dieser Informationen auf Größen wie Volumen oder Masse des zugrundeliegenden dreidimensionalen Objekts zu schließen.
Für die Erstellung unseres Fit-to-circle-Algorithmus, der eine Punktewolke (beziehungsweise ein fotografiertes Schwein) als einen möglichst ähnlichen Kreis darstellen soll, lernten wir außerdem einiges über partielle Ableitungen, also Ableitungen von Funktionen mit mehreren Variablen.

Bei der Anwendung unseres Verfahrens für die praktische Berechnung der Maße eines Schweins standen wir vor neuen Fragen: Wie approximiert man am besten das Volumen eines Objektes anhand seines Flächeninhalts und Umfangs? Wie finden wir ein möglichst einfaches Modell für ein Schwein?

2 Problemstellung

2.1 Problemstellung und Modellierung

Das Ziel unseres Projekts war die Lösung eines inversen Problems: Wir wollten Information über ein dreidimensionales Objekt aus einer zweidimensionalen Aufnahme extrahieren. Aufgrund der Komplexität unseres Problems haben wir eine vereinfachende Annahme getroffen: Wir gingen davon aus, dass das fotografierte Objekt die Form einer Kugel hat. Natürlich sind wir uns bewusst, dass nur wenige Objekte im echten Leben wie eine perfekte Kugel geformt sind. Allerdings hat unsere Modellannahme den entscheidenden Vorteil, dass im Grunde genommen lediglich ein Parameter (der Radius der Kugel) zur Berechnung des Volumens und in weiterer Folge des Gewichts unseres Objekts, sofern wir dessen mittlere Massendichte kennen, benötigt wird.

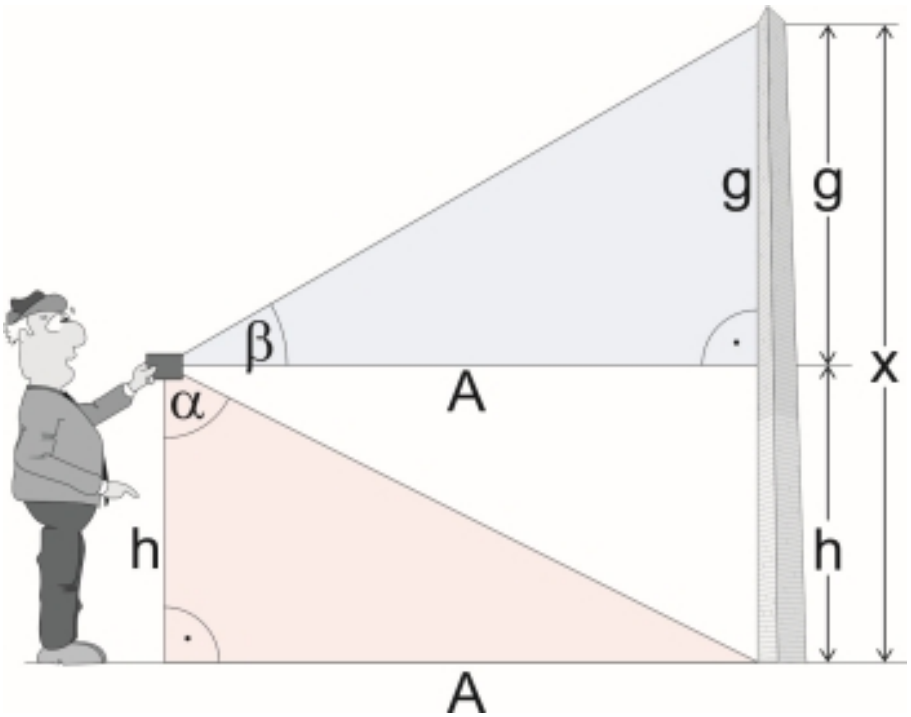
2.2 Längenmessung

Bei unserer Variante zur Berechnung von Längen von Körpern auf Fotos erhalten wir lediglich die Anzahl der Pixel von Beginn bis Ende des Körpers. Da ein Pixel aber nicht genau einer festgelegten Längeneinheit entspricht benötigen wir Möglichkeiten Pixel in Meter umrechnen zu können.

Die einfachere Variante ist es eine Referenzgröße mit abzubilden, von der man ein Verhältnis von Pixeln und Zentimetern aufstellen kann. Indem man neben den Körper von dem man die Länge ermitteln möchte ein Referenzmedium, wie einen Holzstab der 10cm lang ist legt, kann man festlegen, dass in diesem Foto, welches z.B. $1000 \cdot 100$ Pixel groß ist, die Anzahl der Pixel von dem Holzstab (z.B. 100) den 10cm entsprechen. Mit diesem Verhältnis 10 Pixel = 1cm ist es möglich jede beliebige Länge auf diesem Foto zu bestimmen.

Da man aber natürlich nicht immer eine Referenzlänge mit abbilden kann gibt es auch andere Wege die Länge bestimmter Dinge mittels eines Smartphones zu ermitteln. Hierfür machen sich diese Apps den Beschleunigungssensor unserer Smartphones zunutze. Positioniert man sein Handy in einer zuvor bestimmten Höhe h, meist 1.50m, errechnet die App durch den Winkel α des geneigten Smartphones mit Hilfe des Tangens die Entfernung A zum Objekt:

$$A = \tan(\alpha) \cdot h$$



Via $g = \tan(\beta) \cdot A$ wird anschließend die Höhe g berechnet. So lässt sich durch Addition von g und h die gesamte Länge des Objekts bestimmen.

Je größer die Winkel werden, desto größer werden auch die Messfehler bei der Berechnung der Größe. Daher ist es zu empfehlen die Winkel möglichst gering zu halten.

3 Numerische Approximation von Umfang und Flächeninhalt

3.1 Box Counting Verfahren

Als Einführung für das Programm Matlab, mit dem die meisten von uns noch nicht vertraut waren, begannen wir mit dem Zeichnen von einfachen Kreisen. Doch auch das klingt einfacher als es im Endeffekt war. Nach dem Erstellen und Einfärben der Kreise, bekamen wir die Aufgabe, den Flächeninhalt

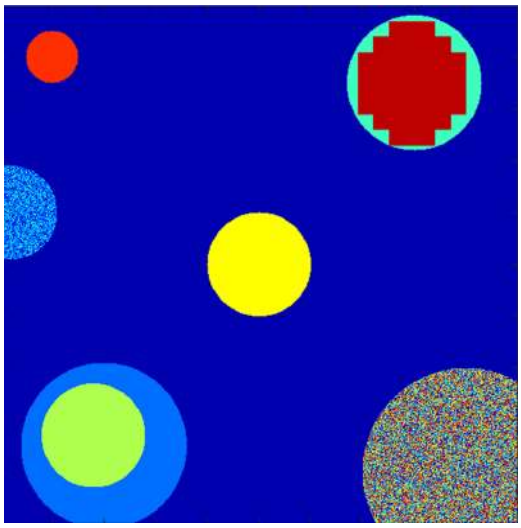


Abbildung 1: Unsere ersten Kreise

des Kreises zu berechnen. Als Lösung verwendeten wir ein Verfahren namens Box- Counting. Dieses Verfahren wird zur Datenanalyse verwendet indem ein großes Objekt (beispielsweise ein Flächenstück) in mehrere kleine kleinere Teilobjekte zerlegt wird, welche dann analysiert werden.

3.1.1 Flächenberechnung

Zur Approximation des Flächeninhalts einer gegebenen Fläche wird diese in viele kleine Flächenstücke, typischerweise Quadrate, zerlegt. Deren Flächeninhalt wird dann bestimmt und aufsummiert. Somit wäre auch die Form der Fläche irrelevant, da man es sowieso in kleinere Abschnitte zerteilt. Wir ver-

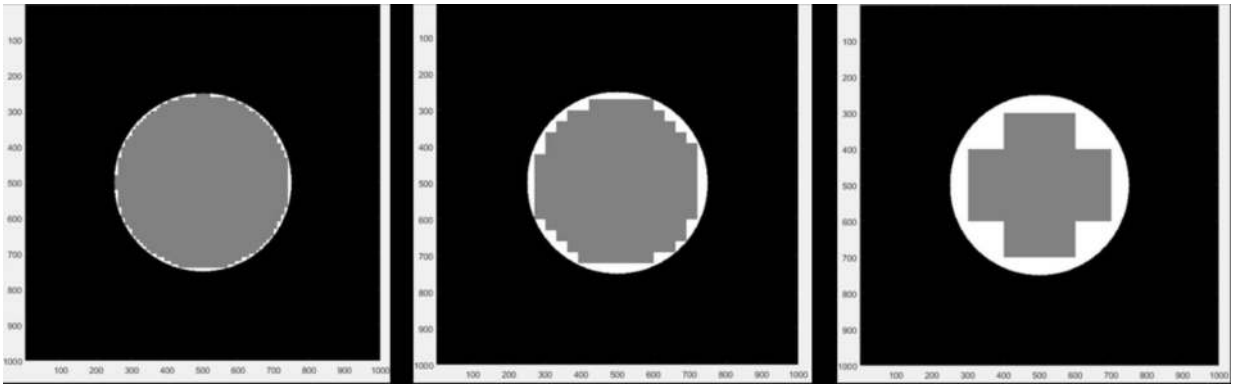


Abbildung 2: Approximation eines Kreises von innen mit Quadraten verschiedener Seitenlänge

suchten nun zunächst den erzeugten Kreis mit Quadraten zu füllen, welche alle dieselben Seitenlängen haben sollten. Hierzu ließen wir die ganze Bildmatrix mit Quadraten ausfüllen, doch nur jene, die auch wirklich im Kreis waren, wurden eingefärbt. Die Quadrate waren gezeichnet, doch der Flächeninhalt noch immer unbekannt.

Wir hatten zwar die Seitenlänge der Quadrate gegeben, doch die Anzahl der Quadrate war uns unbekannt. Als wir auch diese Aufgabe bewältig haben, stand schon die nächste an. Zwar funktionierte unser Modell mit der Voraussetzung, dass der Kreis circa in der Mitte liegt, doch sobald er sich zu weit nach rechts oder unten bewegte, zeigte uns das Programm eine zu kleine Fläche an. Der Grund für das Problem lag in der Festlegung der Seitenlänge der Quadrate und der Matrix. Denn blieb bei der Division - Seitenlänge der Matrix durch die der Quadrate - ein Rest übrig, so blieb auch eine Spalte und eine Zeile in der Matrix frei von Quadraten. Da wir aber nur die Quadrate abzählten, fehlte uns genau diese am Rand liegende Fläche, wenn der Kreis am unteren oder rechten Rand lag.

Eine Lösung für das Problem der fehlenden Quadrate war die Auffüllung der übrig gebliebenen Fläche mit Hilfe von Rechtecken. Die Länge der Rechtecke war die schon bestimmte Seitenlänge der Quadrate, die Breite mussten wir jedoch noch ausrechnen. Doch mit dem Rest der bereits oben beschriebenen Division war die Breite, welche nun die ganze Länge der Matrix genau ausfüllte, bereits gegeben. Somit hatten wir nun fast die ganze Fläche unseres Bildes ausgefüllt, teilweise mit Rechtecken aber hauptsächlich mit Quadraten.

Doch ein winziger Teil in der rechten unteren Ecke fehlte uns noch. Es war ein einzelnes Quadrat mit der Seitenlänge der Breite der Rechtecke. Aber mit diesen genauen Angaben, war auch diese Aufgabe kein großes Hindernis. Letzt endlich hatten wir nun die genaue Fläche unseres Kreises, ganz egal wo sich der Kreis befand, oder welcher Prozentsatz des Kreises in unserer Matrix zu sehen war.
Eine andere Art, das Problem der fehlenden Quadrate zu lösen, ist es, die Bildmatrix so zu erweitern, dass sowohl die Länge als auch die Breite ohne Rest durch die Seitenlänge der kleinen Quadrate dividiert werden kann. Der folgende Code vergrößert die Bildmatrix (hier A mit den Dimensionen N und M), wenn dies nötig ist:

```
1   sl=10;
2   tic;
3   area1= 0.0;
4   for i=1:sl:N-sl+1
5       for j=1:sl:M-sl+1
6           if(min(min(A(i:i+sl-1,j:j+sl-1)))~=0)
7               A(i:i+sl-1,j:j+sl-1)=2;
8               area1 = area1 + sl^2
9           end;
10      end;
11  end;
12  N1=floor(N/sl);
13  M1=floor(M/sl);
14  nsl=M-(M1*sl);

16  area2= 0.0;

18  j = N - nsl;
19  for i=1 : sl : N - sl +1
20    if(min(min(A(i:i+sl-1,j:end)))~=0)
21      A(i:i+sl-1,j:end)=2;
22      area2 = area2 + sl*nsl;
23    end;
24  end;
25  area3= 0.0;
26  i = M - nsl;
27  for j=1 : sl : N - sl +1
28    if(min(min(A(i:end,j:j+sl-1)))~=0)
29      A(i:end,j:j+sl-1)=2;
30      area3 = area3 + sl*nsl;
31    end;
```

```
32              end;
33              %      pr fe allerletztes quadrat;
34              area4= 0.0;
35              i = N-nsl
36              j = M-nsl
37              if(min(min(A(i:end,j:end)))~=0)
38                A(i:end,j:end)=2;
39                area4 = area4 + nsl*nsl;
40              end;
41              t=toc;
42              area = area1 + area2 + area3 + area4;
43              disp([ ' Area = ' , num2str(area)]);
44              disp([ ' Area1 = ' , num2str(area1)]);
45              disp([ ' Area2 = ' , num2str(area2)]);
46              disp([ ' Area3 = ' , num2str(area3)]);
47              disp([ ' Area4 = ' , num2str(area4)]);
48              disp([ ' Zeit = ' , num2str(t)]);
49              disp([ ' neue Seitenl nge = ' , num2str(nsl)]);
50              colormap gray;
51              imagesc(A);
52              axis image;
```

Programmauszug 1: Berechnung des Flächeninhalts via Box- Counting

3.1.2 Umfangsberechnung

Zur Berechnung des Umfangs des Kreises oder auch eines anderen Objekts konnten wir ebenfalls das Box- Counting Verfahren anwenden. Wir benützten, gleich wie bei der Berechnung der Fläche, wieder die Gliederung der Matrix durch Quadrate. Doch wir benötigten nicht die Anzahl aller Quadrate, sondern nur derer, die essentiell für die Berechnung des Umfangs waren - die außenliegenden. Des Weiteren mussten wir wissen, wie oft wir die Seitenlänge eines außen liegenden Quadrates zum Umfang dazu zählen sollten, da dies je nach Lage des Quadrates, unterschiedlich ist. Somit brauchten wir auch eine Formel, die zu jedem dieser Quadrate bestimmt, ob es mit keiner, einer, zwei, drei oder sogar vier Seitenlängen den Umfang des Kreises beschreibt. Um das herauszufinden, schaut sich das Programm bei jedem Quadrat die oben-, unten-, rechts-, und linksliegenden Nachbarn an und kontrolliert einerseits ob das Quadrat im zu kontrollierendem Kreis liegt und andererseits ob oder wie viele Nachbarn eine andere Farbe als der Kreis besitzen. Pro Nachbarsquadrat, das eine andere Farbe besitzt, muss man eine Seitenlänge der Quadrate zum Umfang addieren. Leider kann Matlab aber nicht außerhalb seiner Matrix kontrollieren, welche Farbe die Nachbarn haben, das betrifft also die äußeren Reihen. Da wir also ständig Fehlermeldungen von dem Programm bekamen, legten wir einen Rand um unserer Matrix und erweiterten diese, ohne aber das Suchfeld zu erweitern. Somit hatte die Matrix auch in den Außenzeilen Nachbarn, die von Matrix kontrolliert werden konnten.

Mit dieser Verbesserung berechnete das geschriebene Programm genau den Umfang des abgebildeten Kreises, jedoch weicht der Wert stark von einem ganz runden Kreis mit dem gleichen Radius ab, welcher nicht mittels Pixel gezeichnet wurde, je kleiner man die Seitenlängen der Quadrate definiert. Das liegt an der Oberflächenstruktur des Pixel-Kreises, welche an eine Treppe erinnert, und somit zu einem größeren Umfang beiträgt. Doch mit einer etwas größeren Seitenlänge näherte sich unser Ergebnis stark dem echten Wert an.

```
1           Aneu = zeros (N + 2*sl , M + 2*sl);
2           Aneu(1+sl:sl+N,1+sl:sl+M) = A(:,:);
3           umfangU= 0.0;
4           umfangR= 0.0;
5           umfangL= 0.0;
6           umfangO= 0.0;
7             for i=1+sl:sl:N+sl+1
```

```
8               for j=1+sl:sl:M+sl+1
9                 if (((min(min(Aneu(i:i+sl-1,j:j+sl-1)))==2)) &&
10                    (max(max(Aneu(i+sl:i+(2*sl)-1,j:j+sl-1)))~=2))
11                  umfangU = umfangU + sl
12                end;
13                if (((min(min(Aneu(i:i+sl-1,j:j+sl-1)))==2)) &&
14                    (max(max(Aneu(i:i+sl-1,j+sl:j+2*sl-1)))~=2))
15                  umfangR = umfangR + sl
16                end;
17                if (((min(min(Aneu(i:i+sl-1,j:j+sl-1)))==2)) &&
18                    (max(max(Aneu(i-sl:i-1,j:j+sl-1)))~=2))
19                  umfangO = umfangO + sl
20                end;
21                if (((min(min(Aneu(i:i+sl-1,j:j+sl-1)))==2)) &&
22                    (max(max(Aneu(i:i+sl-1,j-sl:j-1))))~=2)
23                  umfangL = umfangL + sl
24                end;
25              end;
26            end;
```

Programmauszug 2: Berechnung des Umfangs via Box- Counting

3.2 Rekursiver Ansatz

Rekursive Programmierung besteht darin bestimmte Abschnitte eines Programms so in einander zu verschachteln, dass lange und komplizierte Schleifen vermieden werden können. Dabei wird eine Funktion in sich selbst aufgerufen, mit neuen Parametern versehen und so ensteht eine sich immer weiter vertiefende Rekursion. Da das recht schnell zu einer theoretisch endlosen Wiederholung ausarten würde, ist eines der wichtigsten Elemente der rekursiven Programmierung die Abbruchbedingung. Sie ist dazu da um, wie der Name schon sagt, die Funktionsaufrufe abzubrechen wenn bestimmte Bedingungen erfüllt werden. Diese Bedingungen können, je nach Gebrauch, auf viele Arten definiert werden, beispielsweise als 'Zähler' durch den die Rekursion nur begrenzt oft aufgerufen werden kann, oder auch als 'Marker', wenn ein gesuchter Wert gefunden wurde.

```
1         zaehler = 0;
2         rekursion(zaehler)
3         void rekursion(int Zaehler){
4           if (zaehler >= 5)
5             return;
6           else
7             zaehler++;
8           rekursion(zaehler);
9         }
```

Programmauszug 3: Rekursive Programmierung- Ein Beispiel

Bei diesem vereinfachten Beispiel wird die Funktion 'rekursion' fünf mal aufgerufen

```
1         function A = quadrat(A, N, M, c)
2           global area;
3           global maxx;

5           Q1=A(1:N/2, 1:M/2);
6           Q2=A(1:N/2, M/2+1:M);
7           Q3=A(N/2+1:N, 1:M/2);
8           Q4=A(N/2+1:N, M/2+1:M);
9           if(c>9)
10              return;
11          elseif(max(max(A))==0)
```

```
12              return;
13          elseif(min(min(A))==maxx)
14              A(1:N, 1:M)=maxx/2;
15              area = area + M * N;
16              return;
17          elseif(min(min(A))==0 && max(max(A))==maxx)
18              c=c+1;
19              A(1:N/2, 1:M/2) = quadrat(Q1, N/2, M/2, c);
20              A(1:N/2, M/2+1:M) = quadrat(Q2, N/2, M/2, c);
21              A(N/2+1:N, 1:M/2) = quadrat(Q3, N/2, M/2, c);
22              A(N/2+1:N, M/2+1:M) = quadrat(Q4, N/2, M/2, c);
23              return;
24          else
25              disp('tut nicht ;-;');
26          end
```

Programmauszug 4: Rekursive Flächenberechnung

Diese Funktion stammt schon direkt aus einem unserer ersten Programme. Sie überprüft die Farbwerte der Pixel in der Bildmatrix 'A', wenn sowohl der größere als auch der kleinere Wert enthalten sind, teilt sie die Matrix in vier gleich große Quadranten und führt dieselbe Abfrage für die neuen Quadrate, beginnend links oben, erneut aus. Sollten wieder beide Werte vertreten sein wird auch das entsprechende Quadrat erneut geviertelt und so weiter. Sollte allerdings nur die Hintergrundfarbe, in diesem Fall '0' vorhanden sein wird das entsprechende Quadrat sofort verworfen, und die Funktion widmet sich dem nächsten. Wenn nur der gesuchte Wert gefunden wird, wird die betrachtete Fläche mit der Hälfte des Maximalwerts neu eingefärbt. Die oberste Abfrage verwendet den zuvor erwähnten 'Zähler' und sorgt dafür, dass die Funktion nicht mehr als neun mal ineinander ausgeführt werden kann, beziehungsweise nur bis die ursprüngliche Seitenlänge neun mal halbiert wurde.

3.3 Gift Wrapping

Hat man eine gewisse Menge an Punkten in einem Koordinatensystem, und möchte man eine konvexe Hülle um diese erstellen, welche von einem Außenpunkt zum nächsten geht, dann ist Gift-Wrapping das perfekte Verfahren. Um alle richtigen Punkte nach der Reihe herauszufinden wird folgendermaßen vorgegangen:

Erster Iterationsschritt: Als Startwert wird der Punkt mit dem kleinsten x-Wert gewählt. Nun beginnt das Programm Vektoren von unserem Startpunkt zu allen anderen Punkten im Koordinatensystem zu erstellen. Der Computer merkt sich einen zufälligen Vektor und misst den Winkel zwischen jenem Vektor und einem senkrechtem Vergleichsvektor. Danach misst er die Winkel aller anderen Vektoren zu dem vertikalen Vektor und merkt sich immer stets jenen, mit dem kleinsten Winkel zu dem Vergleichsvektor. Findet er einen Vektor mit einem noch kleineren Winkel, so wird der gemerkte Vektor durch den neuen ersetzt.

Hat das Programm alle Punkte mindestens einmal überprüft, so nimmt er den gemerkten Vektor, welcher den kleinsten Winkel mit dem vertikalen Vektor einschließt, da er durch keinen anderen Vektor ersetzt wurde aber alle anderen Vektoren bereits überprüft wurden, und verbindet den Startpunkt mit jenem ausgewählten Vektor.

Nächster Iterationsschritt: Der eben verbundene Punkt wird zum neuen Startpunkt und der neue Vergleichsvektor wird jener, welcher den neuen Startpunkt mit dem davor verbindet, in dem Fall der Vektor vom ersten zum zweiten Startpunkt. Das alles wird so lange als Schleife ausgeführt, bis der erste Startpunkt wieder zum Punkt wird, welcher als Vektor zum momentanen Startpunkt den kleinsten Winkel mit dem derzeitigen Vergleichsvektor einschließt.

```
[val,cur]=min(points(1,:))
hull=cur;
points=[points,points(:,cur)-[0;1]];

[cur,previous]=find_next(points,cur,length(points))
points(:,end)=[];
hull=[hull,cur];
hold on;
plot(points(1,[previous,cur]),points(2,[previous,cur]));

while cur ~=hull(1)
  [cur,previous]=find_next(points,cur,previous)
  hull=[hull,cur]
  plot(points(1,[previous,cur]),points(2,[previous,cur]))
end
```

Programmauszug 5: Gift- Wrapping Hauptprogramm

```
function [cur,previous] = find_next( points,cur,previous )
  for i =1:size(points,2)
    if i ~=cur && i~=previous
    v1=points(:,i)-points(:,cur);
    v2=points(:,cur)-points(:,previous);
    val=v1'*v2/norm(v1)/norm(v2);
    Val(i)=val;
    end
  end
  Val(cur)=-inf;
  Val(previous)=-inf
  previous=cur;
  [val,cur]=max(Val);
```

Programmauszug 6: Gift- Wrapping Funktion zur Suche des nächsten Punktes

Mit diesem Verfahren können wir versuchen den Umfang eines Kreises oder den Umfang eines eine Punktmenge einschließenden Polygonzuges zu bestimmen.

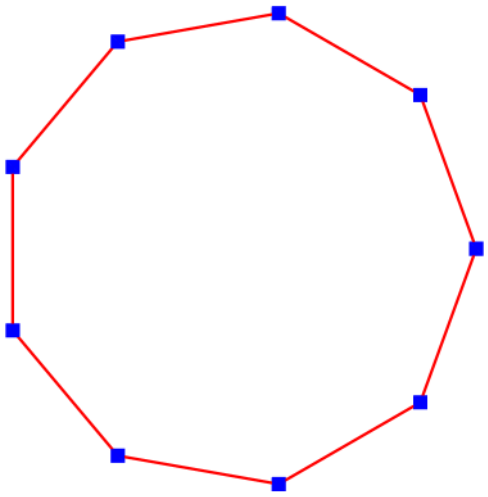


Abbildung 3: Approximation einer Kreislinie mittels Gift- Wrapping

3.4 Fit to Circle

Grundsätzlich wird der Fit to circle-Algorithmus dazu verwendet, aus einer Menge von Punkten $\{(x_i, y_i)\}_{i=1}^N$ den Kreis zu bilden, der diese Punktewolke am besten repräsentiert. Hierbei wird nach dem idealen Radius r sowie den idealen Koordinaten (x_0, y_0) für den Kreismittelpunkt gesucht. Diese Größen lassen sich durch Minimierung der folgendermaßen definierten Fehlerquadratsumme d minimieren:

$$\sum_{i=1}^{N}(\| \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \|_2^2 - r^2)^2 = d(x_0, y_0, r)$$

Ähnlich wie für Funktionen in einer Veränderlichen ist die gesuchte Stelle Teil all jener Stellen, an der sämtliche partielle Ableitungen erster Ordnung gleich Null sind.

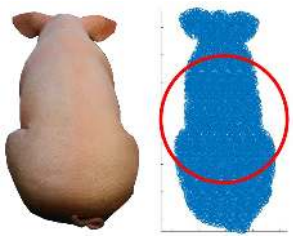


Abbildung 4: Links: Das Originalbild eines Schweins. Rechts: Die Approximation der Punktwolke des eingelesenen Bilds mithilfe eines Kreises.

3.4.1 Partielle Ableitung

Bei einer partiellen Ableitung handelt es sich um die Ableitung einer Funktion mit mehreren Variablen nach einer dieser Variablen, während die anderen wie Konstante behandelt werden. Partielle Ableitungen spiegeln das Änderungsverhalten einer Funktion in Richtung der Koodinatenachsen wieder.
Die partiellen Ableitungen der Fehlerquadratfunktion d sind

$$\begin{aligned}
\partial_{x_0} d(x_0, y_0, r) &= (-4) * [(x_0 - x_i)^2 + (y_0 - x_i)^2 - r^2]^2 * (x_0 - x_i) \\
\partial_{y_0} d(x_0, y_0, r) &= (-4) * [(x_0 - x_i)^2 + (y_0 - x_i)^2 - r^2] * (y_0 - y_i) \\
\partial_{r} d(x_0, y_0, r) &= (-4) * [(x_0 - x_i)^2 + (y_0 - x_i)^2 - r^2] * r
\end{aligned}$$

Nun werden die Gleichungen gleich Null gesetzt, um die gesuchten Minimalstellen zu bestimmen. Mit der Hand wäre dies ein sehr aufwendiger Prozess, daher lässt sich das Gleichungssystem schneller und leichter mit MatLab lösen:

```
1   % Deklaration der Datenpunkte als globale Variablen
2   global x;
3   global y;

5   % Festlegen eines Anfangswerts
6   x0 = [0;0;1];
7   % Numerisches L sen des Gleichungssystems mit fsolve
8   sol = fsolve(@myfunc, x0);

10  % Darstellung der Punktewolke und des eingeschriebenen Kreises
11  scatter(x,y);
12  hold on;
13  t = linspace(0, 2*pi, 100);
14  z1 = sol(3)*cos(t) + sol(1);
15  z2 = sol(3)*sin(t) + sol(2);
```

```
17          plot(z1, z2, 'r', 'LineWidth', 4);
```

Programmauszug 7: Fit to Circle- Hauptprogramm

```
1           function F = myfunc(w)

3             global x;
4             global y;

6             rhs1=0;
7             rhs2=0;
8             rhs3=0;

10            for i = 1 : 1 : length(x)
11                rhs1 = rhs1 + (-4) * [(x(i)-w(1))^2+(y(i)-w(2))^2-w(3)^2] * (x(i)-w(1));
12                rhs2 = rhs2 + (-4) * [(x(i)-w(1))^2+(y(i)-w(2))^2-w(3)^2] * (y(i)-w(2));
13                rhs3 = rhs3 + (-4) * [(x(i)-w(1))^2+(y(i)-w(2))^2-w(3)^2] * w(3);
14            end;

16            F(1) = rhs1;
17            F(2) = rhs2;
18            F(3) = rhs3;
```

Programmauszug 8: Fit to Circle- Funktion

Das Hauptprogramm ermittelt die besten Parameter für die eingegebene Funktion und zeichnet dann die Punktwolke sowie den Kreis. In der ausgelagerten Funktion werden die rechten Seiten des zu lösenden Gleichungssystems dargestellt, sodass mithilfe des MatLab- Befehls fsolve die Gleichungen gelöst werden können.

3.5 Optimierung/Multithreading

Ein Computerprozessor ist heutzutage aus mehreren sogenannten *Kernen* aufgebaut. Ein solcher Kern ist eine separate Recheneinheit, auf die verschiedene Programmabläufe aufgeteilt werden. Diese aufgeteilten Stücke von eigenständigen Aktivitäten eines Prozesses nennt man *Threads*. Jeder Thread besitzt den für die Aufgabe notwendigen Prozesskontext und kann selbst weitere Threads starten. Da immer mehrere dieser Threads ausgeführt werden spricht man von *Multithreading*.

Sehr überspitzt könnte man die Kerne als Wohngemeinschaft sehen und die Aufgaben, die jeder/jede einzelne erledigen muss, um die Wohngemeinschaft aufrecht und organisiert zu halten als Threads. Der Punkt an dem Ganzen ist, dass eine einzelne Person viel länger für die gleichen Aufgaben benötigt, als wenn diese auf mehrere Personen aufgeteilt werden und diese gleichzeitig daran arbeiten.

Multithreading in C++

In C++ gibt es ein Libary namens OpenMP das eine recht leichte Integration von Multithreading ermöglicht. Standardisiert findet die Ausführung eines C++-Programms auf nur einem Thread statt (Master-Thread). OpenMP ermöglicht einen nun die Aufteilung der Ausführung auf die gewünschte Anzahl an Threads.

Einbindung

Um mit OpenMP in C++ arbeiten zu können, müssen wir zunächst die Library einbinden. Dies funktioniert folgendermaßen:

```
1           #include <omp.h>
```

Außerdem muss man für den Compiler die Flag *-fopenmp* gesetzt werden.

Die Syntax

Alle OpenMP Elemente werden durch den Tag

```
# pragma omp
```

gekennzeichnet. Das Pragma-Statement bezieht sich normalerweise nur direkt auf das folgende Statement.

Parallel

Durch die Anweisung

```
# pragma omp parallel
```

wird ein parallel zu bearbeitender Programm block definiert. Der Block läuft auf der Anzahl der definierten Threads und wird auf jedem komplett ausgeführt. Auf welchem Thread was ausgeführt wird ist standardmäßig zufällig.
Die Programmzeilen

```
#pragma omp parallel
{
  int ID =  omp_get_thread_num();
  cout << "Das ist ein Thread >> ID: " << ID << "\n" << endl;
}
```

bewirken, dass in die Variable ID die jeweilige Id des Threads (Bei vier Threads von 0 - 3) geschrieben und diese dann mit dem Text "Das ist ein Thread ≫ ID:äusgegeben wird.

Es resultiert der Output

```
Das ist ein Thread >> ID: 1Das ist ein Thread >> ID:
D
as ist ein Thread >> ID: 0Das ist ein Thread >> ID:
3
2
```

Der Output demonstriert die Paralleliät sehr gut, denn man sieht das der Compiler den Block nicht nacheinander abarbeitet, sondern jeder Thread gleichzeitig seine Ausgabe ausgibt und es so zur Vermischung kommt. Daher ist es notwendig, dass die Ausgabe zuerst in einen *stringstream* geschrieben und aus diesem heraus ausgegeben wird:

```
#pragma omp parallel
{
  int ID =  omp_get_thread_num();
  stringstream ss;
  ss << "Das ist ein Thread >> ID: " << ID << endl;
  cout << ss.str();
}
```

Es entsteht der folgende Output:

```
Das ist ein Thread >> ID: 1
Das ist ein Thread >> ID: 2
Das ist ein Thread >> ID: 3
Das ist ein Thread >> ID: 0
```

Nun wurde der Block jeweils einmal pro Thread (bei vier Threads) ausgeführt.

Wichtig ist hierbei, dass der Block, der aufgeteilt werden soll, innerhalb von geschwungenen Klammern liegen muss. Außerdem darf die geschwungene Klammer nicht direkt nacht dem parallel geöffnet werden, sondern dies muss in der nächsten Zeile passieren.

Parallel for

Parallel for führt im Gegensatz zu parallel nicht einen ganzen Block auf jedem Thread separat aus, sondern teilt die Durchläufe einer for-Schleife auf die einzelnen Threads auf:

```
#pragma omp parallel for
for (int i = 0; i < 8; i++) {
  stringstream ss;
  ss << "Loop (round: " << i << ") on Thread "
  << omp_get_thread_num() << '\n';
  cout << ss.str();
}
```

Folgender Output entsteht:

```
Loop (round: 6) on Thread 3
Loop (round: 4) on Thread 2
Loop (round: 7) on Thread 3
Loop (round: 2) on Thread 1
Loop (round: 0) on Thread 0
Loop (round: 3) on Thread 1
Loop (round: 5) on Thread 2
Loop (round: 1) on Thread 0
```

Man sieht, dass die Schleife wie gewünscht acht Mal durchgelaufen ist, die einzelnen Durchläufe aber auf unterschiedlichen Threads ausgeführt wurden.

Weiterführend

Um ein besseres Verständnis dafür zu entwickeln, was die einzelnen Funktionen machen, ist es wichtig den Unterschied zwischen *parallel*, *parallel for* und *for* zu verstehen. Joel Yliluoma schreibt in einem Online-Guide eine gute Separation dafür:

1. A team is the group of threads that execute currently.
   a) At the program beginning, the team consists of a single thread.
   b) A *parallel* construct splits the current thread into a new team of threads for the duration of the next block/statement, after which the team merges back into one.
2. *for* divides the work of the for-loop among the threads of the current team. It does not create threads, it only divides the work amongst the threads of the currently executing team.
3. *parallel for* is a shorthand for two commands at once: *parallel* and *for*. Parallel creates a new team, and for splits that team to handle different portions of the loop.

If your program never contains a *parallel* construct, there is never more than one thread; the master thread that starts the program and runs it, as in non-threading programs.

Ausgewählte Directives

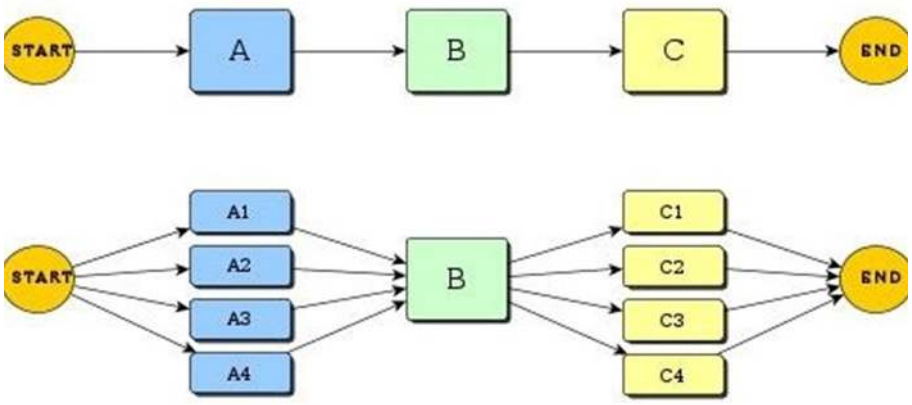


Abbildung 5: Hier wird die Aufteilung des Master-Threads in vier seperate Threads visualisiert.

- *critical* Spezifiziert, dass nur ein Code auf einem speziellen Thread zu gleichen Zeit ausgeführt wird.
- *single* Spezifiziert, dass Code auf einem bestimmten Thread ausgeführt werden soll, aber nicht unbedingt auf dem Master-Thread.

Funktionen

- *omp_set_num_threads(n);* Hiermit kann man festlegen auf wie viele Threads der Code aufgeteilt werden soll, wobei n die Anzahl der Threads ist.
- *omp_get_thread_num();* Liefert die Id des Threads, auf dem der Code momentan ausgeführt wird zurück.
- *omp_get_num_threads();* Liefert die Anzahl der aktiven Threads zurück.

3.5.1 Einsatz in der Bilddatenanalyse

Der Entscheidende Punkt, der über den Einsatz und die Einführung von Multithreading entscheidet, ist, wie bereits geschrieben, die Zeit. Die Bilddatenanalyse ist ein sehr zeitaufwendiger Prozess, da sehr viele Datenmassen überprüft werden müssen. Zur Verdeutlichung: Ein 512x512 Pixel Foto (was im Vergleich zum heutigen Stand der Technik ein lächerlich kleines Bild ist) besteht insgesamt aus 262 144 Pixeln, die einzeln überprüft werden müssen. Diese Analyse funktioniert in Anbetracht von moderner Prozessortechnik in zirka 60 Millisekunden (die reine Flächenbestimmung, ohne Bildeinlese und Visualisierung). In der Realität wird natürlich auch ein individuales Bild eingelesen werden, was wieder Zeit benötigt. Das Einlesen und die Berechnung eines 647x647 Pixel Bildes benötigt bereits um die 100 Millisekunden. Diese Zeiten sind noch sehr passabel, aber mit der Steigerung der Bildgröße steigt auch die benötigte Zeit enorm an. Daher bringt hier die Aufteilung des Prozesses auf mehrere Threads eine Zeiteinsparung. Zum Beispiel könnte jeder von vier Threads ein Viertel des gewünschten Bildes analysieren, was auch zirka zur Minimierung auf ein Viertel der benötigten Zeit führen sollte.

3.6 GraphicalUserInterface

Das GUI vereint alle vorhergehenden Funktionen in einer einfachen grafischen Oberfläche, um die Bedienung zu erleichtern. Der leicht verständliche Drag and Drop 'GUIDE' von Matlab beschränkt den Aufwand der Erstellung auf ein Minimum, die unterschiedlichen Buttons, Textfelder, Pop-Up-Menüs und weitere lassen sich durch selbst festlegbare 'tags' aufrufen und abfragen. Damit die grafische Oberfläche auch funktioniert, wird einiges an Code dahinter benötigt, doch sogar dessen Eingabe wird durch

Matlab erleichtert, indem jedem grafischen Element, mit dem im GUI interagiert werden kann, von selbst eine eigene Funktion zugewiesen wird. Bei einem Button beispielsweise kann in der betreffenden Funktion schnell und einfach festgelegt werden was passieren soll, wenn er gedrückt wird. In unserem GUI kann der Benutzer oder die Benutzerin zwischen vier Arten der digitalen Kreisberechnung wählen und hat beim Input die Wahl zwischen selbst eingegebenen Daten und dem Einlesen eines externen Bildes. Die Ergebnisse können für die zweidimensionale Kreisfläche neben einer veranschaulichenden Grafik, oder die daraus resultierende dreidimensionale Kugel, ebenfalls mit Grafik, ausgegeben werden.

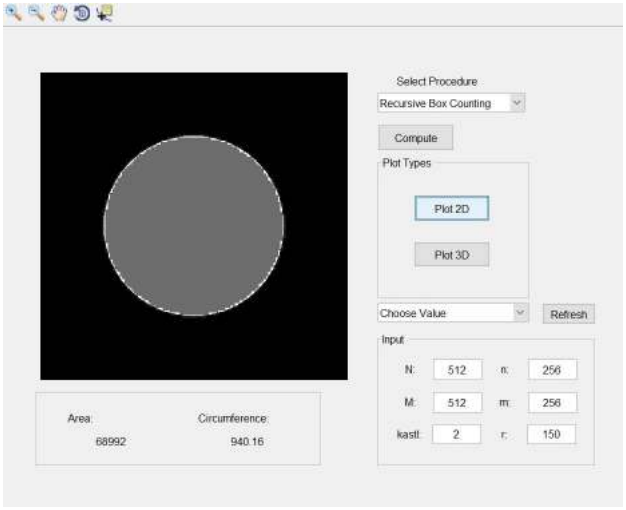


Abbildung 6: Zweidimensionales Objekt und Approximation

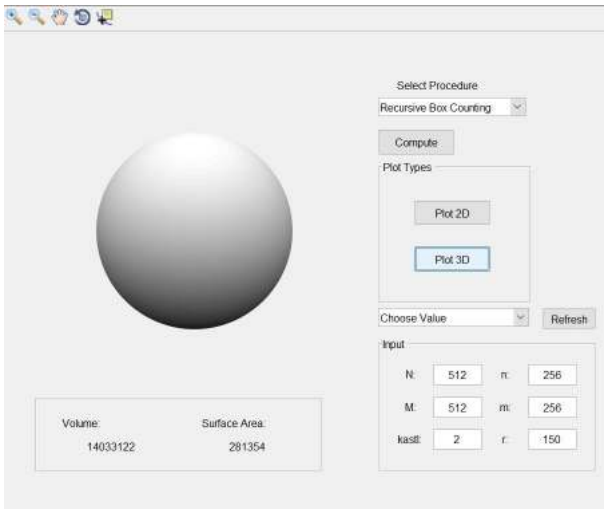


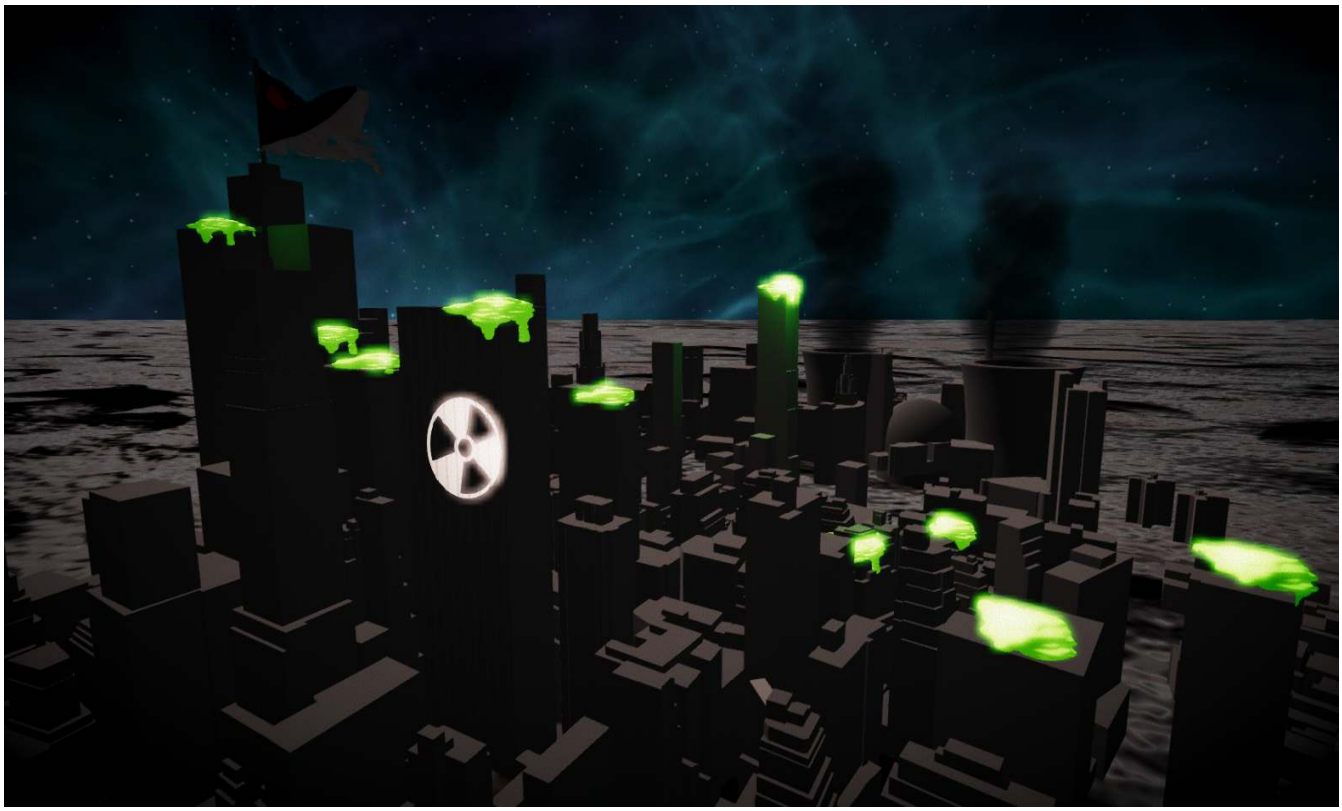
Abbildung 7: Rekonstruiertes dreidimensionales Objekt

Verbreitung von Radioaktiven Partikeln

Max Rappold, Max Schüßler, Mathias Sommerbauer,
Jana Landschützer, Konstantin Krimberger, Felix Windisch

Betreuer: Richard Huber

16. Februar 2018



Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Während den Tagen der Modellierungswoche beschäftigten wir uns mit radioaktiven Zerfällen von Teilchen, der Modellierung und Simulation dieser. Dabei lag der Fokus auf der Ausbreitung der Partikel in der Natur, den Einflüssen, welchen sie unterliegen und die Darstellung in festgelegten Systemen, sogenannten Compartmentmodellen. Die Grundlage der Modellierung war nicht nur die Beherrschung von Matlab, sondern auch das Verständnis von Radioaktivität und den Zerfallsgesetzen. Jeder der Teilnehmer näherte sich, aufbauend auf diesen Grundlagen, auf verschiedene Weisen den Aufgabenstellungen an, so wurden verschiedene Modelle geschaffen, die in den jeweiligen Kapiteln näher erläutert werden.

1.2 Grundlagen der Radioaktivität

Dieses Kapitel soll einen Überblick über das Thema und ebenfalls ein Grundverständnis zu radioaktivem Zerfall schaffen. Im allgemeinen Sprachgebrauch wird Radioaktivität als Strahlung angesehen welche aus Teilchen austritt. Der Fall ist jedoch, dass Radioaktivität eine Eigenschaft von instabilen Atomkernen ist, welche spontan ionisierende Strahlung aussendet. Diese austretende Strahlung ist nicht selbst radioaktiv, und hat nur eine kurze Reichweite. Die radioaktiven Stoffe jedoch, welche bei Zwischenfällen mit radioaktiven Stoffen auftreten, stellen eine nicht mindere Gefahr da, aufgrund ihrer Eigenschaften sich über große Distanzen hinweg zu bewegen. Radioaktivität entsteht bei dem Zerfall eines Atomkerns wobei verschiedene Arten von Strahlung unterschieden werden, und deren Auftreten von dem konkreten Atomkern abhängt. Austretenden Teilchen sind uns als Alpha- oder Betastrahlung bekannt, während Gammastrahlung Strahlungsenergie ist, welche ensteht, wenn ein Kern überschüssige Bindungsenergie (klassischerweise nach vorhergehender Abstoßung von Teilchenstrahlung) abgibt um sich zu stabilisieren. Der Zeitpunkt des Zerfalls eines Atomkerns tritt zufällig ein, jedoch besitzt jedes Isotop (Version eines Atoms mit unterschiedlicher Neutronenzahl) eine bestimmte Zerfallswahrscheinlichkeit mit welcher sich die Halbwertszeit jedes Nuklids bestimmen lässt. Dieser Zerfall ist näherungsweise exponentiell, was sich aus dem Gesetz der großen Zahlen und klassischen Lösungen von Differentialgleichungen herleiten lässt.

Ein Großteil der bekannten Nuklide sind nachgewiesenermaßen Radioaktiv, während nur ein kleiner Teil der Nuklide wirklich stabil ist, und diese Nuklide sind in Abbildung 1 schwarz dargestellt. Dabei hängt die Stabilität besagter Nuklide von dem Verhältnis von Neutronen zu Protonen im Kern ab. Diese Informationen sind essentiell für die Modellierung von Zerfall und Ausbreitung.

Zur Umsetzung dieser Informationen in Modelle kommen vom Einstieg in unsere Aufgabenstellung

bis zu den letzten Arbeitsschritten eine Vielzahl von mathematischen Verfahren zum Einsatz.

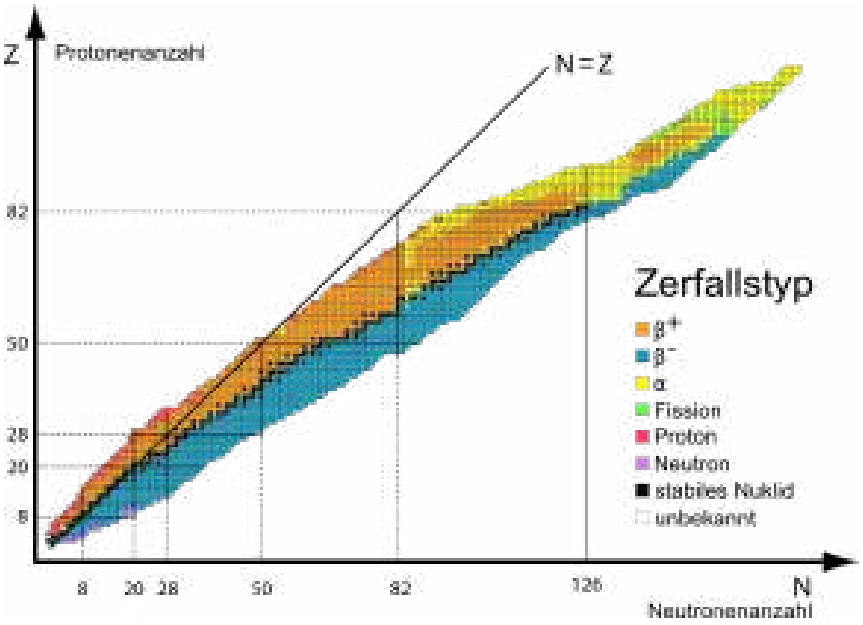


Abbildung 1: Zerfallstypen von Nukliden

2 Mathematische Grundlagen

2.1 Vektoren, Matrizen, Potenzen, Logarithmen und die Eulersche Zahl

Für unsere Arbeit mit MatLab verwendeten wir Vektoren und Matrizen als grundlegende Darstellungsmöglichkeiten der Ausgangs- und Ergebnisdaten. Weiters verwendeten wir zur Herleitung unserer Ausgangsformeln die Potenzrechnung mit positiven und negativen Hochzahlen, die Eulersche Zahl und in weiterer Folge den natürlichen Logarithmus.

2.2 Exponentialfunktionen

Nach einer Einführung in die Exponentialfunktionen arbeiteten wir mit exponentiellen Abnahme- bzw. Zerfallsfunktionen, leiteten Formeln für die Berechnung der einzelnen Parameter und der Halbwertszeiten her, ermittelten die Zerfallskonstanten einiger radioaktiver Stoffe und stellten deren Zerfallsgesetze auf. Aufgrund ebendieser Zusammenhänge tritt auch in vielen später presentierten Modellen und Resultaten exponentielles Verhalten auf.

N_0...Ausgangswert
Nt...Wert zum Zeitpunkt t
τ...Halbwertszeit
λ...Zerfallskonstante

$$N(t) = N_0 \cdot e^{\lambda \cdot t}$$

$$N(\tau) = \frac{N_0}{2}$$

$$\tau = \frac{\ln(0.5)}{\lambda}$$

$$\lambda = \frac{\ln(0.5)}{\tau}$$

2.3 Wahrscheinlichkeitsrechnung

Die Rechnung mit Wahrscheinlichkeiten fand in vielen Schritten unseres Arbeitsprozesses Anwendung. Die Tatsache, dass ein radioaktiver Zerfall durch eine exponentielle Abnahmefunktion dargestellt werden kann, beruht auf der Wahrscheinlichkeitstheorie, da der Zerfallszeitpunkt eines jeden Atoms zufällig und unabhängig von den Zerfallszeitpunkten aller anderen Atome ist. Um aus diesen Zerfallszeitpunkten Rückschlüsse auf ein allgemeines Gesetz der zeitlichen Entwicklung der Konzentration (1) zu erhalten, wird der Erwartungswert und das Gesetz der großen Zahlen verwendet.

Weiters kam die Wahrscheinlichkeitsrechnung bei der Arbeit mit dem Lagrange Ansatz direkt zum Einsatz, da wir die Diffusion mit Hilfe des Diffusionsparameters und einer zufälligen Zahl auf Basis einer Normalverteilung berechneten.

Auch bei vielen anderen Arbeitsschritten, wie zum Beispiel bei der Festlegung der Windgeschwindigkeit, der Windrichtung oder der Änderung dieser beiden Faktoren, griffen wir auf Zufallszahlen zurück, um unverhersehbare Ereignisse in die Modelle und Simulationen miteinzubeziehen.

$$\begin{aligned} x...Zerfallszeitpunkt& \\ P(x \geq 1) &= p \\ P(x \geq 2) &= p^2 \\ P(x \geq t) &= p^t \\ N(t) &= N_0 \cdot p^t \\ Zerfall\ ist\ exponentiell& \end{aligned} \tag{1}$$

2.4 Differentialrechnung

Um in weiterer Folge mit dem Eulerverfahren und dem Erwartungswert des radioaktiven Zerfalls arbeiten zu können, beschäftigten wir uns zuerst mit den Grundlagen der Differentialrechnung, wie dem Differenzenquotienten, dem Differentialquotienten und der Ableitung von verschiedenen Funktionen. Der Betrachtung von damit verbundenen Differentialgleichungen fällt eine besondere Rolle zu, da diese es ermöglichen die zeitliche Entwicklung einer Funktion anhand ihrer Steigung zu bestimmen.

$$N(t) = N_0 \cdot p^t$$

$$N'(t) = N(t) \cdot ln(p)$$

$$N(t) = N_0 \cdot e^{\lambda \cdot t}$$

$$N'(t) = N(t) \cdot \lambda$$

$$\lambda = ln(p)$$

$$Zerfallskonstante = Logarithmus\ der\ Zerfallswahrscheinlichkeit$$

2.5 Eulerverfahren

Wie bereits erwähnt lässt sich die Beschreibung von Funktionen auf deren zeitliche Zu- und Abnahme zurückführen, welche durch Differentialgleichungen dargestellt werden. Allerdings sind die resultierenden Differentialgleichungen oft nur mit großem Aufwand analytisch lösbar, oder entziehen sich gänzlich solcher Ansätze, weshalb numerische Mittel eingesetzt werden um approximative Lösungen zu erhalten. Das Konzept dieses Ansatzes wird im Folgenden anhand des Beispiels der linearen Abnahme

$$N' = -\lambda N \tag{2}$$

diskutiert. Zur Annäherung an eine exponentielle Abnahmefunktion unter der Voraussetzung, dass man nur den Ausgangswert der Funktion kennt, verwendeten wir das (explizite) Eulerverfahren. Dazu verwendet man ausschließlich den Ausgangswert (Startposition) der Funktion und eine beschreibende Differentialgleichung. Dabei nutzt man die Tangentengleichung

$$N(t + dt) \approx N(t) + dt \cdot N'(t). \tag{3}$$

an einem Punkt t als Approximation an die Funktion. Nach einer festgelegten Schrittweite wird erneut die Ableitung anhand der Differentialgleichung berechnet und mit dem anhand der Tangentengleichung bestimmten Punkt eine neue Tangentengleichung bestimmt. Mittels iterativer Anwendung dieser Prozedur erhält man schlussendlich eine Annäherungsfunktion $\tilde{N}$, welche sich in der Tat der exponentiellen Funktion anzupassen beginnt. Je kürzer man die Schrittweite, nach der eine neue Berechnung durchgeführt wird, festlegt, desto genauer wird die Annäherungsfunktion. In unserem Fall nutzten wir dieses Verfahren, um zu visualisieren, wie Zerfallsfunktionen für radioaktive Elemente entstehen. Die Berechnungen zur Annäherung an die Zerfallsfunktion führten wir unter der Verwendung von MatLab durch. Dabei nahmen wir eine beliebige Exponentialfunktion als unbekannt an und versuchten, durch die Erhöhung der Anzahl der Rechenvorgänge (Verkürzung der Schrittweite) eine präzisere Annäherung zu erhalten.

Der Wert der Funktion an der Stelle 0 (Ausgangswert) ist bekannt:

$$N(0) = N_0$$

Die allgemeine Ableitung einer exponentiellen Abnahmefunktion wird ebenfalls vorausgesetzt:

$$N'(t) = -\lambda N(t)$$

Die folgende Formel wird als Annäherung aufgestellt:

$$N(t + h) \approx N(t) + h \cdot N'(t)$$

Durch Einsetzen der Ableitung (welche aufgrund der Differentialgleichung bekannt ist) und Herausheben erhält man

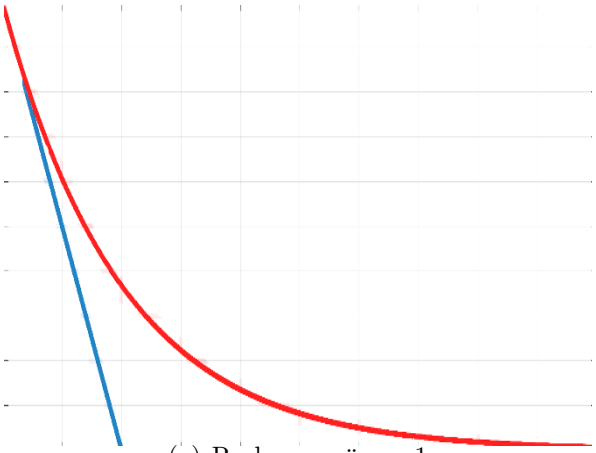
$$N(t+h) \approx N(t) - \lambda N(t)h$$

$$N(t+h) \approx N(t)(1-\lambda h).$$

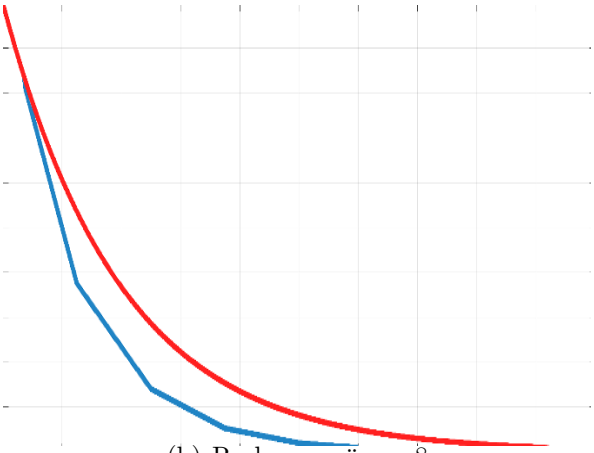
Im letzten Schritt fügen wir h als Variable für die Schrittweite hinzu und erhalten dadurch

$$\tilde{N}(k+h) = \tilde{N}(k) \cdot (1-\lambda h).$$

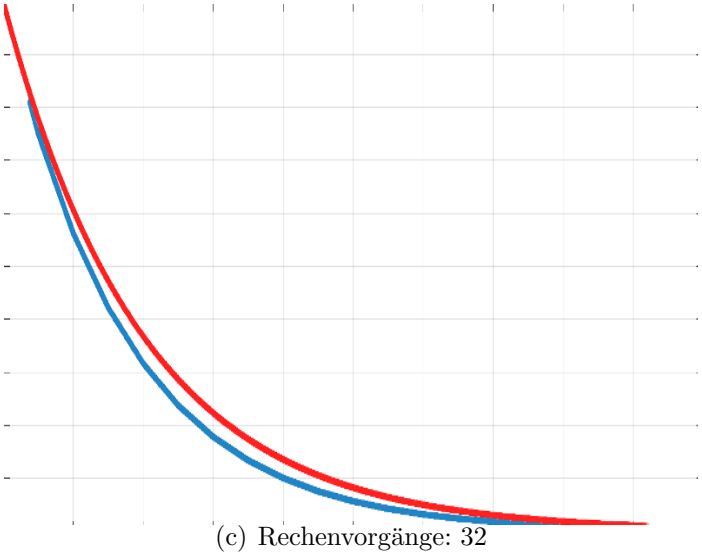
Die Abbildung 2 zeigt die Berechnung der Annäherungen zu einer Exponentialfunktion in Matlab und den Effekt unterschiedlicher Schrittweiten auf besagte Annäherung.



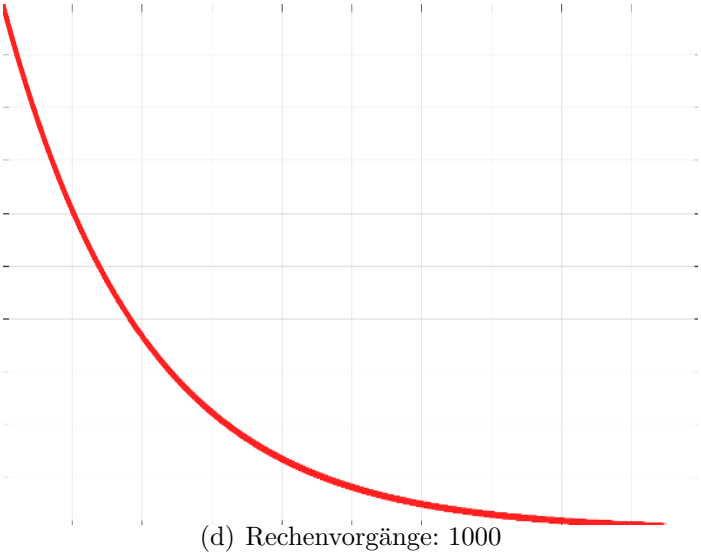
(a) Rechenvorgänge: 1



(b) Rechenvorgänge: 8



(c) Rechenvorgänge: 32



(d) Rechenvorgänge: 1000

Abbildung 2: Effekte unterschiedlicher Diskretisierungsschritte auf das Eulerverfahren

3 Partikelmodell

3.1 Annahmen

Wir wollen an dieser Stelle darauf hinweisen, dass die verwendeten Isotope nicht die einzigen auftretenden Nuklide sind, aber die relevantesten und aktivsten sind. Des Weiteren wollen wir anmerken, dass bis auf Strontium 90 alle Isotope nach dem Zerfall stabil sind. Yttrium 90 ist das Zerfallsprodukt des Strontium und deshalb auch in unserer Liste enthalten.

Wir haben in all unseren Berechnungen die folgenden Werte als Ausgang genommen:

Isotop	Halbwertszeit	Produkt	bq/mg
Cs 137	1497,6 Wochen	Ba 137	250000
Sr 90	1497,6 Wochen	Y 90	$4,6 * 10^{12}$
Cs 134	104 Wochen	Ba 134	1200
I 131	1,14 Wochen	Xe 131	150
Y 90	0,37 Wochen	Zr90	$3,2 * 10^{9}$

3.2 Modell

Das Pratikelmodell verfolgt den Ansatz, eine Gewisse Anzahl an einzelnen Partikeln möglichst repräsentativ für eine größere Mengen Partikeln zu simulieren. Dabei verfolgt man die Bewegung individueller Partikel, deren Bewegung erneut sehr grundlegenden Gesetzen über ihre Positionsänderung unterliegt. Vorteilhaft ist dies selbstverständlich nur wenn die Partikel tatsächlich repräsentativ für die breite Masse sind. In dem Fall ist das Modell aber schnell und nicht sehr rechenaufwendig. Das Modell stützt sich auf das Gesetz der großen Zahlen, welches aussagt, dass eine statistische Annahme an Genauigkeit gewinnt je mehr Werte man hat. Eine Umfrage ist repräsentativer, wenn mehr Personen befragt wurden. Selbiges gilt auch für das Partikelmodell.

3.3 Umwelteinflüsse

Einleitung: Im Falle eines Unfalls, bei dem eine beträchtliche Menge von Radioaktivität freigesetzt wird, ist der erste Schritt, die derzeitig herrschende Situation genauestens zu erfassen und zu analysieren. In weiterer Folge hat das Erstellen einer möglichst genauen Wettervorhersage oberste Priorität, da mit dieser eine Simulation für die weitere Verbreitung der Wolke erstellt werden kann. Um diese Simulation möglichst realitätsnah zu gestalten, müssen viele verschiedene Wetterfaktoren und Umweltfaktoren mit einbezogen werden.

Wind: Der Faktor Wind spielt bei der Verteilung radioaktiver Stoffe in der Luft die wohl ausschlaggebendste Rolle. Er ist in der Lage, riesige Wolken radioaktiv verseuchter Partikel viele Kilometer

weit zu transportieren. Teil unseres Projekt war es, genau diesen Einfluss des Windes auf solch eine Partikelwolke zu untersuchen und zu visualisieren. Bei unseren Simulationen ist es zum Beispiel möglich, den Wind aufgrund einer gegebenen Vorhersage voreinzustellen und dann dessen Auswirkungen auf die Partikelwolke zu beobachten. Des Weiteren ist es jedoch auch möglich, den Wind zufällig generieren zu lassen und auch so seine Auswirkungen auf die radioaktive Wolke zu beobachten.

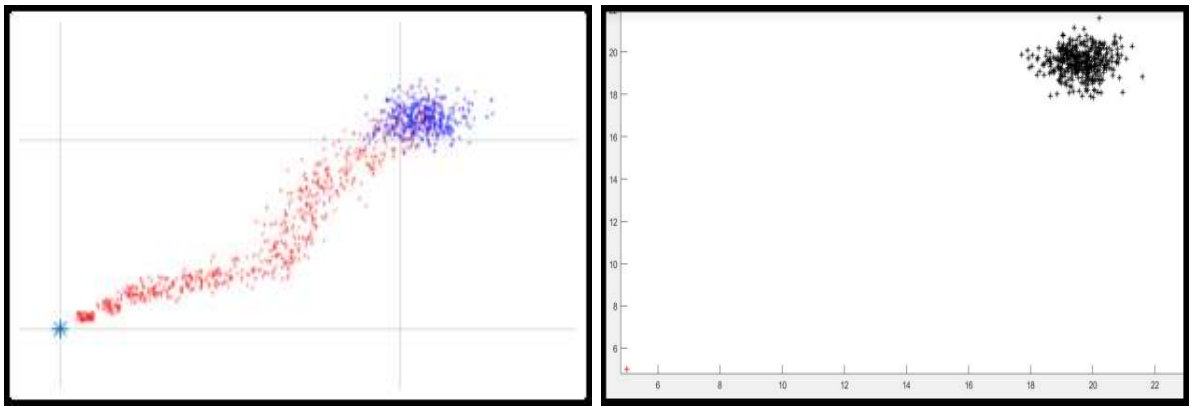
Regen: Regen spielt bei der Verbreitung der Radioaktivität insofern eine sehr wichtige Rolle, da er für eine verstärkte Ablagerung radioaktiver Stoffe am Boden verantwortlich ist. Dieses Phänomen ist sehr einfach zu erklären: Die einzelnen Regentropfen nehmen, während sie von der Regenwolke bis zum Erdboden fallen, radioaktive Staub- Rauch- oder Rußpartikel auf und transportieren somit die Radioaktivität zu Boden. Dies hat eine Kontamination des Erdreichs, der Flora, der Fauna, etc zur Folge. Jener meist stark radioaktiv verseuchte Regen wird in der Wissenschaft häufig als Fallout bezeichnet.

3.4 Lagrange Ansatz

Zur Berechnung der Bewegung einer radioaktiven Partikelwolke verwendeten wir den Lagrange Ansatz. Dabei wird jeder einzelne Partikel als Vektor, welcher die Position des Partikels zum Startzeitpunkt definiert, dargestellt. Um die Bewegung darzustellen wird der herrschende Wind als Vektor angenommen. Zu guter Letzt integrierten wir durch die Verwendung von Zufallszahlen eine Streuung der Partikel (Diffusion) in Abhängigkeit von der Zeit. Das Ausmaß der Streuung konnte durch verschiedene Diffusionsparameter angepasst werden. Die Simulation der bewegten Partikelwolke wurde von uns unter der Verwendung von MatLab durchgeführt. Abbildung 3 zeigt dabei auf der linken Seite die Bewegung radioaktiver Partikel (blaue Punkte) welche eine Spur zu Boden gefallener Partikel (rote Punkte) hinter sich herzieht. Die rechte Seite zeigt alleinig die Bewegung der Wolke, wobei in beiden Simulationen auch der radioaktive Zerfall Einzug gefunden hat.

$p'(t)$...Änderung der Position des Partikels zur Zeit t
$v(t)$...Windigeschwindigkeit zur Zeit t
$dif(t)$...Diffusion zur Zeit t

$$p'(t) = v(t) + dif(t)$$



(a) Partikelmodell mit Ablagerung: Blaue Punkte repräsentieren die Wolke, während die Roten abgelagerte Partikel dastellen. Der blaue Stern kennzeichnet den Startpunkt

(b) Bewegung der Partikelwolke

Abbildung 3: Illustration Partikelmodelle

4 Zellulärer Automat

Zelluläre Automaten sind dynamische Systeme, die in diskrete Zellen in einem Gitter aufgebrochen sind. Der Zustand des Systems wird durch eine 100*100 Matrix dargestellt, wobei jedes Element einen Quadratkilometer Land darstellt und der Wert die Konzentration an radioaktiven Stoffen in mg/km^2. An jedem simuliertem Zeitschritt (die Zeit wird in Stunden angegeben) wird die radioaktive Masse in einer Zelle von verschiedenen Faktoren beeinflusst:

4.1 Brownsche Bewegung

Brownsche Bewegung ist die scheinbar zufällige Bewegung von Teilchen in Abhängigkeit von ihrer Temperatur. Diese zufällige Bewegung führt zu einer Verbreitung der Teilchen. In der Simulation wird angenommen, dass überall die selbe Temperatur herrscht. Weiters ist die Verteilung nicht zufällig, sondern deterministisch. Um eine Verbreitung zu simulieren, verwenden wir eine $3 * 3$ "Einflussmatrix", die angibt wie und in welche Richtung sich die Partikel verteilen sollen. Für jeden Zeitschritt nimmt man für jede Zelle zuerst den linken oberen Nachbar und multipliziert ihn mit dem linken oberen Wert der Einflussmatrix, dann multiplizert man den oberen Nachbarn mit dem oberen Wert und so weiter für alle 9 Werte. Der mittlere Wert wird dabei mit der Zelle selbst multipliziert. Die Summe aller 9 entstandenen Werte ergibt den neuen Wert der Zelle. Um unabsichtliche Vermehrung oder Verminderung der radioaktiven Partikel zu verhindern, muss die Summe der Elemente der Einflu ss matrix 1 ergeben. Dies wird für alle 50.000 Zellen der Simulation durchgeführt. Um zu verhindern, dass die Veränderungen im selben Zeitschritt andere Zellen beeinflussen, verwenden wir einen Doppelbuffer. Die neuen errechneten Werte werden in einen seperaten Buffer geschrieben und erst wenn alle Werte berechnet wurden, werden die Buffer getauscht.

4.2 Wind

Ursprünglich wurde Wind als Gewicht in der Einflussmatrix angenommen, aber da dies zu rechenintensiv war, verwenden wir ein vereinfachtes Modell. Wir nehmen einen 2-dimensionalen globalen Windvektor an, der alle 10 simulierte Stunden zufällig dreht. Jeden Zeitschritt werden alle Elemente des Automaten um den Windvektor verschoben. Ist der Wind beispielsweise ein Vektor mit x = 1 und y =2, dann wird jede Zelle um 1 auf der X- und um 2 auf der Y-Achse verschoben. Dadurch können die radioaktiven Partikel auch aus dem 100 km^2 großen Feld geweht werden, warufhin sie für die Simulation als zerfallen gelten. Nimmt der Wind einen Wert kleiner als 1 an, so staut sich der Wind auf, bis man um eine ganze Zahl verschieben kann.

4.3 Zerfall

In unserer Simulation gibt es 5 verschiedene radioaktive Elemente: Caesium 137, Strontium 90, Caesium 134, Iod 131 und Yttrium 90. Für jedes Element gibt es einen eigenen zellulären Automaten. Die Automaten sind identisch, bis auf die unterschiedlichen Zerfallskonstanten und spezifischen Strahlungsdichten der Elemente. Dadurch zerfällt Caesium 137 mit einer Halbwertszeit von 29 Jahren viel langsamer als Yttrium 90 mit einer Halbwertszeit von 3 Tagen, aber strahlt mit einer spezifischen Strahlungsdichte von $2,5 * 10^5$ bq/mg schwächer als Yttrium 90 mit $3,2 * 10^9$ bq/mg. Radioaktiver Zerfall kann annähernd mit folgender Formel beschrieben werden:

$$N(t + x) = N \cdot e^{\lambda x}$$

Dabei ist N die Stoffmenge, t die Zeit in Stunden, x ist die Größe des Zeitschritts und λ ist die Zerfallskonstante. Für jeden Zeitschritt x wird also der Wert jeder Zelle mit $e^{\lambda x}$ multiplizert.

4.4 Untergrund

In unserer Simulation nehmen wir an, dass die Radioaktive Partikel eher auf der eigenen Höhe bleiben wollen, als abzusinken. Wenn die Simulation gestartet wird, wird ein Höhenprofil der Landschaft mithilfe von 2-dimensionalem Perlin Noise, quasi einem Hintergrundrauschen, wobei schwarze stellen Täler und weiße Stellen Berge sind, generiert. Im Simulationsschritt der Brownschen Bewegung wird auch der Untergrund miteinbezogen. Anstatt nur mit der Einflussmatrix wird jeder der 9 Werte auch durch den Betrag des Höhenunterschieds zwischen der Zelle und dem Nachbarn dividiert.

4.5 Aufbau

In Graphik 4 links oben sieht man die Konzentration von den jewiligen Elementen in mg/km^2. Dabei stellt die erste Graphik die Masse von Caesium 137, die zweite die Masse von Strontium 90 etc. da. Alle Graphiken addiert ergeben die Gesamtkonzentration. Rechts daneben kann man die Strahlung, die von den jeweiligen Elementen ausgeht sehen. Da Strontium 90 und Iod 131 um mehrere Zehnerpotenzen stärker strahlen als die anderen Elemente, wird hier die vierte Wurzel der Radioaktivität in bq/km^2 dargestellt. Rechts oben sieht man die gesamte Strahlung, also alle Ebenen zusammengezählt. Links unten kann man den derzeitigen Windvektor betrachten. Mitte unten sieht man die Oberfläche des Terrains und rechts daneben die Konzentration an zerfallenen Partikeln. Da immer wieder neue radioaktive Substanzen hinzukommen und die zerfallenen Partikel nicht verschwinden, steigt dieser Wert kontinuierlich an.

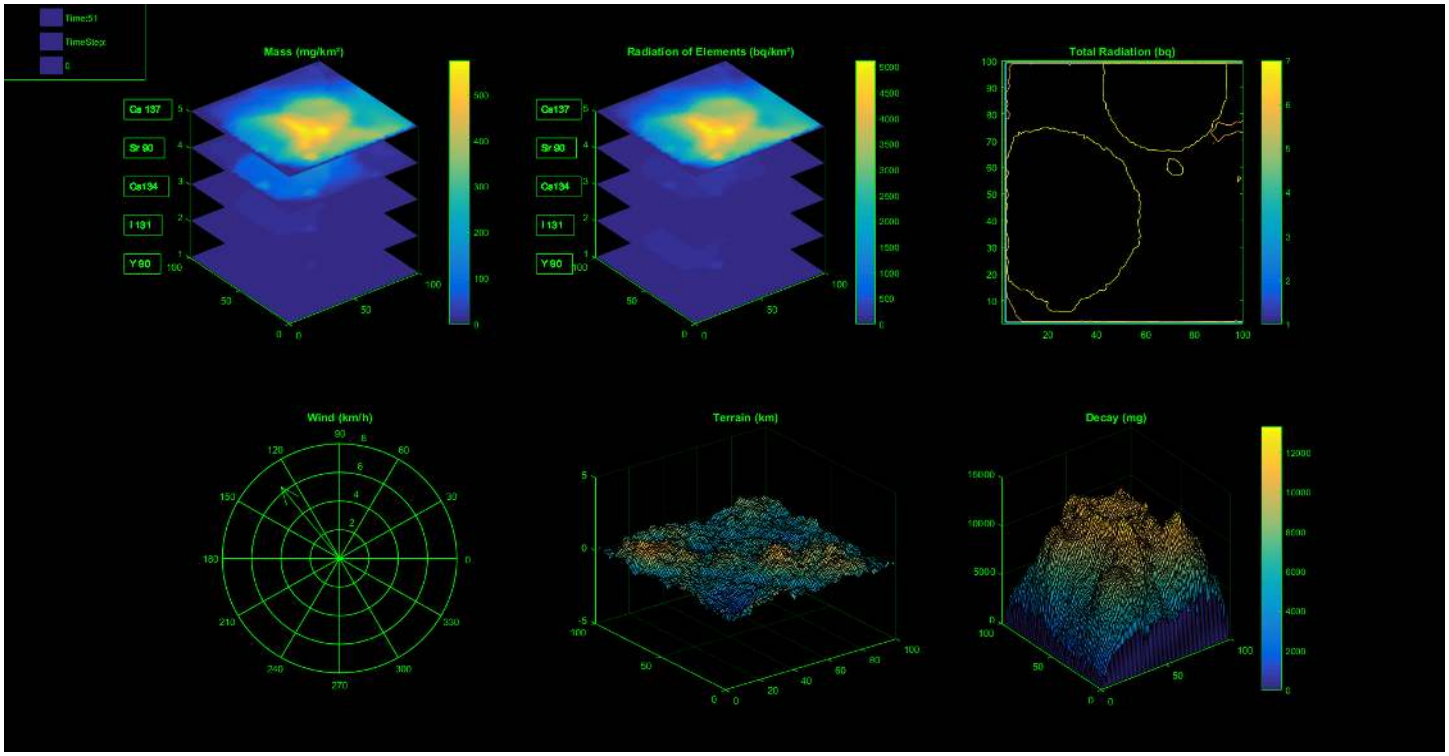


Abbildung 4: Illustration des zellulären Automaten: Links oben Masseverteilung, mitte oben Strahlungsverteilung, oben rechts Radionuklideverteilung, links unten Wind, unten mitte Terrain, unten rechts Zerfall

4.6 Abweichung von der Realität

In unserer Simulation werden viele Annahmen getroffen, die sich nicht vollkommen mit der Realität decken. So ist der Wind natürlich nicht überall exakt gleich und wird in der echten Welt auch von Bergen und Tälern gelenkt. Die Diffusion (der Prozess, in dem sich radioaktive Partikel am Boden ablagern und sich dadurch nicht mehr bewegen, aber trotzdem zerfallen) wird im Modell im Sinne der Einfachheit nicht direkt berücksichtigt, sondern findet in Form der Einflussmatrix und der damit verbundenen Rechenoperationen Einzug in das Modell. Weiters funktioniert die Realität weder in diskreten Zeitschritten, noch in diskreten Zellen. In der Wirklichkeit würden auch die zerfallenen Elemente irgendwann entweder verweht oder erodiert werden.

5 Compartmentmodell

5.1 Hintergrund

Das Compartmentmodel ist vorstellbar wie ein Komplex aus Wassertänken. Durch unterschiedliche Gegebenheiten wechselt eine Größe in unserem Beispiel radioaktives Material aus einem jener Tänke in einen anderen. Hier werden verschiedene Behälter radioaktiver Stoffe (z.B. Erde, Luft, Pflanze,...) betrachtet und es existieren Flüsse an Material zwischen ihnen. Ein Beispiel für einen solchen Wechsel ist Ablagerung radioaktiver Teilchen aus der Luft auf dem Boden. Auf diese Weise wechselt eine gewisse Menge aus dem Compartment Luft in das Compartment Erde. Je nach Komplexität kann man mit diesem Modell sehr genaue Werte erzielen. Abbildung 5 zeigt, wie mögliche Zusammenhänge solcher Compartments aussehen, und veranschaulicht damit, welche Transporte relevant sind.

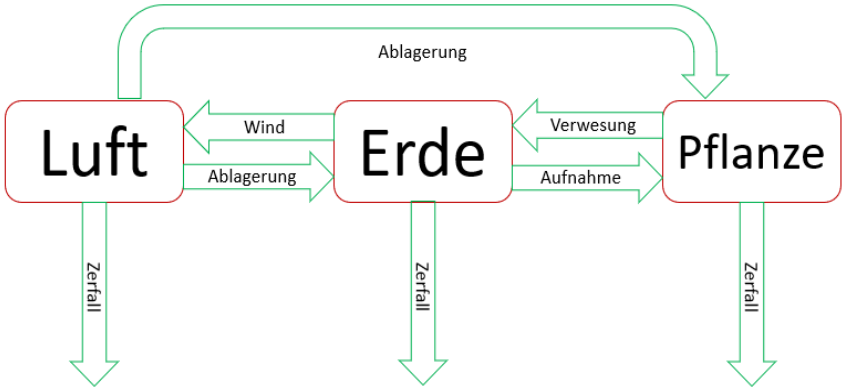


Abbildung 5: Illustration möglicher Zusammenhänge in einem Compartmentmodell mit den Compartments Erde, Luft und Pflanze

5.2 Modell mit Pilztod

Dieses Modell soll die Verteilung radioaktiver Teilchen in der Umwelt zeigen, vereinfacht dargestellt mit drei Compartments: jenes der Luft, der Erde und jenes der Pilze. Wir haben uns für Pilze entschieden, da sie im Vergleich zu beispielsweise Pflanzen, große Mengen an radioaktiven Teilchen aufnehmen. Am Anfang ist die gesamte Radioaktivität in der Luft, gespeist von einem Zufluss, der zum Beispiel von einem Reaktorunglück kommen könnte und der auf Grund vom Verfall radioaktiver Stoffe exponentiell abnimmt.

Aus der Luft lagern sich zirka 80% der Teilchen in der Erde und der Rest direkt auf dem Pilz

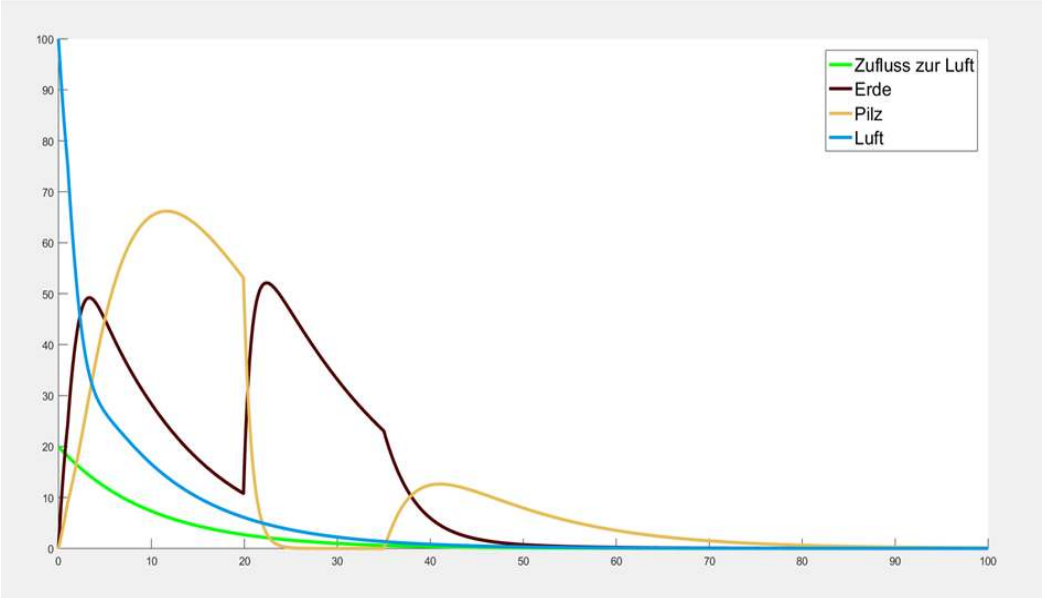


Abbildung 6: Zeitliche Entwicklung eines Compartmentmodells mit Pilztod

ab. Aber nachdem der Pilz den Großteil seiner Nährstoffe aus dem Boden bezieht, bekommt er in unserem Modell auch einen beachtlichen Anteil der verseuchten Teilchen aus der Erde. Zu beachten ist, dass alle Teilchen in diesem Modell demselben radioaktiven Zerfall ausgesetzt sind, da die Umwelt diesen nicht beeinflusst.

Bei 20 Zeiteinheiten, haben wir beschlossen, den Pilz verrotten zu lassen um ein möglichst realistisches Szenario zu simulieren. Die radioaktiven Teilchen die der Pilz in sich getragen hat, gehen dabei wieder in die Erde über. Etwas später, bei 35 Zeiteinheiten, wächst ein neuer Pilz, der wieder Teilchen aus dem Boden zieht. Gegen Ende der Zeitskala sind die meisten radioaktiven Teilchen zerfallen und auch die Quelle hat im gleichen Maße abgenommen. Deshalb nähern sich die Werte im Graphen Null an. Wie in Grafik 6 ersichtlich ist, geht bei dem Zerfall des Pilzes große Mengen wieder auf die Erde über, welche beim Auftreten des neuen Pilzes erneut absorbiert werden.

Erstellt haben wir dieses Modell mit einer zentralen Schleife, die den Zufluss und den vorgegeben Wert in der Luft auf die verschiedenen Compartments aufteilt. Da diese Übertragung von dem Zufluss über die Luft in die anderen Compartments natürlich nicht ohne Zeitverzögerung passiert, haben wir das auch berücksichtigt. Mit einer If-Abfrage mit einigen Elseifs, haben wir den Pilz sterben lassen, zu welchem Zeitpunkt, da keines der Teilchen vom Pilz aufgenommen wird, alle in die Erde gehen. Das wird so lange fortgesetzt bis ein neuer Pilz wächst, dann wird die Verteilung wieder so geregelt wie am Beginn der Simulation.

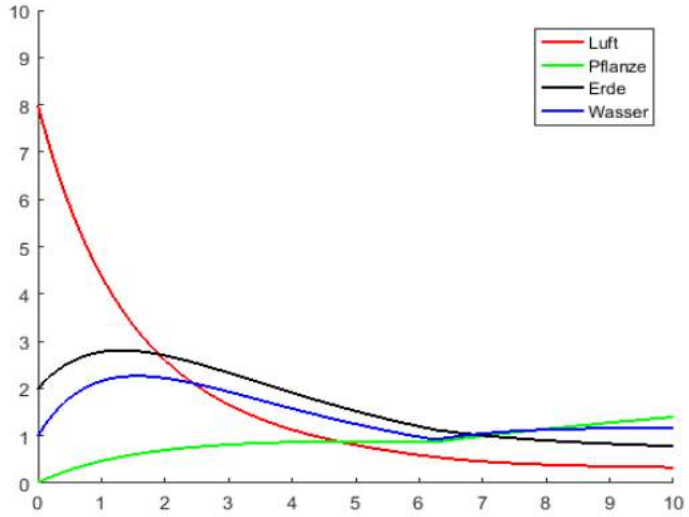


Abbildung 7: Zeitlicher Verlauf eines Modells mit fließendem Gewässer

5.3 Modell mit fließendem Gewässer

Ein weiteres Compartmentmodell zeigt die Verbreitung von radioaktiven Stoffen in einem Umweltsystem bestehend aus (fließendem) Gewässer, Erde, Luft und Pflanzen. Wie in dem vorangegangen Modell nimmt ebenfalls der Zufluss von radioaktiven Stoffen in der Luft mit der Zeit ab. Zudem sinkt die Konzentration der radioaktiven Stoffe des Wasser aufgrund des Abflusses. Nach der sechsten Zeiteinheit stoppt der Wasserabfluss jedoch. (Ein möglicher Grund ist der Bau eines Damms zur Verhinderung der Ausbreitung von radioaktiven Stoffen durch das Wasser.) Dies hat zur Folge, dass sich mehr Stoffe vom Wasser in den Boden ablagern und auch die Konzentration im Wasser wieder steigt. Zudem gibt es ein Pflanzenwachstum, was zur Folge hat, dass mehr Pflanzen Stoffe aus Wasser, Luft und Erde ziehen, was die Konzentration an radioaktiven Stoffen erhöht. All diese Prozesse können in Abbildung 7 beobachtet werden.

5.4 Interaktives Modell

Da die konkret auftretenden Parameter der Übergänge sehr stark von lokalen Bedingungen (Art der Erde, Pflanzenart, Jahreszeit...) abhängen, haben wir auch ein Interaktives Graphical User Interface geschaffen, in dem man adaptiv auf Situationen eingehen kann.

Unter `https://drive.google.com/drive/folders/1J9THr1_vgoPfHxAd_2s3KkHLbmr8bTLp?usp=sharing` können Sie ein Programm herunterladen, mit dem Sie die Paramter in einem 3-Kompartment-Modell beeinflussen können. Geben Sie einfach ihre gewünschten Werte mit den Schiebereglern ein, als Bestätigung dienen die Pfeile im unteren Graphen. Time gibt an für wie viele Zeitschritte die Simulation laufen soll. Zu große Werte können zu Abstürzen führen. Wenn alle Parameter gesetzt sind, startet der "Simulation" Knopf die Simulation und das Ergebnis wird in Abbildung 8 angezeigt.

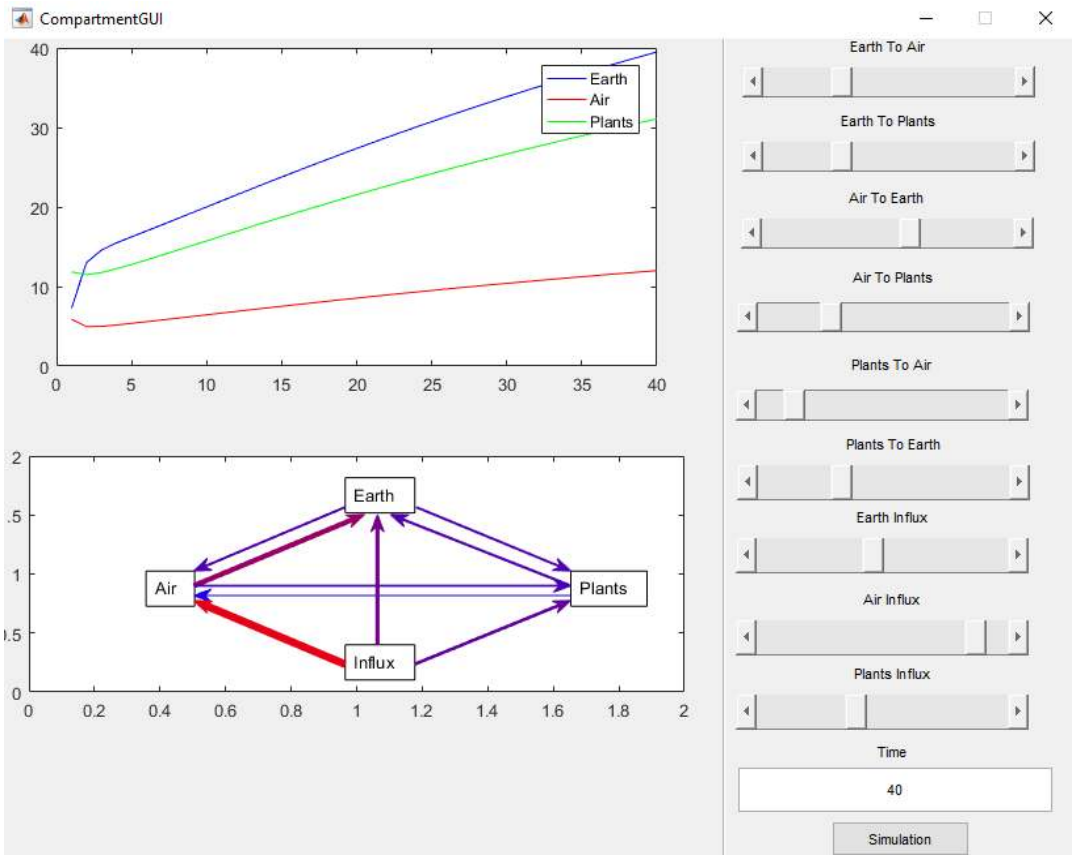


Abbildung 8: Graphical User Interface mit der Möglichkeit Parameter individuell anzupassen

6 Download

Sämtlicher Source Code kann unter
`https://drive.google.com/open?id=1mzkYkKlQr6xdMSX_2MsRnaoAGWTLtty6`
abgerufen werden.

Abbildung 9: Das strahlende Team von links nach rechts:
Max Rappold, Konstantin Krimberger, Max Schüßler, Richard Huber, Felix Windisch, Mathias Sommerbauer, Jana Landschützer

Klanganalyse und -synthese

Gruppenmitglieder:

Anna-Sophie Heibl, Carmen Kahl, Elias Götz, Florentin Stacher, Laura Reicher, Michael Neumann

Betreuer:

Raphael Watschinger, BSc.

Inhalt

Einleitung

Die 14. Woche der Modellierung der Mathematik fand vom 10.02.2018 bis zum 16.02.2018 in der Jugendherberge Pöllau statt. Bearbeitet wurden die Themengebiete Bilddatenanalyse, Radioaktive Kontamination in der Agrarwirtschaft, Mathematische Modellierung und Tuning eines Quadcopters, Ausbreitung von elektrischen Wellen in biologischen Geweben sowie Klangsynthese und -analyse.

Wir, Anna-Sophie Heibl, Carmen Kahl, Elias Götz, Florentin Stacher, Laura Reicher und Michael Neumann befassten uns unter der Betreuung von Raphael Watschinger BSc mit dem Thema Klangsynthese und -analyse.

Mathematik und Musik liegen sehr eng beieinander. Werden akustische und musikalische Grundlagen mit Informatik und Mathematik verknüpft, lassen sich Klänge digital analysieren und erzeugen.

In der heutigen Welt sind Klangsynthese und Klanganalyse in verschiedensten Bereichen der Musikindustrie nicht mehr wegdenkbar, denn dadurch bieten sich Möglichkeiten, Töne künstlich herzustellen oder auch natürliche Klänge abzuwandeln. Instrumente wie das Ondes Martenot, das Theremin oder das Trautonium waren Vorreiter der modernen Synthesizer, die in den 1960er Jahren erfunden und entwickelt wurden. Mit Hilfe verschiedener Varianten der Klangsynthese konnte man damit klassische Musikinstrumente wie Klavier, Gitarre, Streichinstrumente oder auch natürliche Klänge wie beispielsweise das Rauschen einer Welle imitieren. So wurden Synthesizer in den letzten Jahrzehnten immer mehr zu einem regelmäßig verwendeten Werkzeug in der Musikbranche. Die Klangsynthese und Klanganalyse entwickelten sich zudem ständig weiter, sodass immer realistischere Klänge erzeugt werden konnten.

Unser Ziel war es, verschiedene Klänge, beispielsweise den Klang einer Gitarrensaite, zu analysieren und mit Hilfe des Computers unter Verwendung verschiedener Methoden nach zu modellieren. Die Mittel und Methoden zur Klangerzeugung sind sehr vielfältig. In unserem Fall halfen uns *Matlab* und dort integrierte Funktionen, anfänglich Sinustöne und nach weitere Optimierung sehr wahrheitsgetreue Töne wiederzugeben.

Ein zweites Ziel des Projekts war es, eine bzw. mehrere Methoden zu entwickeln, um die Tonhöhe eines gegebenen Klangs möglichst genau zu ermitteln, und diese zu testen.

Einführung und Begriffserklärung

In diesem Kapitel gilt es einige Begriffe zu klären, die für das Verständnis der restlichen Arbeit grundlegend sind. Anfangs werden Bezeichnungen wie etwa Frequenz, Amplitude und Schwebung ausgeführt, die in der Musik verwendet werden. Anschließend werden die Mathematischen Grundlagen zum Aufbau eines Tons erklärt.

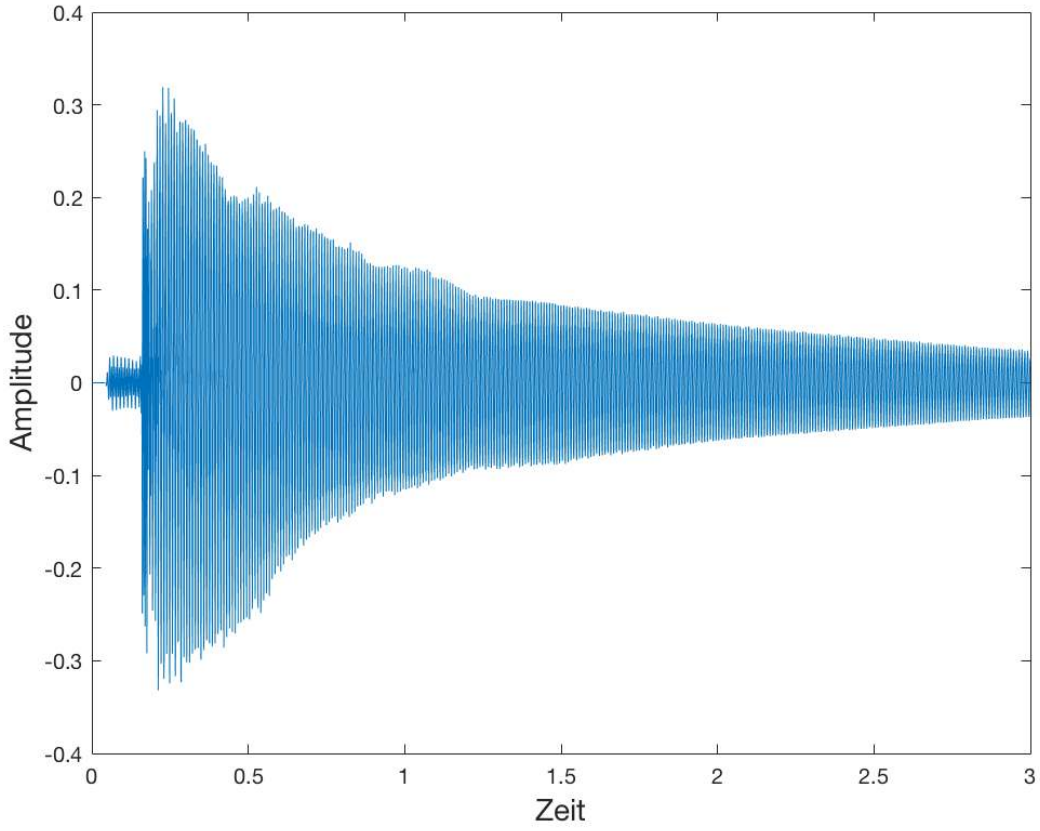
Frequenz:

Die Frequenz ist sowohl ein Maß in der Physik und Technik als auch in der Musik. Sie beschreibt die Schwingungszahl bei einem periodischen Vorgang pro gewählter Zeiteinheit, beispielsweise bei den Schwingungen der Luft, die ein Musikinstrument erzeugt und die wir als Klang wahrnehmen. Daher kann die Frequenz auch als Kehrwert der Periodendauer bezeichnet werden, welche die Länge einer vollständigen Schwingung beschreibt. Als Einheit der Frequenz wird Hertz (Einheitszeichen Hz = 1/s) verwendet. Der hörbare Bereich bei einem durchschnittlichen Menschen befindet sich zwischen 15 und 22000 Hz. Diese Zahl kann jedoch je nach Alter und Belastung variieren.

Die Tonhöhe hängt mit der Frequenz zusammen. Je höher die Frequenz ist, desto höher auch der Ton. Ein Ton besteht in den meisten Fällen jedoch nicht aus einer Frequenz, sondern aus vielen Teiltönen, wobei der tiefste als Grundton bzw. Grundfrequenz verstanden wird.

Amplitude:

Die Amplitude ist ein Begriff aus der Mathematik, Technik, Physik und Musik, um die maximale Auslenkung von Funktionen, von der Null-Linie ausgemessen, zu beschreiben. Bei Tonsignalen hängt diese mit der Lautstärke zusammen. Höhere Amplituden ergeben höhere Lautstärken. Zur visuellen Verdeutlichung, ist in der folgenden Abbildung der Klang der A-Saite einer Gitarre in Abhängigkeit der Zeit und der Länge der Amplitude ersichtlich. Im Diagramm erkennt man, dass beim Anzupfen der Gitarrensaite die Amplitude und demnach die Lautstärke des Tones sich sehr schnell erhöht. Je mehr Zeit nach dem Anschlagen verstreicht, desto kleiner wird die Amplitude und desto leiser wird der Ton.



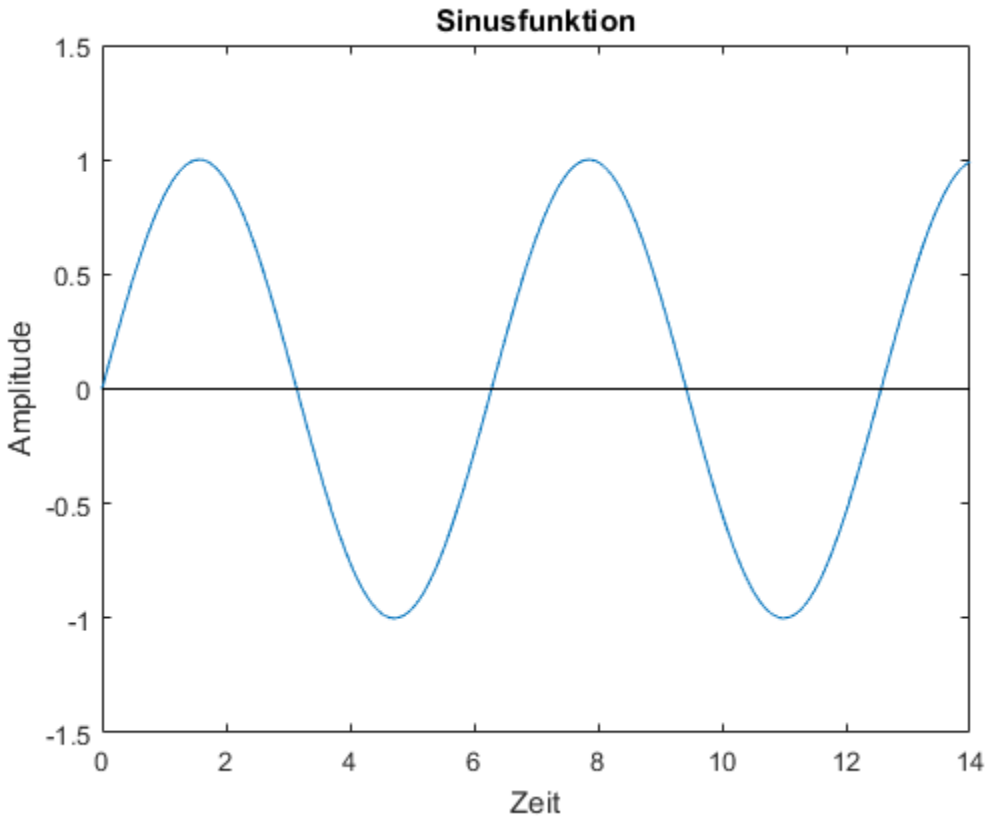
Schwebung:

Als Schwebung wird ein Effekt bezeichnet, welcher auftritt, wenn zwei Töne mit annähernd gleicher Frequenz zusammen erklingen. Es ist nur ein Ton zu hören, mit einer Frequenz, die dem Mittelwert der beiden ursprünglichen Frequenzen entspricht. Dabei hat man als Zuhörer den Eindruck, dass die Lautstärke schwankt.

Abtastrate:

Um ein analoges Signal digital einlesen zu können, muss dieses zunächst abgetastet werden. Die Abtastrate gibt die Anzahl der Werte einer Funktion in einem gewissen Zeitabstand an. Genau wie die Frequenz ist die Einheit der Abtastrate Hertz. Um einen Ton nach der Digitalisierung wieder vollständig wiedergeben zu können, muss die Abtastrate doppelt so groß gewählt werden wie die höchste Frequenz des Tons. Am häufigsten werden 44100 Abtastpunkte pro Sekunde gewählt, da dies nach dem Satz von Nyquist-Shannon ausreichend ist um Audiosignale im menschliche Hörbereich von 15 bis 22000 Hz zu erfassen.

Sinuston:



Ein Sinuston wird in der Akustik vereinfacht als Ton bezeichnet und kann mathematisch mit der Sinusfunktion beschrieben werden. Wie so ein Ton klingt, kann man hören, wenn man folgenden QR Code scannt.

Sinuston 440 Hz

Die allgemeine Funktionsgleichung lautet $f(x) = a * sin(b * x + c) + d$, wobei die Parameter a, b, c, d Einfluss auf Streckung/ Stauchung und Verschiebungen entlang der x- und y- Achse haben. Die y-Achse beschreibt die Amplitude und die x-Achse die Zeit. Der Sinus ist eine periodische Funktion, seine Werte wiederholen sich in regelmäßigen Abständen. Dieser Abstand wird als Periode bezeichnet. Der Parameter b ist hierbei die Kreisfrequenz. Wird b durch 2π dividiert, erhält man die Frequenz.

Arbeiten mit Sinustönen

Rechnen mit Frequenzen

Das erste Ziel des vorliegenden Projektes zum Thema Klanganalyse war es, die Grundlagen zum Arbeiten mit Sinustönen zu verstehen und zu erlernen. Dazu mussten zuerst einige Verbindungen zwischen Musik- und Mathematiktheorie festgelegt werden.

Musikalisch betrachtet besteht das Intervall (=Abstand zwischen zwei Tönen) der Oktave aus zwölf Halbtonschritten. Zudem muss man noch wissen, dass jeder Ton eine bestimmte Frequenz besitzt, welche dessen Tonhöhe bestimmt. Der Kammerton a^1, also jener Ton, welchen die Stimmgabel erzeugt, hat beispielsweise eine Frequenz von 440 Hertz. Allgemein lässt sich

festhalten, dass, wenn man die Frequenz eines Notenwertes verdoppelt, dieser sich genau um eine Oktave erhöht. Dasselbe Prinzip gilt auch bei Halbierung der Frequenz.

Möchte man nun einen Ton um i Halbtonschritte transponieren, seine Frequenz und Tonhöhe also verändern, ist dies durch Multiplikation der Frequenz mithilfe des folgenden Faktors möglich:

$$\sqrt[12]{2^i} = 2^{\frac{i}{12}}$$

Dieser ergibt sich aus dem Wissen, dass die Oktave aus zwölf Halbtonschritten besteht und durch Verdoppelung der Frequenz der Ton doppelt so hoch wird. Indem man also die zwölfte Wurzel vom Multiplikationsfaktor 2 bildet, und das Ergebnis mit einer gegebenen Frequenz multipliziert oder dividiert, wird der Ton um einen Halbtonschritt erhöht oder erniedrigt. Möchte man beispielsweise den Kammerton a^1 nun um eineinhalb Ganztonschritte zum c^1 erhöhen muss man dessen Frequenz, also 440 Hz, mit $2^{\frac{3}{12}}$ multiplizieren, um die Frequenz des Tons c^1 zu erhalten.

Aufgaben, um mit Matlab vertraut zu werden

Unsere erste Aufgabe beim Programmieren mit Matlab war es, kleine Musikstücke oder Tonfolgen mithilfe von Sinustönen zu erzeugen. Die Frequenz des a^1 transponierten wir wie oben erwähnt durch Multiplikation mit dem Faktor $2^{i/12}$ in die jeweilig gewünschten Tonhöhen. Eines der ersten Ergebnisse beim Transponieren von Frequenzen war eine Version des Lieds „Heart and Soul“ von Hoagy Carmichael

```
Fs=44100;
Freq_a=440;

Freq_c=Freq_a*2^(3/12);
Freq_e=Freq_a*2^(7/12);
Freq_a1=Freq_a;
Freq_f=Freq_a/2^(4/12);
Freq_g=Freq_a/2^(2/12);
Freq_h=Freq_a*2^(2/12);

time=0:1/Fs:0.25;
signal_1=sin(2*pi*Freq_c*time);
signal_2=sin(2*pi*Freq_e*time);
signal_3=sin(2*pi*Freq_a1*time);
signal_4=sin(2*pi*Freq_f*time);
signal_5=sin(2*pi*Freq_g*time);
signal_6=sin(2*pi*Freq_h*time);
signal_7=zeros(1,Fs*0.3);

signal = [signal_1, signal_7, signal_1, signal_2, signal_7, signal_2, signal_3, signal_7, signal_3, signal_1, signal_7,signal_1,
signal_4, signal_7, signal_4, signal_3, signal_7, signal_3, signal_5, signal_7, signal_5, signal_6, signal_7, signal_6];

soundsc(repmat(signal,1,4),Fs)
```

In der Abbildung kann man nun die zur Wiedergabe der Melodie benötigten Codeeingaben sehen. „Fs“ steht hier und im Folgenden für die Abtastrate. Als Ausgangston wurde das a^1 verwendet, mit welchem die benötigten Frequenzen der anderen Töne bestimmt wurden. Der Zeitvektor „time“ enthielt Zeitschritte bzgl. welcher die zu den verschiedenen Tönen gehörenden Sinusfunktionen abgetastet wurden. Um die einzelnen Töne nun zu einer Melodie zusammenzufügen, werden sie in dem Vektor „signal“ mit der passenden Tonabfolge hintereinander gereiht und mit dem Befehl „soundsc“ abgespielt. Das Ergebnis kann man sich anhören, wenn man folgenden QR-Code scannt.

Heart and Soul

Klanganalyse

Darstellung

Um Töne darzustellen, gibt es prinzipiell zwei verschiedene Möglichkeiten. Zum einen, gibt es die Darstellung der Amplitude nach der Zeit und zum anderen, die Darstellung der Amplitude nach der Frequenz.

Die Darstellung der Amplitude nach der Zeit wurde im oberen Abschnitt bereits erläutert. Man sieht in dieser Abbildung lediglich, wie laut der Gitarrenton, bei welchem sich mehrere Frequenzen überlagern, zur gegeben Zeit ist. Daraus kann man jedoch nicht schließen, wie laut

die einzelnen im Ton enthaltenen Frequenzen sind. Da dies für eine wahrheitsgetreue Reproduktion eines Gitarrentons unumgänglich ist, muss man den Ton anstatt jener zuvor verwendeter Parameter „Amplitude“ und „Zeit“ anhand der Parameter „Amplitude“ und „Frequenz“ wiedergeben. Dies ermöglicht die Fouriertransformation.

Fouriertransformation

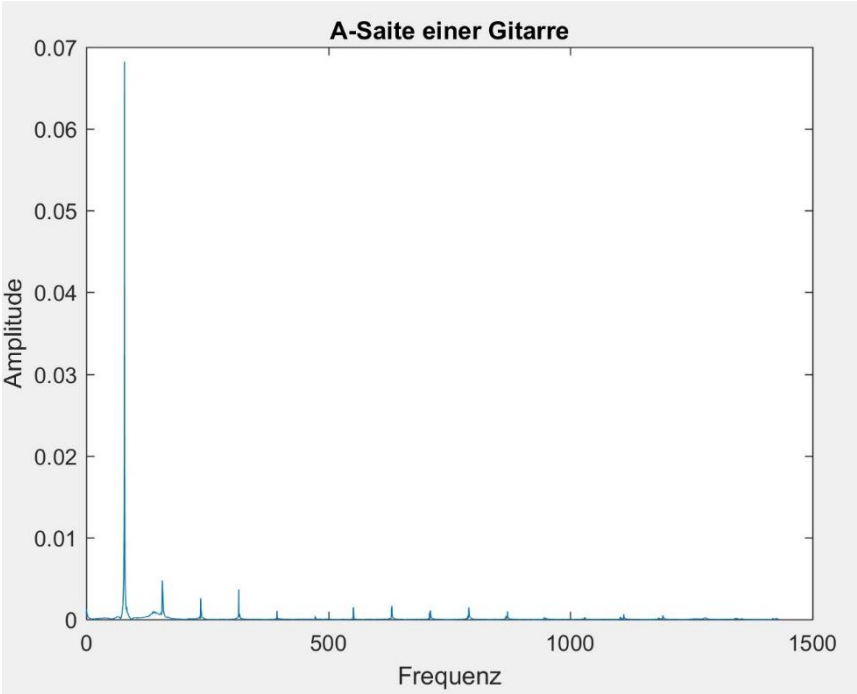
Im Folgenden sieht man die Formel für die von uns angewandte diskrete Fouriertransformation. Dabei gilt es zu beachten, dass die Formel den Vektor „â“ an der Stelle „k“ als Ergebnis liefert.

$$\hat{a}_k = \sum_{j=0}^{N-1} e^{-2\pi \mathrm{i} \cdot \frac{jk}{N}} \cdot a_j \quad \text{für} \quad k = 0, \ldots, N-1.$$

In der ersten Zeile des folgenden Codes wird die Wave-Datei „gitarre_a.wav“ eingelesen, die den a-Ton einer Gitarre enthält, und zu einem Vektor konvertiert, welcher die Amplitude der Audiodatei in Relation mit der Zeit enthält.

```
[signal,Fs] = audioread('gitarre_a.wav');
N = length(signal);
signal = signal(:,1)';
signal = signal(1:3*Fs);
fb2_guitar = fft(signal);
fb1_guitar = fb2_guitar(1 : 1 + N/2 + 1);
fb1_guitar = abs(fb1_guitar)/N;
fb1_guitar (2:end-1) = 2 * fb1_guitar(2:end-1);
freqs = Fs/N*(0:N/2+1);
```

In der fünften Zeile des obigen Codes wird die diskrete Fouriertransformation auf das Signal angewendet, die in Matlab durch die Funktion „fft“ gegeben ist. Die nach der Fouriertransformation folgenden Codezeilen dienen dazu, die Frequenzdarstellung des eingelesenen Signals und die dazugehörigen Frequenzen zu ermitteln. Das Resultat sieht man in der folgenden Grafik.



Mithilfe der Fouriertransformation konnten wir nun also die Amplitude des eingangs von uns eingelesen Signals bezüglich der Frequenz anstatt der Zeit wiedergeben. Dies ermöglichte es uns, die für den Ton grundlegenden Frequenzen zu eruieren.

Frequenzspitzen ermitteln

Als Hilfsmittel für die spätere Klangsynthese schrieben wir einen Algorithmus, der die wichtigsten n Frequenzen aus denen ein Klang besteht, also jene mit den größten Amplituden, ermitteln kann. Die Grundidee dieses Algorithmus ist die folgende: Zuerst sortieren wir den Vektor, in welchem alle Frequenzen gespeichert sind, und zwar abfallend bzgl. der Amplitudenhöhe. Dann suchen wir aus diesem sortierten Vektor die ersten n Frequenzen, zwischen jenen mindestens ein Abstand von 15 Frequenzwerten liegen muss. Der Grund für diese letzte Forderung ist der, dass ansonsten zwei Sinustöne mit ähnlichen Frequenzen für die spätere Rekonstruktion der Klänge verwendet werden könnten, was zu ungewollten Schwebungen führen könnte.

Nachfolgend ist ein Abschnitt des Codes zu sehen, welcher das Suchen der ersten n Frequenzen im sortierten Vektor darstellt.

```
counter = 1;
stopper = 2;

while (stopper <= length(allfreq) && counter < anzahl)

        mom_freq = allfreq(stopper);

   counter2 = 1;
   neu = 1;

       while (neu == 1 && counter2<=counter)

           if (abs(freq(counter2)-mom_freq)>15)
                counter2=counter2+1;
           else
                neu = 0;
           end
       end

       if(neu==1&&ampl(counter))
           counter = counter +1;
           freq(counter)=mom_freq;
           ampl(counter)=a_sorted(stopper);

       end
       stopper = stopper +1;
end
```

Klangsynthese

Additive Synthese

Bei der additiven Synthese werden mehrere Sinustöne mit verschiedenen Frequenzen und Amplituden aufaddiert, was zu komplexeren Klängen führt. Damit können beispielsweise Gitarrentöne annähernd realistisch reproduziert werden.

Beim Wiederherstellen des Klangs einer Gitarrensaite, der A-Saite zum Beispiel, erkennt man anhand der Ergebnisse der Klanganalyse, welche Frequenzen und Amplituden nötig sind um gute Annäherungen an den Originalton zu erreichen. Mit diesen Informationen werden Sinustöne erstellt, die jeweils als Vektor gespeichert werden, und zusammengefasst einen Ton ähnlich dem Originalton produzieren.

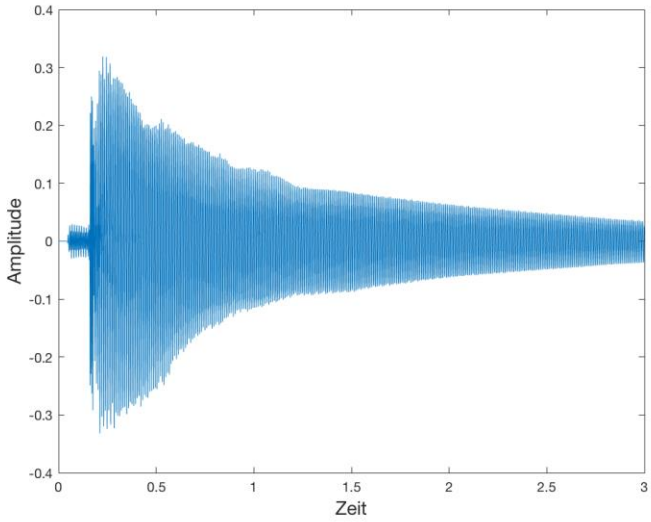
Diese Sinustöne sollen die Obertöne und den Grundton nachstellen. Der Grundton ist bei jedem Instrument derselbe, die Obertöne immer unterschiedlich und somit charakteristisch für jenes Instrument, mit dessen Hilfe sie kreiert wurden. Außerdem ist der Grundton generell meist der Ton mit der höchsten Amplitude, das heißt der dominanteste, jedoch stellten wir bei unseren Versuchen fest, dass dies nicht immer der Fall sein muss. Zum Beispiel, fiel dieses Phänomen

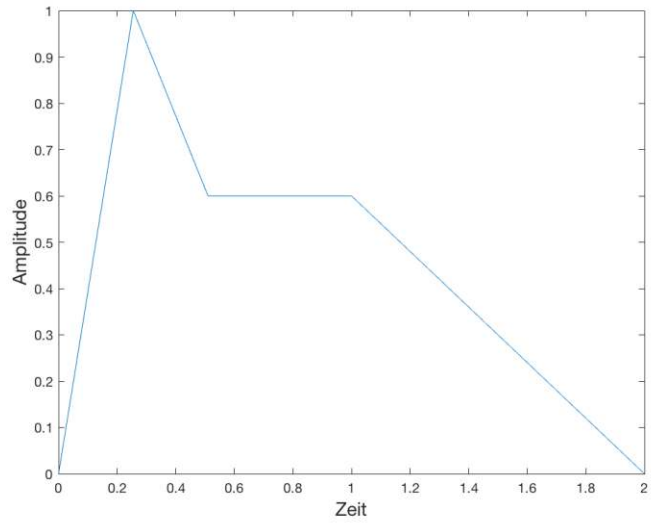
besonders auf, als wir den gleichen Ton von zwei verschiedenen Gitarren verglichen und sich herausstellte, dass die Amplitude des Grundtons bei einer der beiden sogar nur einem Fünftel der höchsten Amplitude entsprach.

Je mehr Informationen, das heißt Frequenzen und Amplituden, man miteinbezieht, desto genauer ist der entstehende Klang. Allerdings ist der Ton in der Zeit konstant, wenn man nur wenige Sinustöne zur Rekonstruktion verwendet, das heißt seine Lautstärke ändert sich nicht.

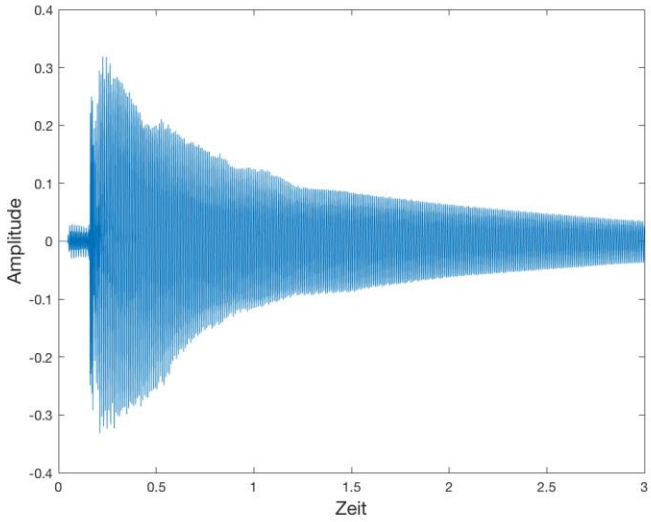
Um jedoch den wahren Klang einer A-Saite zu reproduzieren, haben wir eigene Funktionen geschrieben, die den Verlauf der Amplitude und dadurch der Lautstärke in der Zeit darstellen sollten. Solche Kurven werden in der Fachsprache auch Hüllkurven genannt. Diese sollten das Abklingen des Tons darstellen, da eine konstante Lautstärke eher künstlich wirkt.

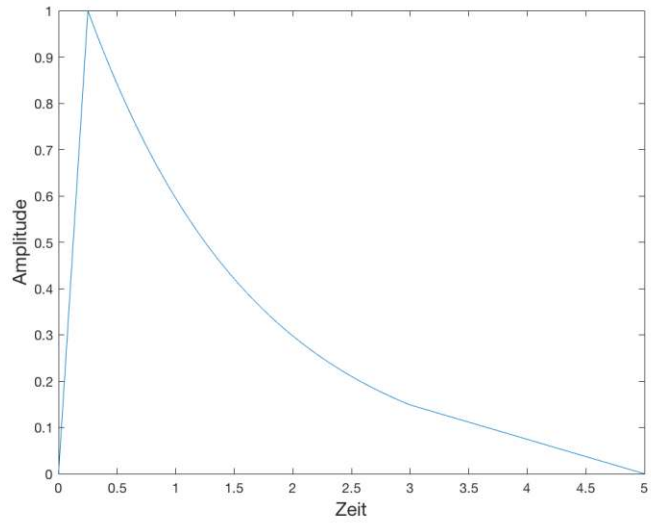
Zur Näherung an den Gitarrenton benutzen wir das Prinzip von ADSR, welches ein Akronym für die Worte „Attack“, „Decay“, „Sustain“, „Release“ und eine häufige Darstellungsform für Tonveränderungen in Bezug auf die Zeit bildet. „Attack“ ist die Phase, in der die Amplitude von Null auf ihren Höchstwert ansteigt. Bei der Gitarre ist dieser Verlauf einer linearen Funktion ähnlich, weshalb wir auch eine solche wählen, um diese Zeitperiode zu repräsentieren. In der „Decay“ Phase fällt die Amplitude zum ersten Mal bis sie das „Sustain level“ erreicht und da dieser Abfall in der Theorie wieder einer linearen Funktion ähnelt, wählten wir in unserem ersten Annäherungsversuch wieder diese Darstellungsform. Danach, während der „Sustain“ Phase, sollte die Amplitude konstant auf dem „Sustain level“ bleiben, bis sie schließlich im Laufe der „Release“ Phase endgültig wieder auf null sinkt. Für beide Prozesse wählten wir anfangs eine lineare Funktion.





Bei unserer zweiten Herangehensweise konzentrierten wir uns eher auf die Form des Originaltons als auf die typische ADSR-Darstellungsweise. Um den Gitarrenton korrekter zu imitieren, verwendeten wir zwar noch immer eine lineare Funktion zur Beschreibung der „Attack time“ und „Release time“, aber ersetzten jene, welche die „Decay time“ und die „Sustain time“ zeigten, durch eine Exponentialfunktion, da der originale Ton eher mit dieser Abfolge übereinstimmt und somit einen ADR-Verlauf bildet.





FM-Synthese

Trotz unserer Anstrengung konnten wir mit der additiven Synthese andere Töne, wie beispielsweise den einer Glocke, nicht genau nachbilden. Um diese Klänge besser modellieren zu können, versuchten wir es mit einer anderen Variante der Klangsynthese, der sogenannten FM-Synthese.

Als Grundlage der FM-Synthese diente folgender Ansatz für das Klangsignal:

$$y(t) = a\ sin\left(2\pi\left(f_0 t + I\ sin(2\pi\ f_m\ t)\right)\right)$$

a… Amplitude

f_0… Grundfrequenz

f_m… Modulationsfrequenz

t… Zeit

I… Modulationsindex

Statt verschiedene Sinustöne aufzuaddieren, wird bei dieser Methode also einfach das Argument der Sinusfunktion verändert. Mittels Fourierreihenentwicklung kann man jedoch den folgenden Zusammenhang zeigen:

$$y(t) = a \sum_{n=-\infty}^{\infty} J_n(\mathrm{I})\ sin(2\pi(f_0 + n\ f_m)\ t)$$

$J_n(I)$… n-te Besselfunktion an der Stelle I

Aus diesem wird klar, dass sich auch hier der entstehende Klang aus Sinustönen zusammensetzt, deren Frequenz durch $|f_0 + n\ f_m|$ und deren Amplitude durch $J_n(I)$ gegeben ist.

Der Vorteil bei der FM-Synthese ist, dass mit einem geschlossenen Ausdruck und damit wenig Rechenaufwand obertonreiche Klänge erzeugt werden können. Allerdings ist die genaue Wahl der Parameter I und f_m nicht so einfach, speziell wenn man an der Nachahmung eines bestimmten Klanges interessiert ist.

Tonhöhenerkennung

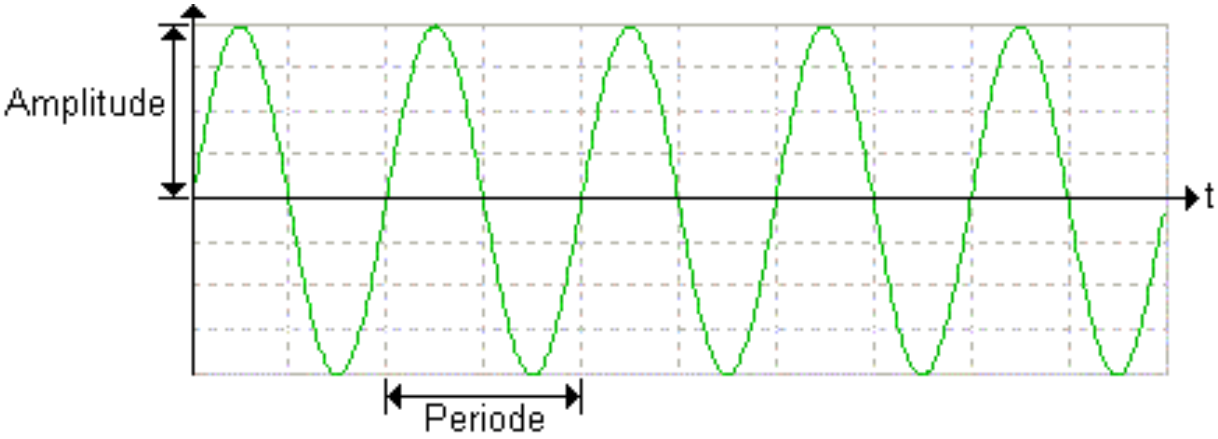
Nach den Erfahrungen und Erfolgen, die wir bei der Klangsynthese gemacht hatten, beschäftigten wir uns mit der Tonhöhenerkennung. Ziel war es, den Grundton eines gegebenen Klangs zu finden. Hierzu entwickelten wir zwei Methoden. Die erste basiert auf dem Finden von Nullstellen einer Funktion und ist verhältnismäßig einfach. Der Nachteil dieser Methode ist, dass sie in der von uns implementierten Weise im Grunde nur für einfache Sinustöne richtig funktioniert. Deshalb entwickelten wir eine zweite Methode, die auf der Frequenzdarstellung des Klanges und

insbesondere auf dem von uns vorher bereits implementierten Algorithmus zum Finden der Spitzen in dieser beruht. Nachfolgend werden die Algorithmen beschrieben.

Erster Algorithmus: Nullstellen ermitteln

Die Idee dahinter war folgende:

Die Frequenz f einer Sinusschwingung ist gegeben als Kehrwert der Periodendauer T. Das folgende Bild stellt eine Sinusschwingung dar.



Wie man sieht, schneidet diese nach jeder halben Periode die X-Achse. Wir schreiben zur Ermittlung einer Periode, also der Abstände jeder zweiten Nullstelle, eine Funktion, welche uns jede zweite Nullstelle ermittelt. Mit diesem Wissen können wir anhand der Forme f$=\frac{1}{T}$ die Frequenz berechnen.

Da unser Code bei hochfrequenten Sinusschwingungen die Frequenz nur sehr unzureichend ermittelte, ergänzten wir den anfänglichen Code noch. Anstatt nur zwei Nullstellen und damit nur eine mögliche Periodenlänge zu berechnen, berechneten wir mehrere, z.B. tausend. Unsere Annahme war, dass falls die bestimmten Ausprägungen dieser Periodenlänge um den echten Wert schwanken, wir durch Bildung des arithmetischen Mittels die Periode und damit die Frequenz genauer bestimmen können. In der Tat erhielten wir mit dieser Änderung nun auch für hochfrequente Schwingungen einen ausreichend genauen Wert.

Im Folgenden ist wieder eine mögliche Implementierung dieses Algorithmus gegeben.

```
function [frequenz] = sinus_nullstellen(signal_vektor,Fs);
nullstellen=1000;
zeiger = 1;
counter = 0;
nullstellen_Periode = zeros(1,100);
dif = 0;
freq = 0;
lambda=0;
frequenz=0;
all_freq=zeros(1,nullstellen);
x=0;

    while(counter<nullstellen&&counter<length(signal_vektor))

        akt_f = signal_vektor(zeiger);
        vgl_f = signal_vektor(zeiger+1);

        if(akt_f>0&&vgl_f<0)
            counter=counter+1;
            zeit=zeros(1,2);
            b = akt_f;
            zeit(1)=zeiger/Fs;
            zeit(2)=(zeiger+1)/Fs;

            %k = (y(x)-y(x0)/(x-x0))

            k = (vgl_f-akt_f)/(zeit(2)-zeit(1));
            x = (k*zeit(1)-akt_f)/k;

            nullstellen_Periode(counter)=x;

        end

        zeiger = zeiger+1;
    end

   run=1;
   while(run<nullstellen)
      all_freq(run)=1/(nullstellen_Periode(run+1)-nullstellen_Periode(run));
      frequenz=all_freq(run)+frequenz;
      run = run+1;
   end

   frequenz=frequenz/(nullstellen-1);

end
```

Zweiter Algorithmus: Bestimmen der Grundfrequenz aus der Frequenzdarstellung

Nach einigen Untersuchungen verschiedener Klänge mit Hilfe des ersten Algorithmus haben wir bemerkt, dass das Programm bei einem Ton, der selbst kein reiner Sinuston ist, sehr ungenau sein kann. Aus diesem Grund haben wir einen weiteren Algorithmus geschrieben, der nicht auf dem Finden der Nullstellen eines Signals beruht, sondern dessen Frequenzdarstellung verwendet. Die Idee ist, dass man den Grundton bestimmt, indem man die tiefste Frequenz findet, bei der es eine

relevante, das heißt ausreichend hohe, Spitze in dieser Darstellung gibt. Als Grundlage dafür wurde das Programm verwendet, das wir zur Ermittlung der Spitzen in der Frequenzdarstellung eines Signals geschrieben hatten. Das einzige, was daran verändert wurde, ist, dass eine Mindestfrequenz von 50 Hz festgelegt wurde. Das liegt daran, dass der tiefste Grundton, den beispielsweise eine Gitarre in Standardstimmung erzeugen kann, bei ungefähr 80 Hz (tiefe E-Saite) liegt. Alle Frequenzen, die in einem Gitarrenklang vorkommen können und tiefer sind, lassen sich durch ein gewisses Rauschen erklären, das mit dem Spielen auf einer Gitarre einhergeht, und können somit vernachlässigt werden. Um einen Spielraum für eine tiefere Stimmung oder auch Verstimmungen zu haben, haben wir die unterste Grenze nicht bei 80 Hz, sondern bei 50 Hz festgelegt. Zusammengefasst sucht dieser Algorithmus also eine Frequenz, die größer als 50 Hz ist und bei der sich die erste Spitze in der Frequenzdarstellung befindet.

Erweiterung des Algorithmus:

Die Ergebnisse waren deutlich genauer als die, die wir mit dem ersten Algorithmus erhalten hatten.

Doch uns ist aufgefallen, dass auch dieser Algorithmus ungenaue Ergebnisse liefert, wenn er auf ein kurzes Signal angewendet wird. Dies hat damit zu tun, dass die Frequenzauflösung Δf von der Länge N des Signals und der Abtastrate Fs abhängig ist. Es gilt nämlich der Zusammenhang $\Delta f = \frac{Fs}{N}$. Bei einem kurzen Signal erreicht man mit einer hohen Abtastrate wie 44100 Hz nur eine schlechte Frequenzauflösung, weshalb es schwierig ist, den Grundton zu bestimmen.

Die Erweiterung des Algorithmus besteht nun darin, das Signal in der Nähe der gefundenen Grundfrequenz quadratisch zu interpolieren. Das bedeutet, man nimmt diese Frequenz, deren Nachbarfrequenzen und dazu die jeweiligen Amplitudenwerte aus der Frequenzdarstellung des Signals und erstellt eine quadratische Funktion, auf deren Graph diese Punkte liegen. Das Maximum dieser quadratischen Funktion liegt in der Regel näher an der Grundfrequenz als der Wert, der mit dem ursprünglichen Algorithmus gefunden wurde.

Ergebnisse

Klavier

Aufgrund unseres musikalischen Themas verbrachten wir die letzten Tage in dem einzigen Seminarraum mit einem Klavier. Dies erwies sich jedoch nicht so vorteilhaft wie vermutet, da man jenes Klavier seit Jahren nicht gestimmt hatte und somit keine der Tasten den richtigen Ton auslöste. Aus diesem Grund beschlossen wir anhand unserer Ergebnisse in Sachen Klanganalyse zu prüfen, wie verstimmt die Klaviersaiten wirklich waren.

Um dies zu erreichen, nahmen wir mehrere der Klaviertöne auf, speisten sie in Matlab ein und bestimmten ihre Grundfrequenz. Die nachfolgenden QR-Codes führen zu Audio-Files, die die aufgenommenen Klaviertöne sowie die Sinustöne enthalten, deren Frequenz die von uns bestimmte Grundfrequenz ist.

Verstimmtes A

Die Ergebnisse zeigen, dass die von uns implementierte Tonhöhenerkennung gut funktioniert. Ebenso zeigen sie, wie verstimmt das Klavier ist. Der erste Klavierton, der eigentlich ein A darstellen sollte, also eine Frequenz von 110 Hz haben sollte, hat eigentlich eine Frequenz von 102,24 Hz.

Verstimmtes G

Ebenso hat der zweite Klavierton, der eigentlich ein g sein sollte, eine Frequenz von 191,11 Hz statt einer Frequenz von 196 Hz.

Klangsynthese: Gitarre

Mithilfe der additiven Synthese und der entsprechenden Hüllkurve konnten wir letztendlich den Klang einer Gitarrensaite unter Verwendung von 20 Sinustönen passabel imitieren, wie man unter folgendem Link hören kann.

Gitarrentöne 20 Frequenzen

Um zu sehen, ob mehr Sinustöne auch wirklich zu einem realistischeren Klang führen, haben wir als Test Gitarrentöne mit 20.000 Sinustönen zu unterschiedlichen, nicht separierten Frequenzen erstellt. Selbst ohne zusätzliches Hinzufügen einer Hüllkurve sind die entstandenen Klänge ziemlich realitätsnah:

Gitarrentöne

Klangsynthese: Glocke

Mithilfe der FM-Synthese haben wir es geschafft Klänge zu erzeugen, die an Glockenklänge erinnern. Ein Beispiel dafür findet man unter folgendem Link.

Glocke

Als Beweis für unsere Leistungen, benutzten wir die bereits reproduzierten Gitarrensaiten, um das Intro des Liedes „Hey, Hey, Wickie“, das Titellied der bekannten Kinderserie „Wickie und die starken Männer“, mit ebendiesen Gitarrensaiten nach. Um jene Töne zu erhalten, die nicht auf den leeren Gitarrensaiten liegen, transponierten wir diese. So erhielten wir alle fehlenden Töne und letztendlich auch die vollständige Melodie, wobei die Tonaufnahme kaum unterscheidbar von der einer echten Gitarre ist.

Wiki Titelmelodie

WOCHE DER MODELLIERUNG

Projekt: KONTROLLTHEORIE

Titel: Modellierung und Tuning eines Quadcopters

Namen der Teilnehmer

Konrad Eisenberger

Johannes Binninger

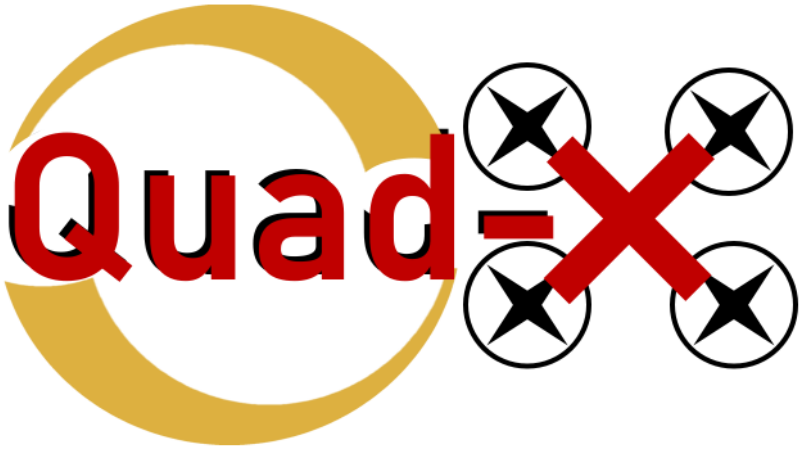
Jakob Pertl

Anja Bierbauer

Simon Nitsch

Marlene Anzenberger

Betreuer: Philip Trautmann



Inhaltsverzeichnis

1 Einleitung

2 Problemstellung

Beim Fliegen eines Quadcopters können einige Fehler auftreten, die die Lage und das Flugverhalten des Quadcopters beeinflussen. Besonders als Laie hat man aber sowieso schon genug damit zu tun, einen Quadcopter in einer halbwegs stabilen Lage zu halten, sodass solche zusätzlichen Fehler viel zu überfordernd sind.
Dafür wird bei Quadcoptern ein sogenannter P.I.D. (Proportional Integral Derivative)-Kontrollmechanismus verwendet. Dieser Mechanismus kontrolliert drei Werte: Die Abweichung von der Höhe und der Ausrichtung des Quadcopters (Proportional), das Integral dieser Abweichungen, um mögliche Systemfehler zu entdecken und auszugleichen (Integral) und die Ableitung der Abweichungen, um den Trend von den Abweichungen zu erkennen und die Ausgleichsbewegungen dementsprechend anzupassen.

3 Grundlagen

3.1 Rechnen mit Matrizen

Matrizen sind mit Vektoren oder anderen Matrizen multiplizierbar, vorausgesetzt die Dimensionen stimmen. Um eine Matrix mit einem Vektor zu multiplizieren nimmt man das Skalarprodukt der Zeilen der Matrix und dem Vektor. Das sieht dann so aus:

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_1x + a_2y + a_3z \\ b_1x + b_2y + b_3z \\ c_1x + c_2y + c_3z \end{pmatrix}$$

Bei der Multiplikation einer Matrix und eines Vektors kommt also ein Vektor als Produkt heraus. Anders verhält es sich bei der Multiplikation einer Matrix mit einer Matrix.
Um zwei Matrizen zu multiplizieren, nimmt man das Skalarprodukt von den Zeilen der ersten Matrix und den Spalten der zweiten Matrix. Jede der Zellen des Produktes ist das Skalarprodukt der Zeile und der Spalte. Diesen Vorgang kann man sich folgendermaßen vorstellen:

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} * \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{pmatrix} = \begin{pmatrix} a_1x_1 + a_2y_1 + a_3z_1 & a_1x_2 + a_2y_2 + a_3z_2 & a_1x_3 + a_2y_3 + a_3z_3 \\ b_1x_1 + b_2y_1 + b_3z_1 & b_1x_2 + b_2y_2 + b_3z_2 & b_1x_3 + b_2y_3 + b_3z_3 \\ c_1x_1 + c_2y_1 + c_3z_1 & c_1x_2 + c_2y_2 + c_3z_2 & c_1x_3 + c_2y_3 + c_3z_3 \end{pmatrix}$$

Wichtig ist, dass bei der Multiplikation von Matrizen das Kommutativgesetz nicht gilt. Es ist also nicht egal, ob eine Matrix bei einer Multiplikation an erster oder zweiter Stlle steht. Das heißt:

$$A * B \neq B * A$$

Multipliziert man eine Matrix mit ihrer inversen Matrix, ist das Produkt die Einheitsmatrix. Die Einheitsmatrix ist eine quadratische Matrix, die, wenn man sie mit einer Matrix oder einem Vektor multipliziert, genau diese Matrix oder diesen Vektor als Produkt ergibt.
Einheitsmatrizen haben auf der Hauptdiagonale den Wert 1 und an den restlichen Stellen den Wert 0.

$$A * A^{-1} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} * B = B$$

Spiegelt man eine Matrix entlang der Hauptdiagonale, ist diese Matrix transponiert. Diese Eigenschaft wird auch mit einem hochgestellten T signalisiert.

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}^T = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & b_3 \\ a_3 & b_3 & c_3 \end{pmatrix}$$

Entspricht die Inverse einer Matrix der Transponierten dieser Matrix, ist die Matrix orthogonal.

4 Modellierung des Quadcopters

4.1 Drehmatrizen

Bei einem Quadcopter braucht man zwei Koordinatensysteme: Das Inertialsystem, das immer gleich bleibt und das körperfeste Koordinatensystem, das den Nullpunkt in der Mitte des Quadcopters hat und zusammen mit dem Quadcopter rotiert. Dieses körperfeste Koordinatensystem ist nicht immer gleich ausgerichtet wie das Inertialsystem, da sich die Lage des Qudcopters stetig verändert. Um diese Rotationen auf das körperfeste Koordinatensystem zu übertragen, sind Drehmatrizen vonnöten. Eine Drehmatrix multipliziert mit einem Vektor ergibt nämlich den rotierten Vektor.
Eine Drehmatrix enthält alle möglichen Operationen, die bei der Rotation angewendet werden. In der ersten Spalte steht der Vektor, der entsteht, wenn man den Vektor $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ rotiert, in der zweiten Spalte der Vektor, der entsteht, wenn man den Vektor $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ und in der dritten Spalte der Vektor von dem rotierten Vektor $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.
Als Beispiel wird die Drehmatrix für eine Rotation um die z-Achse mit dem Winkel γ hergeleitet. Rotiert man die x-Achse um die z-Achse mit dem Winkel γ, erhält man also eine neue x-Achse. Der 1. Eintrag des Vektors ist $\cos\gamma$ und der 2. Eintrag ist $\sin\gamma$. Da die Achse horizontal rotiert wurde, hat die Drehung keine Auswirkung auf die z-Achse. Das heißt, die erste Spalte der Matrix sieht so aus:

$$\begin{pmatrix} \cos\gamma \\ \sin\gamma \\ 0 \end{pmatrix}$$

Als nächstes rotieren wir die y-Achse. Der 1. Eintrag des rotierten Vektors wird mit einem zusätzlichen Minuszeichen versehen, da die Rotation in die mathematisch positive Richtung, also gegen den Uhrzeigersinn geschieht. Der 1. Eintrag des rotierten Vektors beträgt also $-\sin\gamma$ und der 2. Eintrag ist $\cos\gamma$. Da um die z-Achse rotiert wurde, ist der letzte Wert auch in der zweiten Spalte 0. Die zweite Spalte ist also:

$$\begin{pmatrix} -\sin\gamma \\ \cos\gamma \\ 0 \end{pmatrix}$$

Die letzte Spalte der Drehmatrix zeigt die neuen Werte der rotierten z-Achse. Da wir aber um die z-Achse rotieren, verändert sich diese nicht. Das heißt, die letzte Spalte ist:

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Jetzt haben wir alle Spalten der Drehmatrix und können sie somit erstellen.

$$\begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Multipliziert man einen Vektor mit dieser Drehmatrix, erhält man als Ergebnis den gleichen Vektor, aber mit dem Winkel γ um die z-Achse gedreht. Hier ein Beispiel.

$$\begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} \cos\gamma - 2\sin\gamma \\ \sin\gamma + 2\cos\gamma \\ 3 \end{pmatrix}$$

Um die Rotation in allen drei Dimensionen darzustellen, verwenden wir die Euler-Winkel und die zyx-Konvention. Das heißt, wir drehen zuerst um die z-Achse mit dem Winkel ψ, dann um die neue y-Achse mit dem Winkel θ und schlussendlich um die zweimal rotierte x-Achse mit dem Winkel ϕ. Die Rotationsmatrix errechnet man sich, indem man sich die Drehmatrizen für die drei Achsen multipliziert.

$$R^{-1} = R_x * R_y * R_z =$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} * \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} * \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\phi * \cos\theta * \cos\psi - \sin\phi * \sin\psi & -\cos\phi * \cos\theta * \sin\psi - \sin\phi * \cos\psi & \cos\phi * \sin\theta \\ \sin\phi * \cos\theta * \cos\psi + \cos\phi * \sin\psi & -\sin\phi * \cos\theta * \sin\psi + \cos\phi * \cos\psi & \sin\phi * \sin\theta \\ -\sin\theta * \cos\psi & \sin\theta * \sin\psi & \cos\theta \end{pmatrix}$$

Setzt man in diese Matrix die Euler-Winkel des körperfesten Koordinatensystems ein, erhält man schließlich die Drehmatrix für die Transformation vom Inertialsystem zum körperfesten System des Quadcopters.
Will man einen Vektor im körperfesten System darstellen, dreht man das Koordinatensystem anstatt dem Vektor. Das heißt, die drei Drehmatrizen sind invertiert. Da diese Matrizen orthogonal sind, reicht es, sie zu transponieren. Die Rotationsmatrix erhält man also durch folgende Multiplikation.

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} * \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} * \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4.2 Winkelgeschwindigkeit

Quadcopter messen nicht direkt die Ausrichtung im Inertialsystem, sondern nur die Winkelgeschwindigkeiten. Deshalb muss man diese Winkelgeschwindigkeit erst umrechnen, bevor man weitere Berechnungen anstellen kann.
Die Winkelgeschwindigkeit ist ein Vektor, der sich aus den Bewegungen entlang jeder der drei Achsen

des körperfesten Koordinatsystems zusammensetzt. Diese wiederum, sind die Ableitungen der Eulerwinkel multipliziert mit einer Rotationsmatrix. Die Eulerwinkel werden zusammen als η angeben. Die Winkelgeschwindigkeit wird als ν angeben und besteht aus der Geschwindigkeit in x-Richtung (p), der Geschwindigkeit in die y-Richtung (q) und der Geschwindigkeit in die z-Richtung (r).

$$\eta = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \quad \eta' = \begin{pmatrix} \phi' \\ \theta' \\ \psi' \end{pmatrix} \quad \nu = \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

Um η' zu bestimmen, multipliziert man ν mit der Rotationsmatrix W_η^{-1}. Diese Rotationsmatrix sieht aber anders aus als die oben beschriebene.

$$W_\eta^{-1} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{pmatrix}$$

$$\eta' = W_\eta^{-1}\nu$$

4.3 Zustand des Quadcopters

Der Zustand x wird beschrieben durch den Ort, die Geschwindigkeit, den Eulerwinkeln und die Winkelgeschwindigkeit.

$$x = \begin{pmatrix} \xi^T & v^T & \eta^T & \nu^T \end{pmatrix}^T \in \mathbb{R}^{12}$$

Jeder dieser vier Werte hat drei Dimensionen, damit ist der Zustand des Qudcopters zwölf-dimensional. Zusammen mit der Kontrolle $u = \begin{pmatrix} \omega_1 & \omega_2 & \omega_3 & \omega_4 \end{pmatrix} \in \mathbb{R}^4$, wobei ω_i, für die Winkelgeschwindigkeit des Rotors i steht, kann man das Kontrollsystem für den Quadcopter herleiten.

$$x' = f(x,u) = \begin{pmatrix} v \\ (G + RT)/m \\ W_\eta^{-1}\nu \\ I^{-1}(-\nu \times I * \nu - \Gamma + \tau) \end{pmatrix}$$

Dabei steht T für die Schubkraft,

$$\begin{pmatrix} 0 \\ 0 \\ b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \end{pmatrix}$$

I für die Trägheitsmatrix,

$$\begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix}$$

G für die Erdanziehungskraft,

$$\begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix}$$

d für den Abstand zwischen Rotor- und Masseschwerpunkt, Γ für das Kreiselmoment und τ für das externe Drehmoment:

$$\begin{pmatrix} 0 & -db & 0 & db \\ -db & 0 & db & 0 \\ -k & k & -k & k \end{pmatrix} * \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \begin{pmatrix} -db\omega_2^2 + db\omega_4^2 \\ -db\omega_1^2 + db\omega_3^2 \\ -k\omega_1^2 + k\omega_2^2 - k\omega_3^2 + k\omega_4^2 \end{pmatrix}$$

k und b sind Konstanten, die mit dem Strömungswiderstand und dem Auftrieb zu tun haben.
Die externen Drehmomente initiieren die drei Drehungen, die ein Quadcopter machen kann: Roll (Drehung um die x-Achse), Pitch (Drehung um die y-Achse) und Yaw (Drehung um die z-Achse). Verringert man also die Rotation vom zweiten Rotor und erhöht die Rotation vom vierten Rotor, führt der Quadcopter eine mathematisch positive Drehung um die x-Achse aus, also gegen den Uhrzeigersinn. Umgekehrt macht der Quadcopter eine mathematisch negative Drehung.
Das Gleiche gilt für den Pitch, wie man in der zweiten Zeile des Vektors sieht, nur für den ersten und den dritten Rotor.
Den Yaw muss man aber anders ansteuern; Damit sich der Quadcopter nicht einfach so dreht, drehen sich zwei Rotoren im Uhrzeigersinn und zwei Rotoren gegen den Uhrzeigersinn. Die Rotoren, die sich in die gleiche Richtung drehen, stehen sich diagonal gegenüber, das heißt, die beiden benachbarten Rotoren eines Rotors drehen sich in die entgegengesetze Richtung des Rotors. Somit fallen die Drehmomente normalerweise weg. Um eine Rotation um die z-Achse auszuführen, drehen sich zwei gegenübergesetzte Rotoren schneller, während sich die anderen beiden langsamer drehen.
Der Vektor x' besteht aus den vier abgeleiteten Werten von x. In der ersten Komponente steht die Ableitung vom Ort, also die Geschwindigkeit und in der zweiten Komponente wird das 2. Newtonsche Gesetz $f = ma$ angewandt, damit werden die Kräfte, nämlich die Erdanziehungskraft und der Schub, der zunächst mittels der Rotationsmatrix vom körperfesten System ins Inertialsystem umgewandelt werden muss, in die Beschleunigung umgewandelt. In der dritten Komponente wird die Winkelgeschwindigkeit wie in der obigen Rechnung in die Ableitung der Eulerwinkel umgewandelt und in der vierten Komponente wird die Ableitung der Winkelgeschwindigkeit hergeleitet.
Das Drehmoment, das durch die Euler-Kraft entsteht, ergibt sich aus der abgeleiteten Winkelgeschwindigkeit multipliziert mit der Trägheitsmatrix, also $I\nu'$. Für ein besseres Verständnis schauen wir uns die Eulergleichung an.

$$I\nu' = -\nu \times I\nu - \Gamma + \tau$$

Der Term $-\nu \times I\nu$ beschreibt das Drehmoment, das durch die Zentripetalkraft entsteht. Die Konstante Γ steht, wie schon oben erwähnt, für das Kreiselmoment und τ steht für das oben besprochene Drehmoment.
Multipliziert man die Gleichung mit der invertierten Trägheitsmatrix I^{-1}, erhält man die letzte Komponente des Kontrollsystems.

5 Numerische Simulation vom Quadcopter

Wir wollten uns eine Einsicht in die Funktionsweise eines Quadcopters verschaffen indem wir ein möglichst wahrheitsgetreues Programm erstellten. In diesem haben wir den Großteil der auf den Quadcopter wirkenden Faktoren berücksichtigt, sodass wir eine Art Simulator bekamen. Da wir über vollständige Bewegungsgleichungen verfügen, die die Dynamik des Systems beschreiben, konnten wir eine Simulation erstellen, in der die Ergebnisse verschiedener Eingaben und Zustände getestet und angezeigt werden. Obwohl es auch fortgeschrittenere Methoden gibt, konnten wir relativ schnell einen Simulator schreiben, der die Euler-Methode zur Lösung von Differentialgleichungen verwendet. Mit dem Programm hatten wir die Möglichkeit, einige Grafiken zu erstellen.

5.1 Main Funktion

Unsere Main Function auf Matlab beinhaltet alle nötigen Parameter zur numerischen Simulation vom Quadcopter. Diese Parameter umfassen etwa 40 Werte, wie Gravitationskonstante, Drag Force (Luftwiderstand) oder Trägheitsmatrix des Quadcopters um die verschiedenen Achsen. Ebenfalls wichtige, in der Main Function angegebene Parameter sind die Daten des Quadcopters, wie die Masse oder der Abstand der Rotoren vom Masseschwerpunkt. Außerdem sind die Parameter für die Control Function definiert. Hier befinden sich die Bedingungen vom Desired Zustand, welcher für uns der Zustand ist, in dem der Quadcopter sich stabilisieren soll. Wir können von der Main Funktion aus den Anfangszustand regeln, indem wir die Werte in der folgenden Matrix verändern.

$$X(:,1) = zeros(12,1)$$

$$X(1:3,1) = [0;0;0]$$

$$X(7:9,1) = [0;0;0]$$

Die Eulerwinkel und dessen Ableitungen sind in diesem Fall alle auf null gestellt. Wenn dies nicht der Fall ist, ist es unser Ziel, den Fehler auf null konvergieren zu lassen.

5.1.1 Anwendung des expliziten Euler Verfahren auf unser System

Das explizite Euler Verfahren ist eine numerische Lösungverfahren für das Anfangswert-Problem. Die berechneten Werte X_k sind Approximationen der tatsächlichen Werte der Lösung X. Wir zerlegen das Zeitinterval in t_k Punkte. Die Differenz zwischen zwei aufeinanderfolgenden Zeitpunkte nennen wir τ. Je kleiner die Schrittweite τ ausgelegt ist, desto genauer werden auch die approximierten Werte. Dazu muss man aber mehr und mehr rechnen.

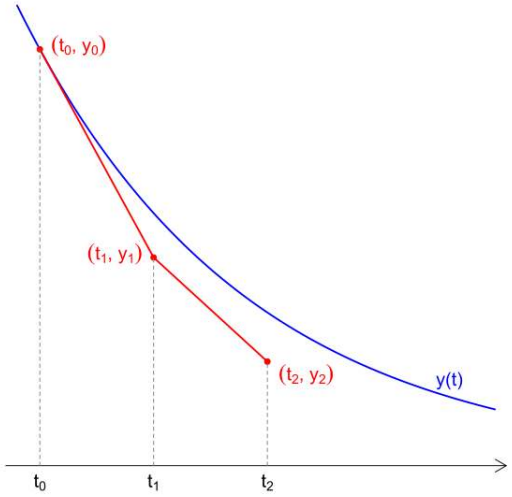


Abbildung 1: Annäherung

In der Main Funktion wird insbesondere diese Differentialgleichung gelöst:

$$\dot{X} = F(X, U(X)), X(0) = c$$

Dies tun wir mit dem expliziten Euler Verfahren. Das bedeutet, dass die folgende Berechnung durchgeführt werden muss:

$$X_k = \tau * F(X_{k-1}, U(X_{k-1})) + X_{k-1}$$

Das heißt, dass die Lösung am Zeitpunkt t_k aus der Ableitung

$$F(X_{k-1}, U(X_{k-1}))$$

und aus dem Zustand am letzten Zeitpunkt berechnet wird. Um diese Ableitung zu berechnen, müssen wir zunächst die Kontrolle berechnen. Da wir eine Feedback Kontrolle verwenden, welche vom Zustand am letzten Zeitpunkt X_{k-1} abhängt, müssen wir diesen Zustand an die Control weitergeben. Dies ist der beobachtete Zustand. In der Control Funktion wird die Funktion U berechnet, dies wird später erklärt.
Diese Kontrollfunktion wird zusammen mit dem Zustand am letzten Zeitpunkt an die RHS Funktion weitergegeben, siehe Abschnitt 5.4. Diese berechnet die Funktion $F(X_{k-1}, U(X_{k-1}))$ (die Ableitung von X zum Zeitpunkt t_k).
Da der Zustand X_{k-1} benutzt wird, um die Control Funktion zu berechnen, liegt eine Feedback Loop vor.

5.2 Grundgerüst der Simulation

Um den Quadcopter erfolgreich in Matlab zu simulieren galt es eine große Schleife bestehend aus drei verschiedenen Funktionen zu implementieren. Die erste Funktion (Main) beeinhaltet alle Parameter der Simulation und die zugrundeliegende Feedback Loop. In dieser Loop wird aus dem Zustand des letzten Zeitwertes und den Ergebnissen der anderen zwei Funktion der neue Zustandsvektor berechnet. Die zweite Funktion (Control) berechnet aus dem Zustand dieses Zeitwertes die notwendigen Änderungen der Rotoren um den Quadcopter wieder in die gewünschte Lage zu bringen. Die dritte Funktion (Right Hand Side) errechnet aus den Informationen der Control Funktion die physikalischen Implikationen und die daraus resultierenden Änderungen am Zustandsvektor des Quadcopters, und gibt diese an die Main Funktion weiter. Dadurch hat die main Funktion wieder einen neuen Input und kann den

aktuellen Zustandsvektor berechnen. Im ersten Schritt der Loop müssen wir der Control Funktion unsere Anfangswerte geben, da sie noch kein Feedback hat, und müssen unsere physikalischen Parameter setzen. Dadurch können wir die Bewegung, Geschwindigkeit und weitere Attribute des Quadcopters simulieren und voraussagen.

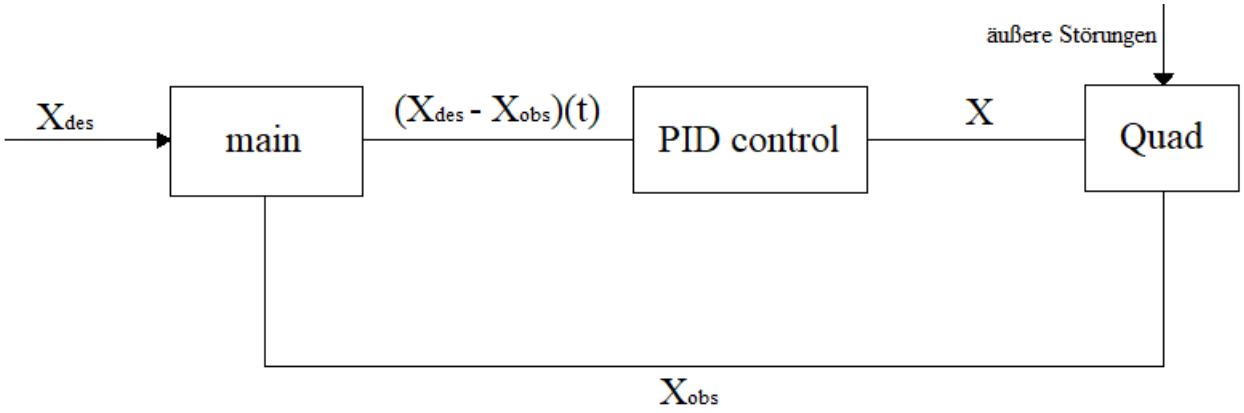


Abbildung 2: Grundgerüst der Simulation

5.3 Control

Das Ziel der Control Funktion ist den Fehler zwischen dem gewünschten Zustand und dem observierten Zustand zu minimieren: $E = X_{des} - X_{obs} \rightarrow 0$. Dafür verwenden wir eine sogenannte PD Kontrolle. Der Desired Zustand ist in unserem Fall den Quadcopter normal in der Luft stehen zu lassen. Das bedeutet das die Winkelgeschwindigkeiten und die Euler-Winkel gleich Null sind. Um die aktuellen Winkel und deren Geschwindigkeiten zu messen verwenden wir einen Gyrometer und ein Accelerometer. Ein Gyrometer misst die Ableitung der Euler-Winkel beziehungsweise die Winkelgeschwindigkeiten und ist standardmäßig in Quadcoptern eingebaut. Das Accelerometer misst die eigentlichen Euler-Winkel aufgrund der Gravitation. Die PD Kontrolle besteht aus zwei Komponenten. Einerseits dem Proportional Kontrollmechanismus, andererseits dem Derivative Kontrollmechanismus. Der Proportional Mechanismus ist direkt proportional zum Fehler: $(\tau_\phi, \tau_\theta, \tau_\psi) = P * ((\phi_{des}, \theta_{des}, \psi_{des}) - (\phi_{obs}, \theta_{obs}, \psi_{obs})) \rightarrow 0$. Der Derivative Mechanismus bezieht sich auf die Veränderung des Wertes: $(\tau_\phi, \tau_\theta, \tau_\psi) = D * ((\dot{\phi}_{des}, \dot{\theta}_{des}, \dot{\psi}_{des}) - (\dot{\phi}_{obs}, \dot{\theta}_{obs}, \dot{\psi}_{obs})) \rightarrow 0$. Durch geschicktes Setzen des Parameters P und D kann so der Fehler minimiert werden. Die Summe beider Terme geben an welches Drehmoment wir aufwenden müssen um den Quadcopter wieder zu stabilisieren. Der Derivative Mechanismus versucht den Fehler zu minimieren in dem er die Tangente des Errors gering hält. Außerdem hält er auch den Fehler in den Ableitungen selber niedrig. Ein Problem das bei dem PD Kontrollmechanismus auftreten kann, sind Überschwinger. Wenn der Fehler unter die Nullgrenze fällt versucht die Derivative-Kontrolle weiterhin die Tangente zu minimieren, welches zu Oszillationen um die Nullgrenze führen kann. Nachdem wir die Fehler berechnet haben mussen wir noch die notwendigen Änderungen an den Rotoren ausrechnen um den Fehler möglichst gering zu halten. Die

Rotorgeschwindigkeiten lassen sich durch folgenden Vektor ausdrücken:

$$\begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \begin{pmatrix} \frac{T}{4*b} - \frac{\tau_\theta}{2*b*d} - \frac{\tau_\psi}{4*k} \\ \frac{T}{4*b} - \frac{\tau_\phi}{2*b*d} + \frac{\tau_\psi}{4*k} \\ \frac{T}{4*b} + \frac{\tau_\theta}{2*b*d} - \frac{\tau_\psi}{4*k} \\ \frac{T}{4*b} + \frac{\tau_\phi}{2*b*d} + \frac{\tau_\psi}{4*k} \end{pmatrix}$$

Die Control Funktion schickt ihren Output an die Right Hand Side Funktion, damit die Right Hand Side Funktion die Ableitung des Zustands berechnen kann.

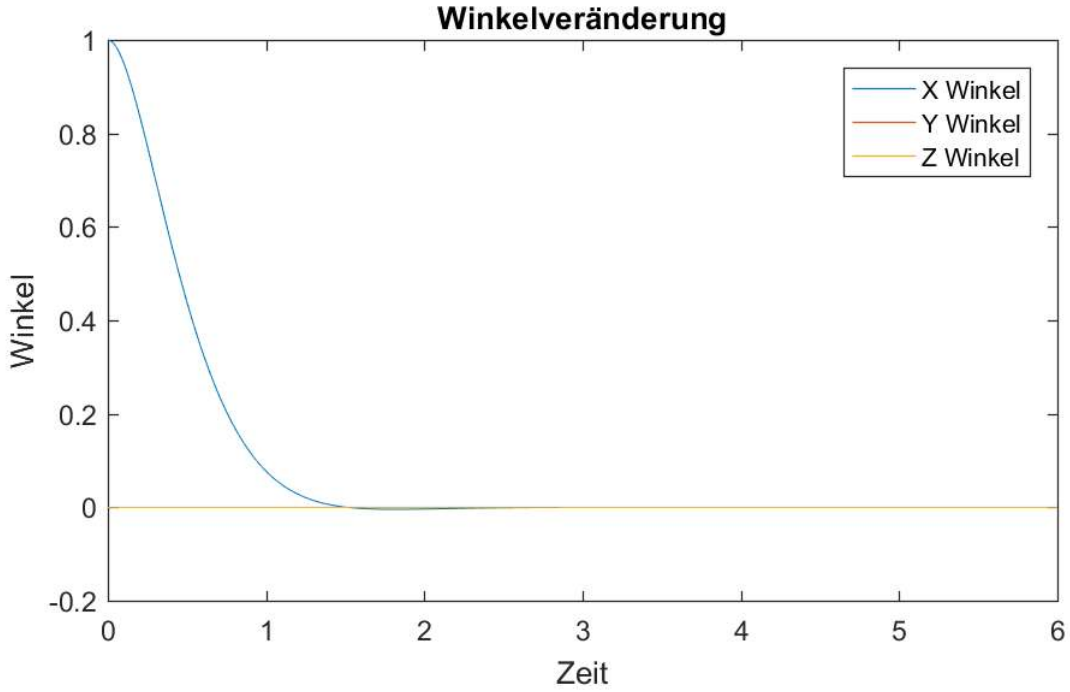


Abbildung 3: Ausgleich des X Winkels durch PD Kontrolle

5.4 Right Hand Side

Die RHS Funtion nimmt als Input den letzten Zustand und die berechnete Kontrolle und gibt den Vektor F aus. Dieser Vektor F beinhaltet 12 Komponenten aus welchen die main Funktion den neuen Zustandsvektor berechnen kann. Die ersten drei Komponenten sind einfach die Geschwindigkeiten des letzten Zustands: $F_{(1:3)} = v$. Die nächsten Drei geben die Beschleunigung an. In unserem Modell berücksichtigen wir die Gravitation und den Schub des Quadcopters als Beschleunigung: $F_{(4:6)} = \frac{1}{m} * (G + R^{-1} * T)$. Diese Formel ist im Prinzip eine Umformung der Gleichung auf die Beschleunigung: $F = m * a$. Die Kraft ergibt sich aus der Summe der Gravitationskraft und der Schubkraft. Die Schubkraft wird noch multipliziert mit der Rotationsmatrix da bei einer Schieflage des Quadcopters der Schubvektor im Intertialsystem nicht ausschließlich in die Z-Richtung zeigt. Die dritte Komponente rechnet die aktuellen Winkelgeschwindigkeiten vom Bodyframe in das Inertialsystem zurück: $F(7:9) = W_\eta^{-1} * \nu$. Die letzte Komponente ist die Euler-Gleichung: $F(10:12) = I^{-1} * (-\nu \times I * \nu - \Gamma + \tau)$. Die Euler- Gleichung berechnet uns die Winkelbeschleunigungen. Hat das Programm den Vektor fertig ausgewertet gibt die Funktion rhs ihn an die main Funktion weiter. Die main Funktion rechnet dann aus dem letzten Zustand und der Ableitung den neuen Zustand aus.

6 Exkurs: Simulation eines Tempomaten

6.1 Herleitung der Formel

Das Auto soll aus dem Stand in fünf Sekunden auf eine konstante Geschwindigkeit von 10 m/s beschleunigen. Hierfür verwenden wir die Differentialgleichung $mx'' + bx' = u$, wobei m die Masse des Autos und b die Reibungskraft ist. Als erstes legen wir fest, dass die erste Ableitung von x die Funktion v ist, da die erste Ableitung vom Weg die Geschwindigkeit ist. Als nächstes formulieren wir die sich daraus ergebende Formel $mv' + bv = u$ in Matrizen und Vektoren um.

$$\begin{pmatrix} x' \\ v' \end{pmatrix} = \begin{pmatrix} v \\ \frac{1}{m}(u - bv) \end{pmatrix} = \frac{1}{m}\begin{pmatrix} 0 \\ u \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & -\frac{b}{m} \end{pmatrix}\begin{pmatrix} x \\ v \end{pmatrix}$$

Da wir auch diese Gleichung noch nicht lösen können, verwenden wir den expliziten Euler um diese Differenetialgleichung zu diskretisieren und die Lösung mit dem Computer zu berechnen. Dazu diskretisieren wir das Intervall $[0; T]$ mit n Zeitpunkten. Weiterhin führen wir die diskreten Variablen x_k und v_k ein.

$$\frac{1}{\tau}\begin{pmatrix} x_k - x_{k-1} \\ v_k - v_{k-1} \end{pmatrix} = \frac{1}{m}\begin{pmatrix} 0 \\ u_{k-1} \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & -\frac{b}{m} \end{pmatrix}\begin{pmatrix} x_{k-1} \\ v_{k-1} \end{pmatrix}$$

Entsprechend dieser Definition ergibt sich folgende diskrete Version der Differentialgleichung:

$$\begin{pmatrix} x_k \\ v_k \end{pmatrix} = \frac{\tau}{m}\begin{pmatrix} 0 \\ u_{k-1} \end{pmatrix} + \begin{pmatrix} 1 & \tau \\ 0 & 1 - \tau\frac{b}{m} \end{pmatrix}\begin{pmatrix} x_{k-1} \\ v_{k-1} \end{pmatrix}$$

6.2 Eingabe in MATLAB

Die daraus entstandene Funktion *cart*, die wir in MATLAB verwendeten, beschrieb das von uns gewollte Ergebnis, nämlich, dass das Auto über längere Sicht auf die gewünschte Geschwindigkeit von $10m/s$ beschleunigt. Dazu verwenden wir eine konstante Kontrolle von 500 nm, damit wir nach ungefähr 10 Sekunden die gewünschte Geschwindigkeit von 10 m/s erreichen. Die 500 nm sind nötig, da sonst die Reibungskräfte nicht soweit überwunden werden könnten, um auf 10 m/s zu beschleunigen.

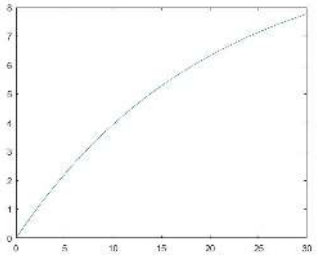


Abbildung 4: Geschwindigkeitsfunktion

Im nächsten Schritt wollen wir, dass der modellierte Tempomat nicht nur auf einer geraden Strecke funktioniert, sondern auch, wenn eine negative Kraft dagegen wirkt. Wir nehmen als Beispiel einen simulierten Berg. Unseren sogenannten Berg stellen wir mit der Kraft -200 nm dar. In der daraus entstandenen neuen Gleichung soll das Auto auch die neuen Kräfte ausgleichen und auf 10 m/s bleiben. Dazu erstellen wir eine zweite Funktion (berg) in MATLAB, in welcher wir mit einer if-Abfrage modellieren, dass in der Zeit von $0-10$ keine negative Kraft und ab 10 Sekunden eine dauerhafte Kraft

von -200 nm wirkt. Um diese Störung auszugleichen, wollen wir eine Feedback-Kontrolle in Form einer IP-Kontrolle verwenden, die folgende Form hat:

$$u(k-1) = p(10 - v_{k-1}) + I \sum_{i=1}^{k-1} (10 - v_i)\tau \tag{1}$$

Beim Integral wird die Fläche zwischen dem Graph und der x-Achse berechnet. Dadurch werden alle vergangenen Fehler aufsummiert.

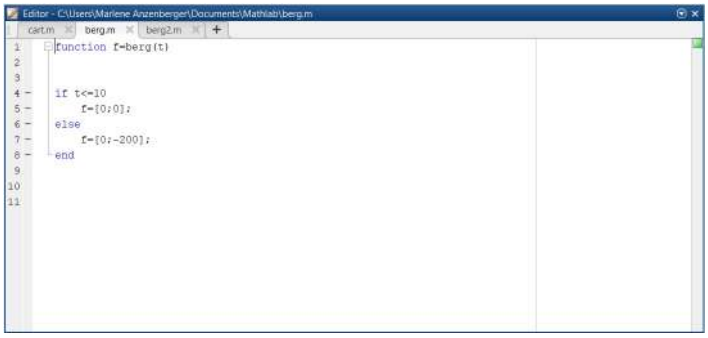


Abbildung 5: Funktion des Berges

Da wir den entsprechenden Parameterwert nicht kennen, um den Berg wieder auszugleichen, müssen wir länger herumprobieren. Wenn der Parameterwert zu hoch war, erhöht sich die Geschwindigkeit, wenn dieser zu neidrig war, wurde der Berg nicht ausreichend ausgeglichen. In der unteren Abbildung (6) sieht man den Graphen bei passenden Parameterwerten. Zum Zeitpunkt 10 erkennt man, wie der Berg das Auto abbremst. Danach gleicht der Tempomat die Störung sofort wieder aus.

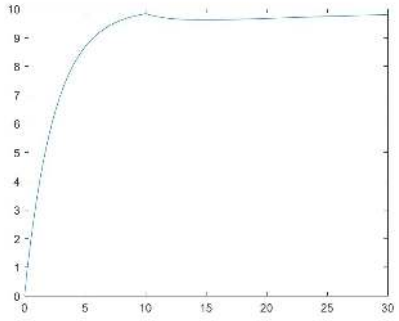


Abbildung 6: Graphik bei Beifügung des Berges

Nachdem das funktioniert, fügen wir noch eine dritte Funktion (berg2) mit einer Kraft von $250nm$ zum Zeitpunkt 20 hinzu, die ein Tal simulieren soll. Dafür wiederholen wir die Schritte, die wir auch bei der Modellierung des Berges verwendeten, sprich, wir erstellen eine neue Funktion und legen die Werte durch eine if-Abfrage fest. u belassen wir wie vorhin.

```
function [g,u]=cart(x,v,n,T)

t=linspace(0,T,n);
g=zeros(2,n);
tau=t(2)-t(1);
g(:,1)=[x;v];
inte=0;
u=zeros(n,1);
for i=2:n
    inte=inte+(10-g(2,i-1))*tau;
    if i>=3
        dif=((g(i-1))-(g(i-2)))/tau;
    else
        dif=0;
    end
    u(i)=400*(10-g(2,i-1))+20*inte+0*dif;
    g(:,i)=tau/1000*([0;u(i)]+berg(t(i-1))+berg2(t(i-1)))+[1 tau;0 1-0.05*tau]*g(:,i-1);
end
```

Abbildung 7: Tempomat mit Berg und Tal

Daraus resultiert diese Graphik:

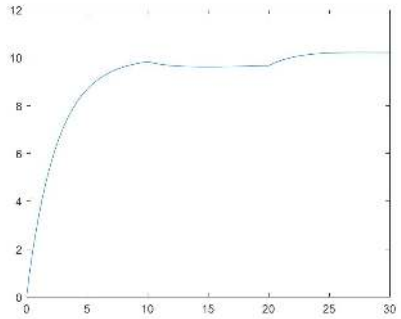


Abbildung 8: Tempomat mit Berg und Tal als Graphik

In der letzten Graphik kann man gut die Arbeit des Tempomaten erkennen. Das Auto beschleunigt binnen 5 Sekunden aus dem Stand auf die gewünschte Geschwindigkeit von $10m/s$ und hält dann zunächst das eingestellte Tempo. Zum Zeitpunkt 10 trifft das Auto auf die negative Kraft, welche das Auto abbremst. Das erkennt man an der kleinen Kante bei $t = 10$. Danach fällt der Graph ganz leicht ab, wird aber sofort wieder vom Tempomat stabilisiert. Zum Zeitpunkt 20 wirkt dann die positive Kraft; die Wirkung ist wieder durch die kleine Kante erkennbar. Auch beim simulierten Tal gleicht der Tempomat die Geschwindigkeit aus. Hierbei fällt auf, dass der Ausschlag der Kurve größer ist und die Stabilisation länger braucht als beim Berg. Das ist, aber eher auf den höheren Wert beim Tal zurück zu führen. Die entsprechende Kontrolle u dafür sieht so aus:

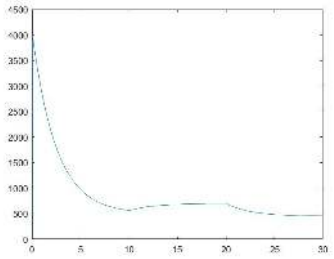


Abbildung 9: Kontrolle für den Tempomat mit Berg und Tal

Als Vergleich dazu verwenden wir die konstante Kontrolle u mit dem Wert 500, um den Vergleich zu sehen.

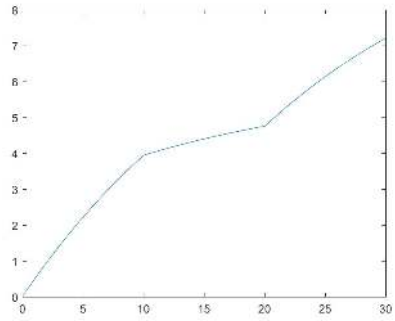


Abbildung 10: Berg und Tal ohne Kontrolle

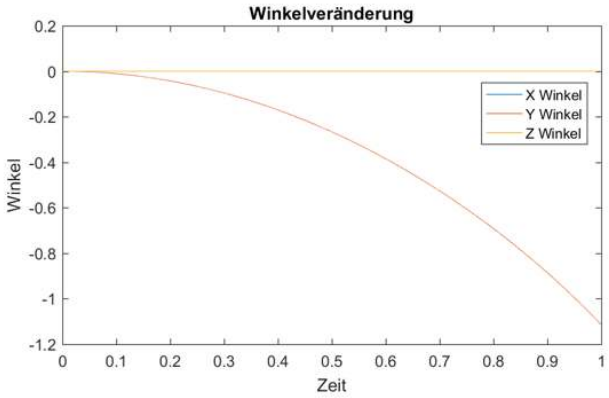
Vergleicht man die Abbildung 5 und 7 erkennt man deutlich, wie wichtig die Kontrolle für die Beibehaltung beziehungsweise für die Beschleunigung auf die gewünschte Geschwindigkeit ist. Dadurch, dass keine Kontrolle verwendet wird, beschleunigt das Auto viel langsamer und gleicht auch den Berg beziehungsweise das Tal nicht aus.

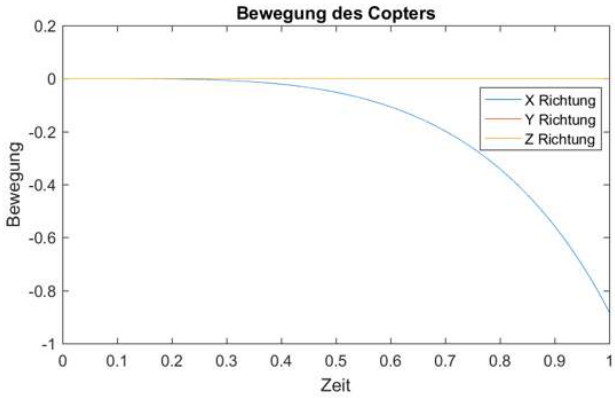
7 Szenarien mit Hilfe der Ergebnisse

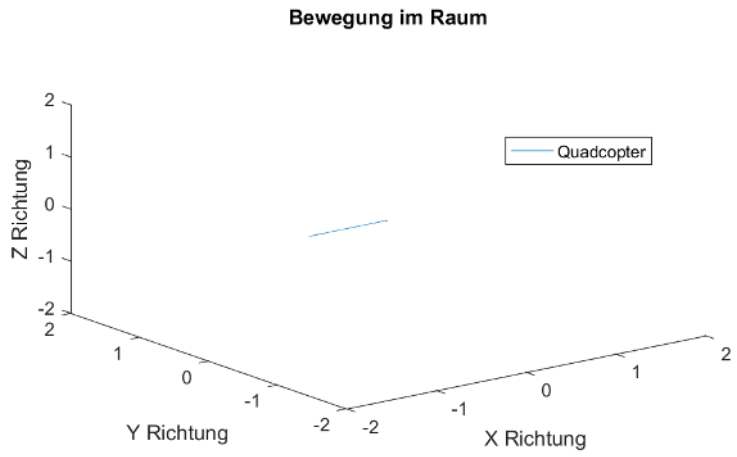
Am Anfang nehmen wir an, dass unser Quadcopter im Ursprung des Raums steht/fliegt. Alle Rotoren bewegen sich mit der gleichen Geschwindigkeit was dazu führt, dass der Quadcopter so viel Schub erzeugt, dass er im Raum schweben kann.

7.0.1 Erstes Szenario – Bewegung in Richtung X - Pitch

In diesem Szenario erhöhen wir beziehungsweise erniedrigen wir den Schub and zwei gegenüberliegenden Rotoren. Dadurch neigt sich der Quadcopter und es entsteht eine Neigung um die y-Achse und damit eine Bewegung in x-Richtung.

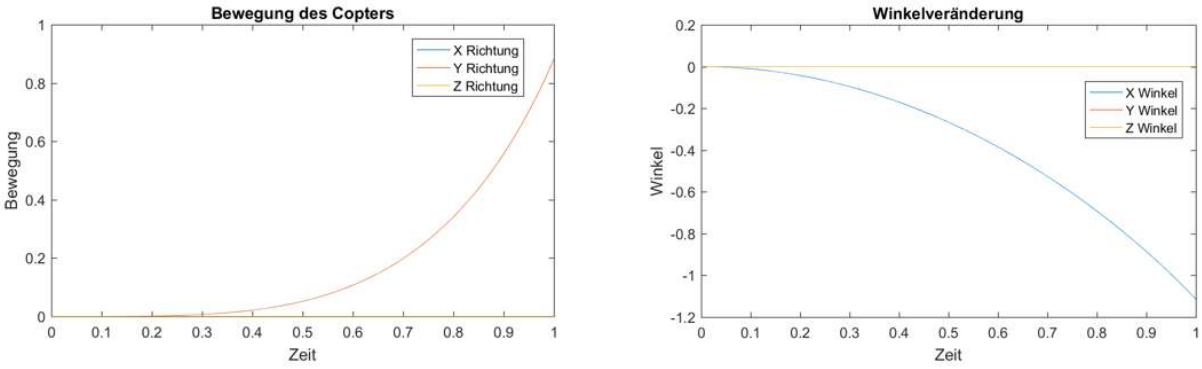


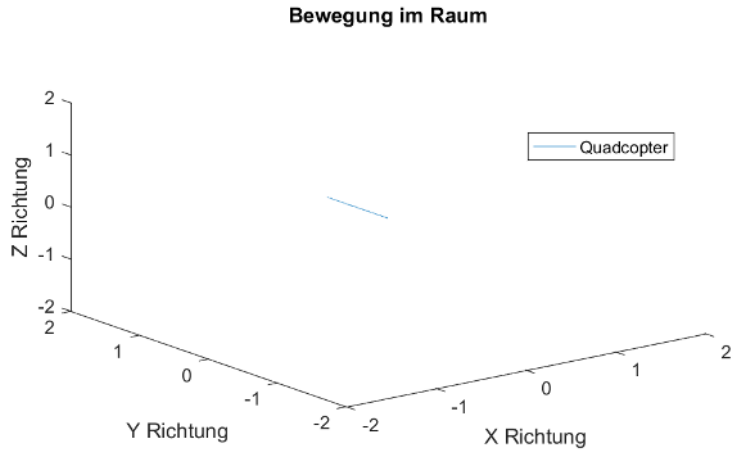




7.0.2 Zweites Szenario – Bewegung in Richtung Y - Roll

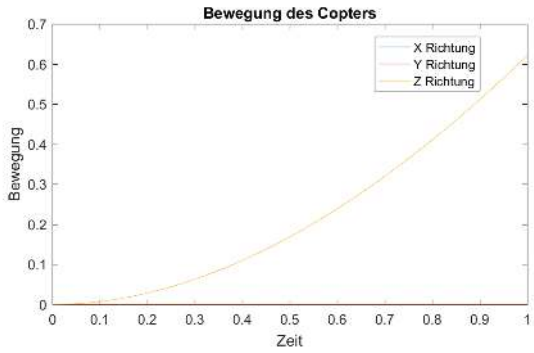
Wie im vorherigen Szenario, erhöhen wir beziehungsweise erniedrigen wir an dem anderen Paar gegenüberliegneder Rotoren den Schub. Nun wird sich der Quadcopter um die x-Achse neigen und er wird sich in die y-Richtung bewegen.

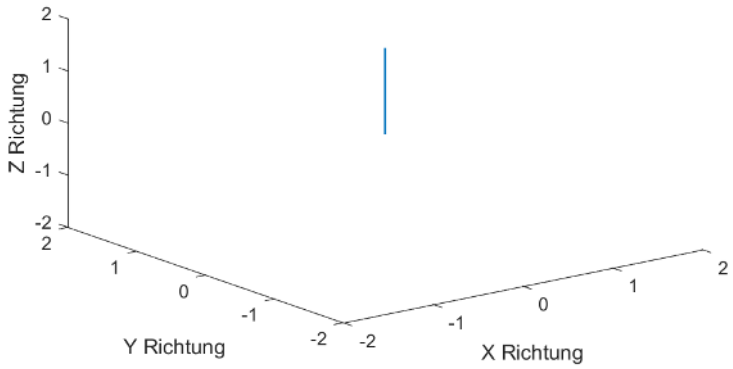




7.0.3 Drittes Szenario – Bewegung in Richtung Z - Thrust

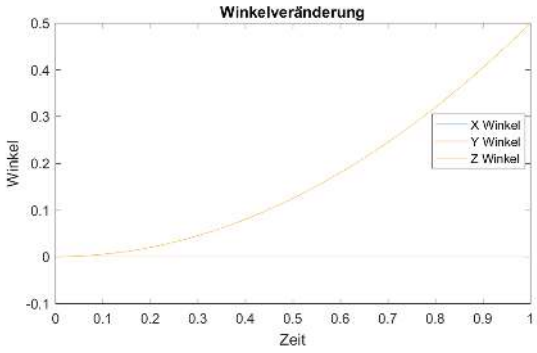
Wenn wir nun bei allen vier Rotoren den Wert für den Schub verändern, wird der Quadcopter entweder steigen oder sinken. In diesem Szenario werden wir den Schub erhöhen. Dadurch wird der Quadcopter entlang der z-Achse steigen.





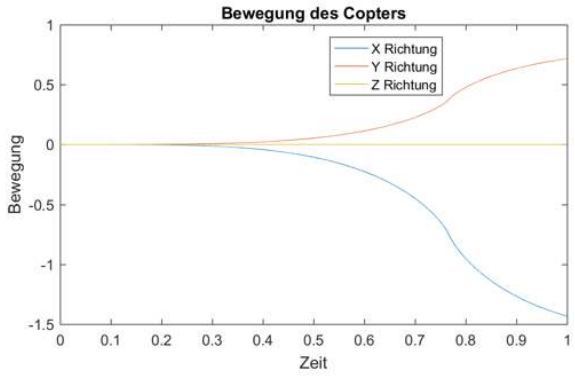
7.0.4 Viertes Szenario – Rotation um die Z Achse – Yaw

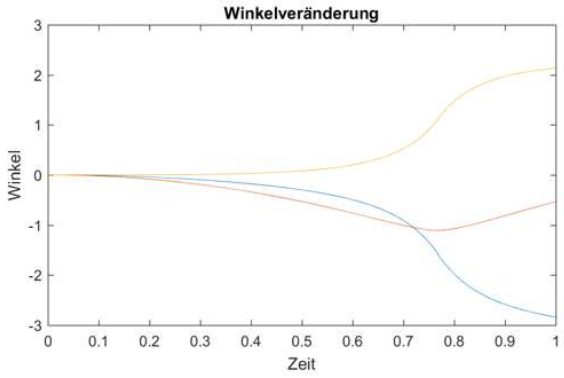
In diesem Szenario wollen wir eine Rotation um die z-Achse simulieren. Um den Quadcopter rotieren zu lassen, müssen wir die diagonalen Rotorenpaare verschieden einstellen. Einem der Rotorenpaaren wird ein positiver Schub hinzugeführt und dem anderen Paar wird ein negativer Schub hinzugeführt. In Summe gleichen sich Auftriebsschub und Abwärtsschub sich aus und es entseht keine Veränderung der Position. Doch die Drehung um die z-Achse wird in dem folgenden Graphen deutlich erkennbar.

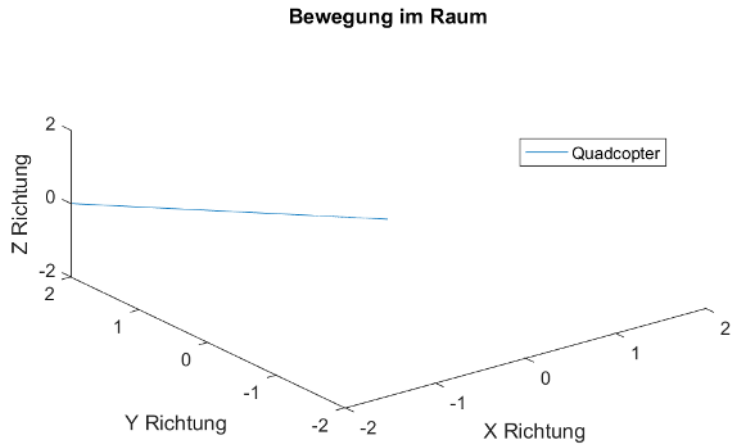


7.0.5 Fünftes Szenario - Schräger Flug - Pitch und Roll Kombination

Um einen schrägen Flug zu modellieren muss man eine Rotation sowohl um die x-Achse als auch um die y-Achse erstellen. Daher müssen also wieder die diagonalen Rotorenpaare unterschiedlich eingestellt werden. Hier bekommt wieder ein Rotor Schub addiert und einem anderen wird er abgezogen. Somit rotiert man den Quadcopter um die y-Achse und er bewegt sich in x-Richtung. Nun muss man aber, um einen schrägen Flug zu zeigen, auch das zweite Paar diagonaler Rotoren einstellen. Gleich wie bei dem ersten Paar, muss man die diagonalen Rotoren unterschiedlich einteilen. Dadurch bekommen wir jetzt auch eine Rotation um die x-Achse und der Quadcopter fliegt auf einer schrägen Bahn.

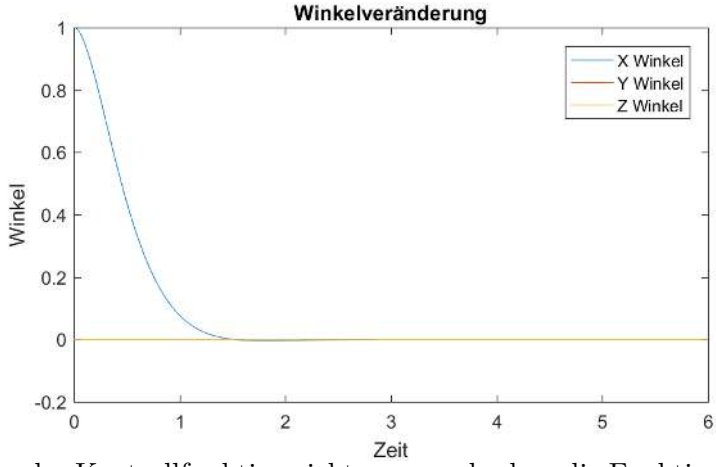




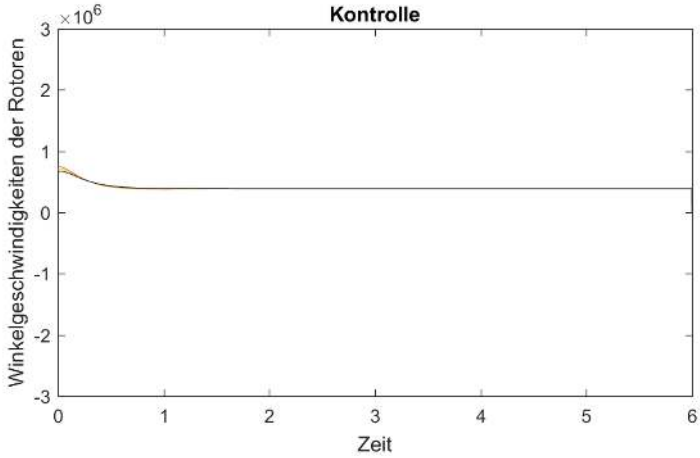


7.0.6 Feedback Stabilisierung

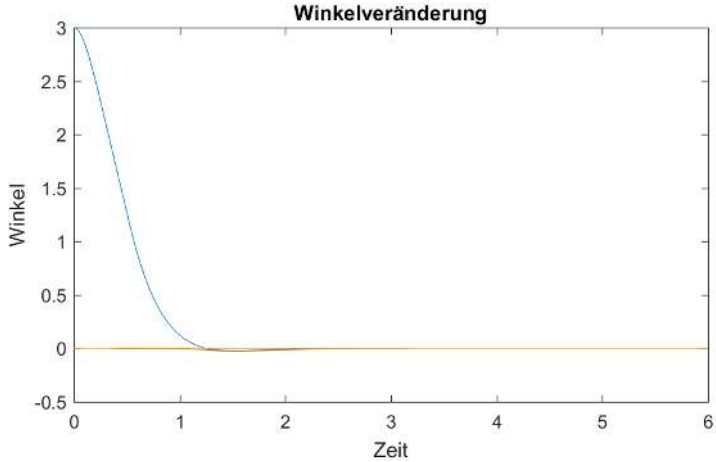
Das Ziel der Kontrolle ist es den Unterschied zwischen dem gewünschten und dem observierten Zustand zu null konvergieren zu lassen. In unserem Bespiel, startet der Quadcopter nun in einer geneigten Position. Das Ziel ist es dann, dass der Copter wieder zurück in eine waagerechte Stellung kommt. Dabei hilft ihm die Feedback Kontrolle die aus Messung der Winkel und der Winkelgeschwindigkeiten eine adäquate Kontrollfunktion berechnet. Wir geben also vor, dass der Quadcopter um Eulerwinkel $\phi = 1$ oder $\phi = 3$ Grad geneigt ist. Mit Hilfe der Kontrolle soll sich der Copter nun in einer waagerechten Lage stabilisieren, das heißt die Eulerwinkel sind null und die Ableitungen der Euler Winkel sind ebenfalls null. In dem ersten Beispiel nehmen wir an, dass der Quadcopter mit einem Eulerwinkel $\phi = 1$ Grad startet. In den gegebenen Graphen ist es deutlich zu erkennen, dass der Winkel von dem Startwert eins, schon in den ersten zwei Sekunden gegen null geht.



Im Graphen der Kontrollfunktion sieht man auch, dass die Funktion zu Beginn des Intervalls die höchsten Werte annimmt, dieser fällt dann aber monoton und konvergiert gegen eine Konstante die dem Schub entspricht die der Quadcopter benötigt um zu schweben.

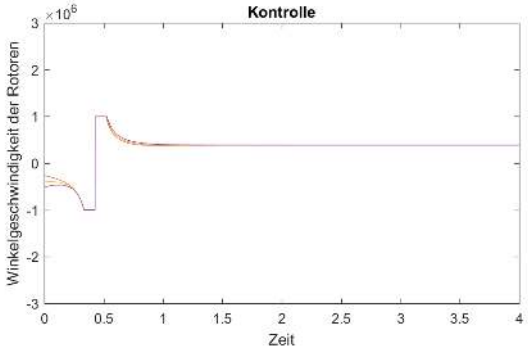


In dem zweiten Beispiel nehmen wir an, dass der Quadcopter mit einem Eulerwinkel $\phi = 3$ Grad startet. In den gegebenen Graphen ist deutlich zu erkennen, dass der Winkel auch hier von dem Startwert drei, schon in den ersten zwei Sekunden gegen null geht.



Die Kontrollfunktionen nehmen zuerst negative werte an, dies ist aber sehr unrealistisch, da das bedeuten würde, dass die Rotoren des Copters ihre Drehrichtung alle umkehren. Dies bedeutet so viel bedeutet, dass alle Rotoren ihre Schubrichtung wechseln würden. Weiterhin haben wir durch vorherige Versuche herausgefunden, dass die Kontrolle unrealistisch hohe werte annehmen würde deshalb haben wir den Wertebereich der Kontrolle auf $\pm 10^6$ eingeschränkt. Das kann man an den konstanten Teilen in dem Kontrollgraphen erkennen. Danach springt die Kontrolle auf einen positiven Wert, das heißt

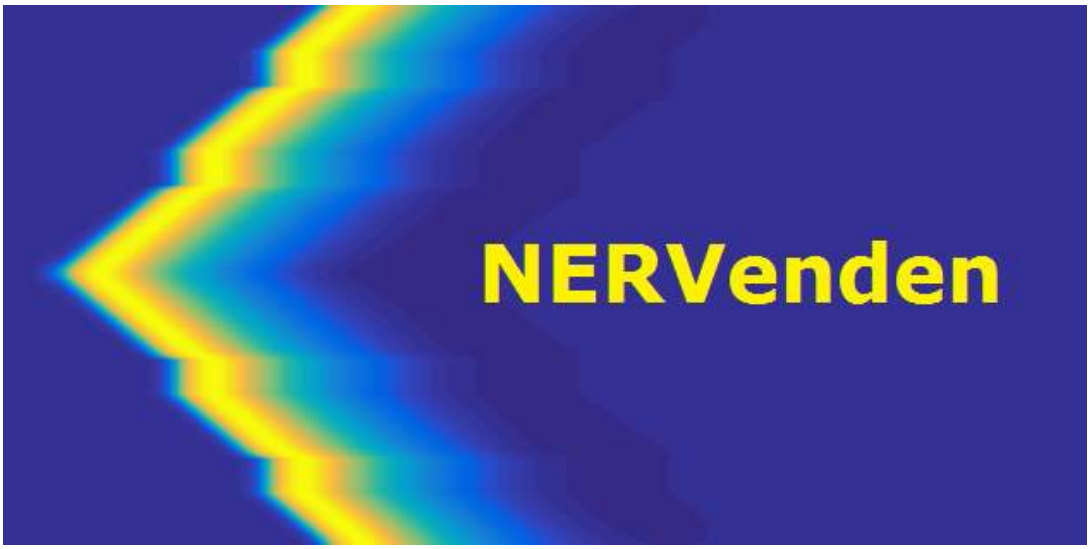
die Drehrichtung der Rotoren ist wieder normal. Dann nimmt die Kontrolle monoton ab und konvergiert auf einen Wert der es dem Copter ermöglicht weiterhin zu schweben. Trotz der Limitierung der Kontrolle, konnten wir eine Stabilisierung des Quadcopters erreichen.



Physiologie

Ausbreitung von elektrischen Wellen in biologischen Geweben

Matthias Likawetz, Jakob Prettenthaler, Kathatharian Schmidt, Teresa Reisner, Sebastian Grosinger, Georg Mandl
Betreuer:Mag. Dr. Stephen Keeling



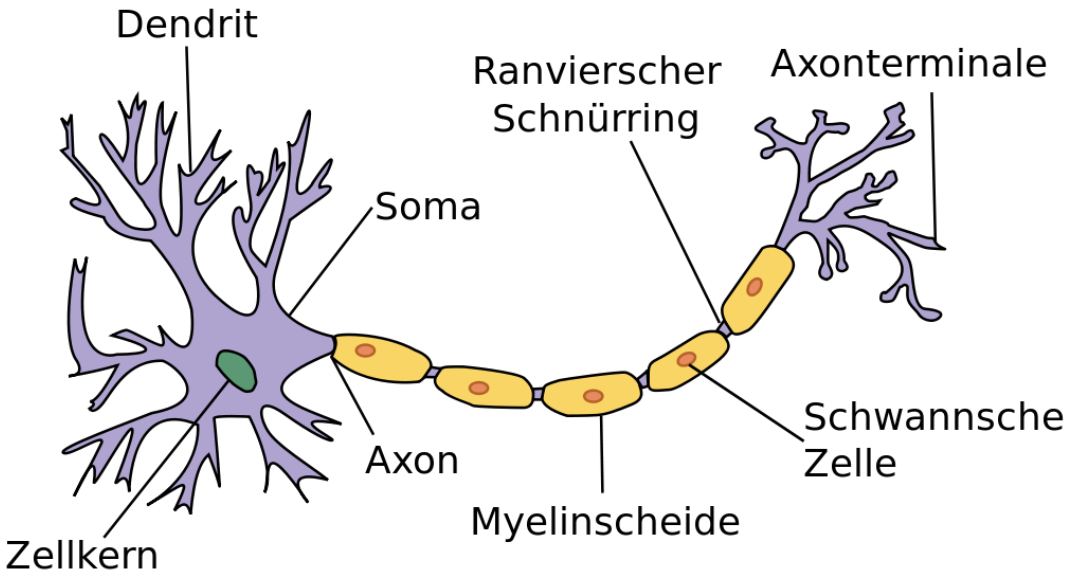
1 Einführung

Anhand ihrer Experimente und theoretischen Überlegungen erstellten Alan Lloyd Hodgkin und Andrew Fielding Huxley ein physiologisches Modell im Jahr 1952, um die ionischen Mechanismen zu erklären, die dem Aktionspotential im Riesenaxon des Tintenfisches unterliegen. Für diese Arbeit erhielten sie den 1963 Nobelpreis in Medizin.
Erstaunlicherweise kann man das Modell mit (forgeschrittenem) Schulwissen verstehen. Das erste Ziel des Projekts ist, das Hodgkin-Huxley Modell und die entsprechende Mathematik zu beherrschen und mit einem Computerprogramm zu realisieren. Dabei müssen wir uns auch mit der räumlichen Ausbreitung eines Impulses beschäftigen. Dann haben wir ein Werkzeug, mit dem wir die Entstehung und die Ausbreitung von elektrischen Wellen in biologischen Geweben simulieren können, besonders in Nerven.
Mit dem entwickelten Werkzeug können wir z.B. folgende weitere Themen untersuchen. Was kann die Erregbarkeit eines Nerven beeinflussen, und wie kann diese Erregbarkeit extern oder intern gesteuert werden? Was ist der Effekt von der Myelinscheide auf einem Nervenimpuls? Was für eine Krankheit kann entstehen, wenn die Myelinscheide beschädigt wird? Wie können Impulswellen in höhreren räumlichen Dimensionen simuliert werden, wie z.B. in Muskelgeweben? Wie kann man ein Nervenmodell vereinfachen (wie z.B. mit dem sogenannten FitzHugh-Nagumo Modell), um eine mathematische Einsicht in die Dynamik der Aktionspotentiale zu gewinnen?

2 Grundlagen

2.1 Biologisches

Um sich überhaupt mit der Ausbreitung der elektrischen Wellen im menschlichen Körper befassen zu können ist ein gewisses Grundwissen erforderlich. Deswegen wird zu Beginn das Ausbreitungsmedium Nerven erklärt. Die Nervenbahnen im menschlichen Körper bestehen aus unzählig vielen, kleinen Nerven. Nerven kann man in 3 Abschnitte gliedern: den Zellkörper, das Axon und das Axonterminal. Im Zellkörper befindet sich der Zellkern, des Weiteren führen vom Zellkörper die Dendriten weg, die Impulse empfangen können. Das Axon ist die Verbindung von Zellkörper und Axonterminal. Dieses kann mit Schwann'schen Zellen umgeben sein, jedoch ist dies nicht bei allen Lebewesen der Fall. Am Ende eines Neurons befindet sich das Axonterminal, welches mit den Synapsen zur Impulsweiterleitung dient. Die elektrischen Impulse breiten sich im Axon aus, wo sich auch das Aktionspotental bildet und sich mithilfe der Diffusion von Natrium und Kalium über das ganze Axon ausbreitet.



2.2 Aktionspotential

Ein Aktionspotential ist ein zeitliches Verhaltensmuster eines dynamischen Systems, das nach einem ausreichend großen Reiz entsteht, wobei der Abstand des Zustandes vom Ruhestand anfänglich steigend ist und beschränkt bleibt. Ein ausreichend großer Reiz muss eine gewisse Schwelle überschreiten, damit dieses Muster entsteht, denn sonst fällt der Abstand des Zustandes vom Ruhezustands monoton. Nachdem diese Schwelle überschritten wird, kann der Zustand sich später an den Ruhestand oder möglicherweise an eine Grenzzyklus annähern.

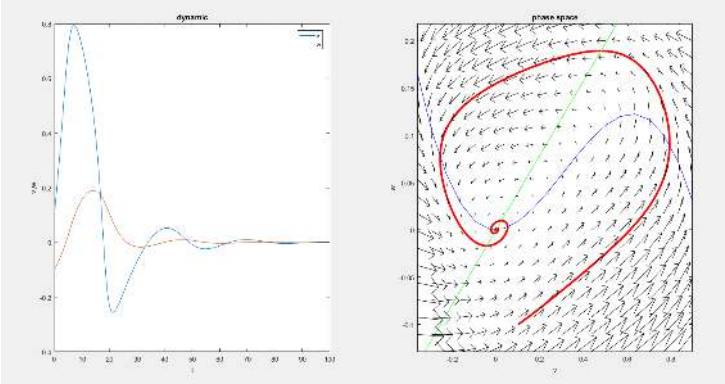
2.2.1 Phasenräume mit dem FitzHugh-Modell

In der unteren Grafik wird das Aktionspotential sowie der Phasenraum der Ionenkanäle gegen das Membranpotential mit dem FitzHugh-Nagumo-Modell dargestellt.
In der Grafik 'Dynamik' beschreibt der blaue Graph ein Aktionspotential. Die Ionenkanäle sind zusammengefasst im roten Graphen, wobei man erkennt, dass das Gleichgewicht der Ionenkanäle und des Membranpotentials bei 0 liegt.
In der rechten Abbildung wird der Phasenraum der Ionenkanäle gegen das Membranpotential dargestellt. Dabei stellt sich ein Gleichgewicht im Schnittpunkt der beiden Graphen ein. Die Anziehung des Gleichgewichts innerhalb des Phasenraums wird mit den Pfeilen grafisch dargestellt. Je näher man dem Gleichgewicht kommt, desto größer ist dessen Anziehung. Für diesen Phasenraum gelten folgende Bedingungen.

$$\begin{cases} 0 &= v(k) + dt * (f(v(k)) + veq) - f(veq) - w(k) \\ w(k+1) &= 0 \end{cases}$$

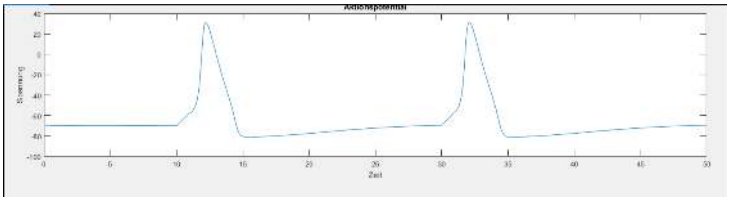
Für den Code in Matlab gilt in einer for-Schleife um den Phasenraum numerisch zu lösen und darzustellen folgendes

$$\begin{cases} v(k+1) &= v(k) + dt * (f(v(k)) + veq) - f(veq) - w(k) \\ w(k+1) &= w(k) + dt * e * (v(k) - g * w * (k)) \end{cases}$$



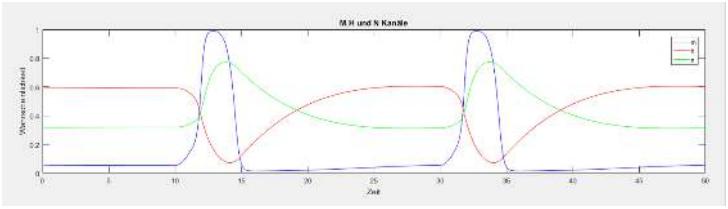
2.2.2 Biologischer Aspekt des Aktionspotentials

In Ruhelage hat ein Nerv eine Spannung von -70 mV. Um ein Aktionspotential zu erreichen, muss der Stimulus, der von einer anderen Nervenzelle geschickt wird, die Spannung auf
-55 mV erhöhen. Erst bei -55 mV werden genügend Neurotransmitter ausgeschüttet um ein Aktionspotential auszulösen. Durch die Ausschüttung der Neurotransmitter werden Natrium und Kaliumkanäle geöffnet. Diese Kanäle besitzen 4 Schleusen, durch die Ionen durchdiffundieren können. Die Kaliumkanäle bestehen aus vier n-Kanälen und die Natriumkanäle aus drei m-Kanälen und einem h-Kanal.



In der Abbildung oben sind aufeinanderfolgende Aktionspotentiale dargestellt.

$$v = (v + (dt/C) * (gE + izero(t)))/(1 + (dt/C) * g) \tag{1}$$



Wie im Bild zu sehen, stehen die m-Kanäle und die h-Kanäle in Wechselwirkung. Jedes Neuron hat eine Ionenpumpe, ohne die die Zelle ihr Volumen nicht konstant halten könnte. Diese Pumpe ist ein Membranprotein, welches Natriumionen aus der Zelle heraus und Kaliumionen in die Zelle hineinpumpt. Die Pumpe benötigt Energie in Form von ATP und pumpt je drei Na^+ und zwei K^+ Ionen. Auf dieses Verhältnis werden wir später noch zu sprechen kommen. Im Folgenden gehen wir von einer Pumprate von 1:1 aus. In der Nervenzelle erfüllt diese Pumpe neben der Erhaltung des Volumens noch eine andere Aufgabe. Durch die Diffusion der Ionen verändert sich das Membranpotential und der Unterschied der Ladung auf den beiden Seiten der Membran.

$$\begin{cases} m &= \alpha m(v)/(\alpha m(v) + \beta m(v)) \\ h &= \alpha h(v)/(\alpha h(v) + \beta h(v)) \\ n &= \alpha n(v)/(\alpha n(v) + \beta n(v)) \end{cases}$$

3 Diffusion

Im wesentlichen unterscheiden wir zwei Arten von Diffusion; Neumann und Dirichlet. Um diese beiden Arten recht einfach erklären zu können, wird es anhand der Temperaturanpassung eines Stabes erklärt und dehnen es erst später auf die Neuronen aus.

3.1 Dirichlet

Bei den Rahmenbedingungen von Dirichlet wird davon ausgegangen, dass es an den beiden Enden des Stabes fixe Temperaturen gibt. Teilen wir nun den Stab (mit der Länge L) in N Teile auf, und geben den ersten sowie den n-ten Teil eine Anfangstemperatur;

$$\begin{cases} T_1 &= 0.5 \\ T_N &= 0.25 \end{cases}$$

Nun wollen wir die Diffusion zeitlich und räumlich ableiten um sie so auch grafisch darstellen zu können, dafür benötigen wir die folgenden Kriterien um dies zu machen

$$\begin{cases} \partial_t T_t &= \alpha T_{xx}, & x \in (0, L), & t \in (0, Te) \\ T(0,t) &= T_0, & T(t) = T_L, & t > 0 \\ T(x,0) &= T_0(x), & x \in (0, L), & t = 0 \end{cases}$$

Wenn das Programm (in Matlab) vollständig ausgeführt wurde begibt sich die Funktion,die zeitlich und räumlich abgeleitet wurde, in ein Gleichgewicht der beiden gegebenen Temperaturen.

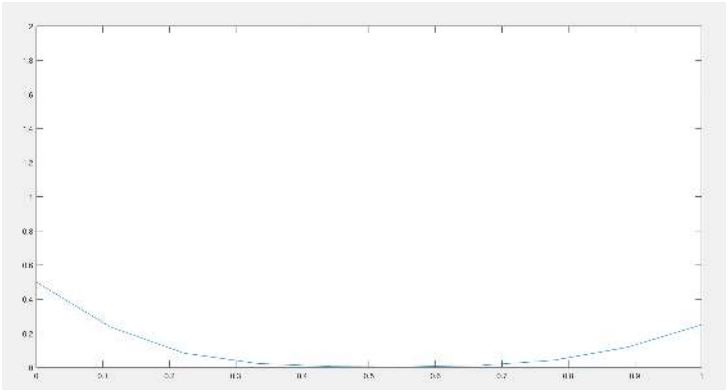
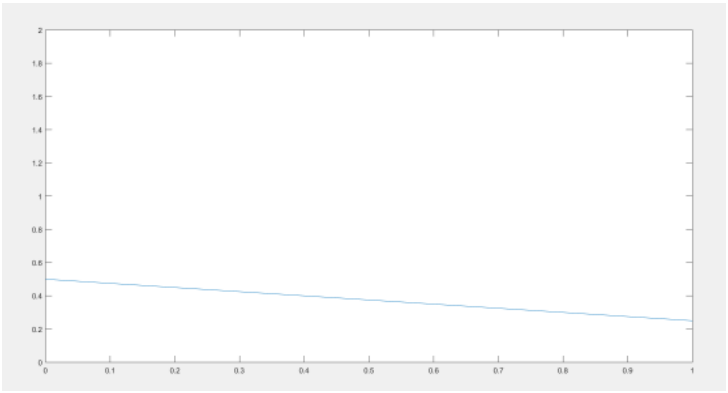


Abbildung oben: der Beginn der Diffusion mit zwei fixen Temperaturen
Abbildung unten: die vollständig abgelaufene Diffusion der Temperatur entlang des Stabes



3.2 Neumann

Bei den Neumannschen Rahmenbedingungen teilen wir den Stab, wie zuvor auch bei den Dirchleten Rahmenbedingungen, in N Teile. Der mittlere Teil des Stabes sei T_i, welcher eine fixe Temperatur hat. Diese Temperatur diffundiert nun von der Zelle T_i zu den beiden Nachbarzellen $T_i - 1$ und $T_i + 1$. Diesr Vorgang wiederholt sich so lange, bis die Diffusion der Temperatur bei den beiden äußersten Zellen angelangt.

$$\begin{cases} \partial T_i' &= \alpha_{T_{i+1}-T_i} + \alpha_{T_{i-1}-T_i} \\ T_i' &= \alpha_{T_1-T_N} \\ T_i' &= \alpha_{T_{N-1}-T_N} \end{cases}$$

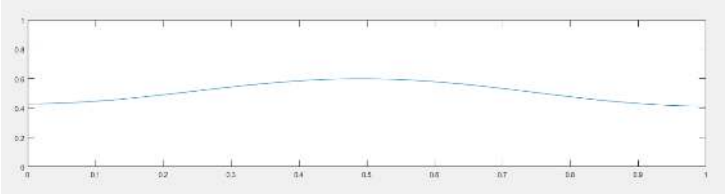
Um dies auch numerisch zu lösen und im Anschluss auch grafisch darzustellen muss eine Matrix erstellt werden, die folgende Bedingungen erfüllt;

$$\begin{cases} T' &= A * T \\ T(0) &= T_0 \end{cases}$$

$$\begin{bmatrix} T_1' \\ T_2' \\ \vdots \\ T_i' \\ \vdots \\ T_{N-1}' \\ T_N' \end{bmatrix} = (1/\ dt^2) * \begin{bmatrix} -1 & +1 & 0 & 0 & 0 & 0 & 0 \\ +1 & -2 & +1 & 0 & 0 & 0 & 0 \\ \ddots & & & & & & \\ 0 & 0 & +1 & -2 & +1 & 0 & 0 \\ \ddots & & & & & & \\ 0 & 0 & 0 & 0 & +1 & -2 & +1 \\ 0 & 0 & 0 & 0 & 0 & +1 & -1 \end{bmatrix} * \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{N-1} \\ T_N \end{bmatrix}$$

Durch die Verwendung dieser Matrizen ist es möglich, die Neumannsche Diffusion in Matlab numerisch zu lösen und grafisch auszugeben. Die mittlere Matrix wird A genannt um die folgende Formel zu vereinfachen und die Diffusion nun auch im Raum darzustellen.

$$T = T + dt * A * T \tag{2}$$



4 Hodgkin Huxley Modell

4.1 Allgemeines

Das Hodgkin-Huxley-Modell ist ein mathematisches Modell zur Simulation von Aktionspotentialen in Neuronen. Dieses Modell geht von einem räumlich fixierten Axon aus, wodurch nur die Variable t für Zeit betrachtet werden muss. Dies hat den Effekt, dass die Berechnungen sich auf gewöhnliche Differentialgleichungen beschränken. Betrachtet man zusätzlich die räumliche Variable x, stellt man die Verbindung zur Diffusion her und erreicht ein dreidimensionales Modell. Dieses werden wir später genauer betrachten.

4.2 Liste der verwendeten Variablen:

Variable	Bedeutung
p	Pumprate der Zelle
q	Elementarladung
Na^+	Natrium Ion
K^+	Kalium Ion
α_{Na}/α_K	Permeabilität der Zellmembran gegenüber Na/K
v	Membranpotential (Unterschied in Potential innerhalb und außerhalb der Zelle)
I	Ionischer Strom
C	Kapazitanz der Zellmembran
$g_{Na}/g_K/g_{Cl}$	Leitfähigkeit der Membran gegenüber Na/K/Cl
g	gesamte Leitfähigkeit der Membran
E	Gleichgewichtsspannung der Zelle entlang der Membran
n(t)	Wahrscheinlichkeit dass ein n-Tor zum Zeitpunkt t offen ist
m(t)	Wahrscheinlichkeit dass ein m-Tor zum Zeitpunkt t offen ist
h(t)	Wahrscheinlichkeit dass ein h-Tor zum Zeitpunkt t offen ist
$\alpha_n(v)/\alpha_m(v)\alpha_h(v)$	Öffnungsrate der $n/m/h$ Tore abhängig von v
$\beta_n(v)/\beta_m(v)/\beta_h(v)$	Rate der Schließung der $n/m/h$ Tore abhängig von v
t	Zeit

Im Gegensatz zu einer gewöhnlichen Zelle hat die Nervenzelle die Möglichkeit, ihre Leitfähigkeit gegenüber Ionen zu kontrollieren. Sie tut dies mithilfe sogenannter Natrium und Kaliumkanäle.

Das Membranpotential v ist der Unterschied in Potential innerhalb und außerhalb der Zelle und kann durch diese Gleichung dargestellt werden:

$$Cv = q(V[Na^+]_i + V[K^+]_i - V[Cl^-]_i - xz) \tag{3}$$

C ist die Kapazität der Zellmembran, V das Volumen der Zelle und xz die Anzahl der negativen Elementarladungen innerhalb der Zelle. Diese Elementarladungen entstehen durch Makromoleküle in der Zelle und werden als konstant angenommen.

Der Kaliumkanal besteht aus vier Toren derselben Art, den n-Toren. Der Natriumkanal besteht aus drei m-Toren und einem h-Tor. Ihre Öffnungs-, bzw Schließungsraten α bzw. β sind vom Membranpotential v abhängig. Wir nennen die jeweilige Wahrscheinlichkeit, dass ein Tor offen ist $n(t), m(t)$ und $h(t)$. Diese Werte sind Funktionen von v, also vom aktuellen Membranpotential, welches wiederum von t abhängt, und liegen zwischen 0 und 1. All diese Tore müssen gleichzeitig offen sein, damit der Kanal Ionen durchlässt. Daher ist die Wahrscheinlichkeit eines offenen Kaliumkanals n^4 und die eines offenen Natriumkanals $m^3 * h$.

Die Formeln für den jeweiligen Strom von Natrium, Kalium und Chlor sind:

$$I_{Na} = g_{Na}(v - (kT/q)log([Na^+]_o/[Na^+]_i)) + pq \tag{4}$$
$$I_K = g_K(v - (kT/q)log([K^+]_o/[K^+]_i)) - pq, \tag{5}$$
$$I_{Cl} = g_{Cl}(v + (kT/q)log([Cl^-]_o/[Cl^-]_i)). \tag{6}$$

Da der gesamte Ionenstrom die Summe aller spezifischen Ionenströme ist, lässt sich pq, die Anzahl der gepumpten Ionen, kürzen. In der Realität beträgt das Verhältnis der gepumpten Natriumionen zu Kaliumionen nicht 1:1, sondern 3:2. Wir werden diesen Fall später genau betrachten.

$$E = \frac{g_{Na}E_{Na} + g_K E_K + g_{Cl}E_{Cl}}{g_{Na} + g_K + g_{Cl}} \tag{7}$$

E ist der gewichtete Durchschnitt der einzelnen Gleichgewichtsspannungen E_{Na}, E_K und E_{Cl} entlang der Membran. Das Membranpotential v nähert sich immer dem Wert E. Ist also $v < E$, so ist die erste Ableitung von v $v' > 0$, und ist $v > E$ gilt das Gegenteil, also $v' < 0$.

Sind die Kanäle geschlossen, kann der Ionenfluss als 0 angenommen werden.

Wird nun ein Reiz empfangen, verändert sich das Membranpotential v, was zu veränderten Werten von α und β führt. Dadurch öffnen sich erst Natrium-, und dann Kaliumkanäle. Dies wiederum führt dazu, dass die gesamte Leitfähigkeit g der Membran steigt.

$$g = g_{Na} + g_K + g_{Cl} \tag{8}$$

Die Leitfähigkeit der Membran setzt sich aus den Leitfähigkeiten gegenüber den einzelnen Ionen zusammen.

Steigt nun die Leitfähigkeit der Membran, verändert sich der Ionenstrom I. Um dies mathematisch auszudrücken, leiten wir Gleichung 3 zeitlich ab.

$$C\dot{v} = -g_{Na}(v - E_{Na}) - g_K(v - E_K) + g_{Cl}(v - E_{Cl}) \tag{9}$$

Beachtet man Gleichungen 7 und 8 lässt sich dies so anschreiben:

$$C\dot{v} + g(v - E) = 0. \tag{10}$$

4.3 Code

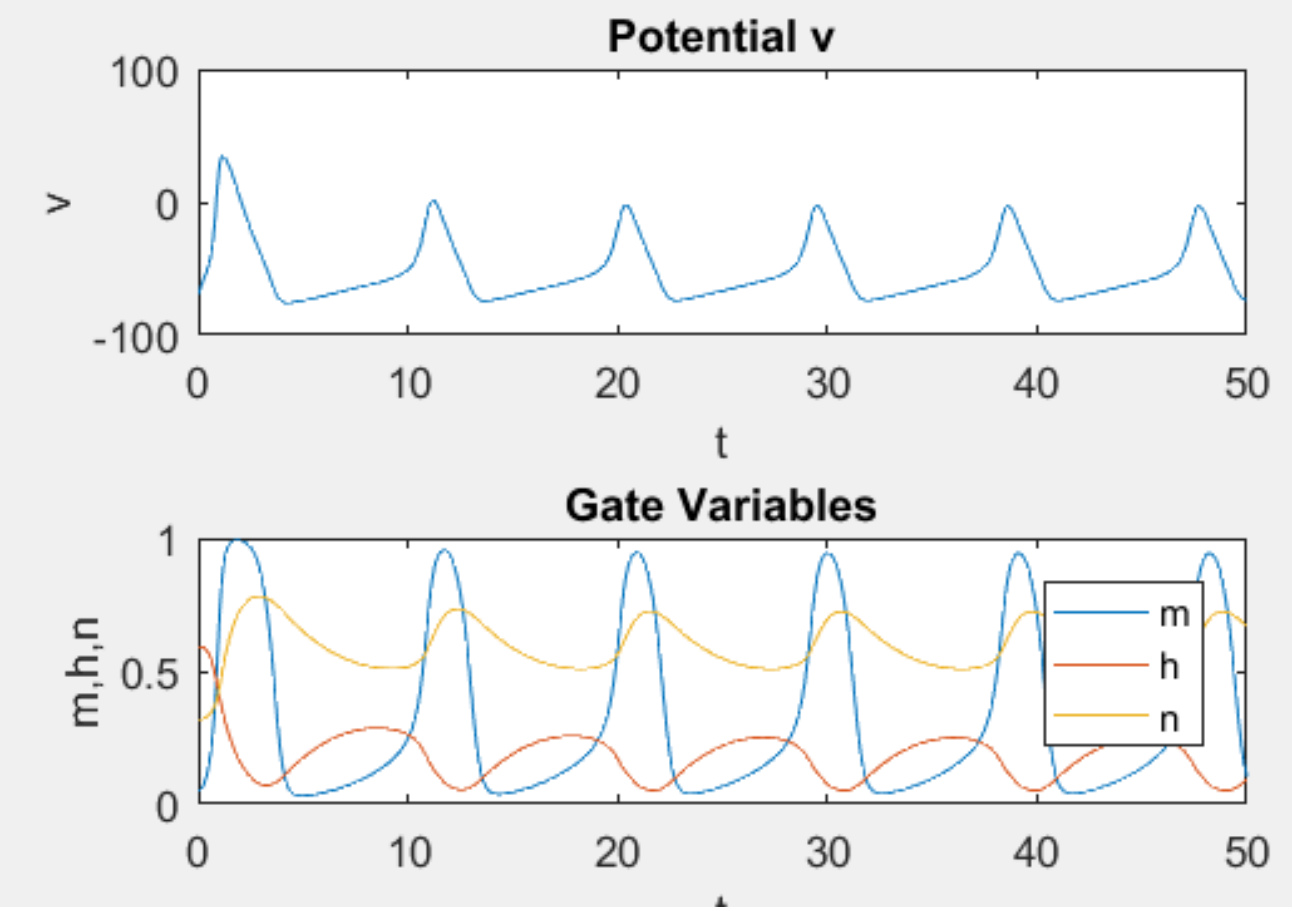
4.3.1 Numerik

Um nun das Hodgkin-Huxley-Modell in eine Form zu bringen, sodass es in Matlab programmiert werden kann, müssen wir die obige Gleichung numerisch anschreiben und lösen.

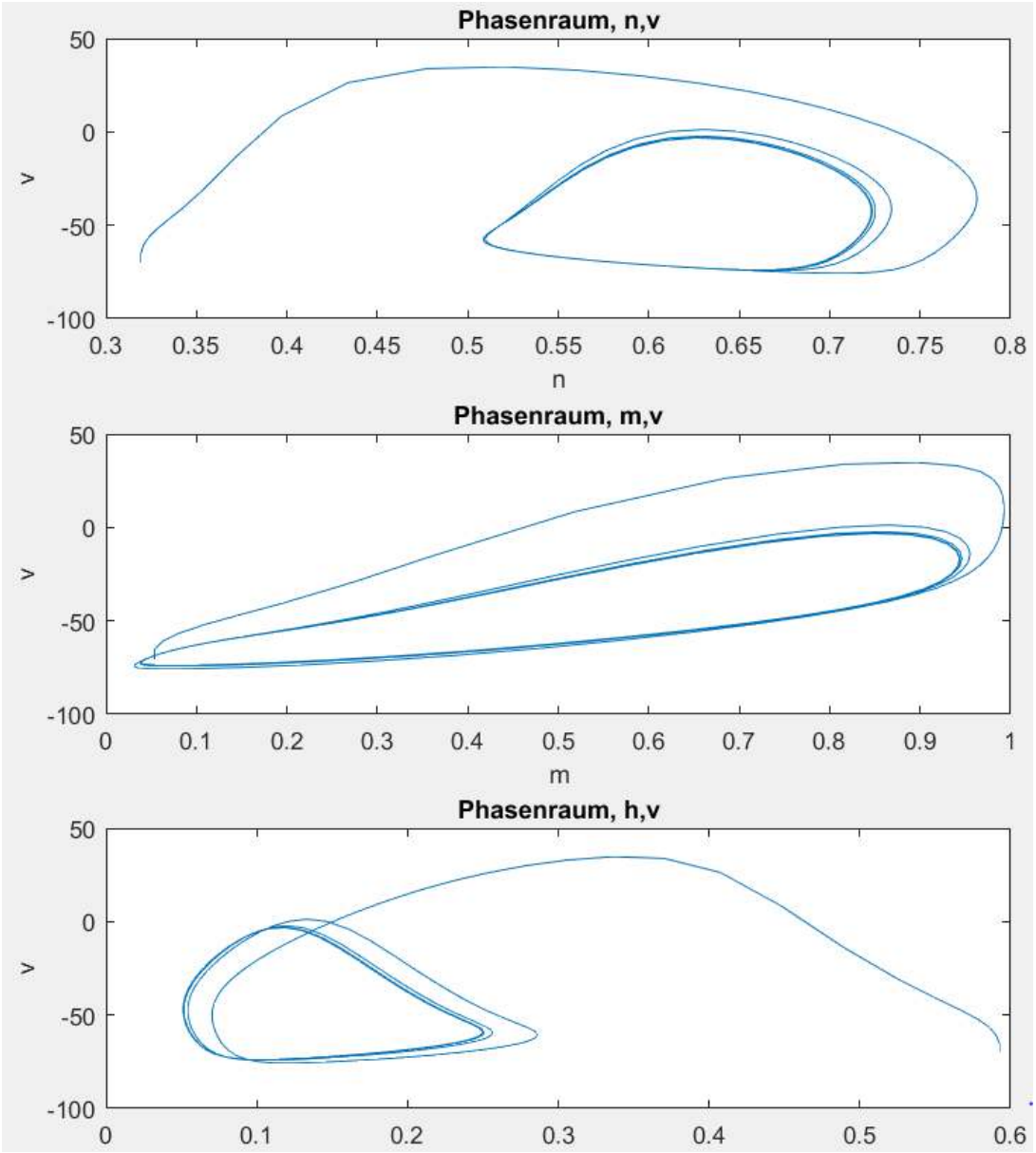
```
for k=1:Nt
v = (C*v + dt*(gE + i0(t)))/(C + dt*g);
```

4.3.2 Phasenraum

Der Phasenraum beschreibt hier das Membranpotential in Abhängigkeit aller Schleusen m, h, n. Mit zunehmender Zeit stellt sich ein sich ständig wiederholender Zyklus, der einem Aktionspotential mit dessen Refraktärzeit entspricht. Der wiederholende Zyklus erklärt sich biologisch aufgrund der 1:1-Rate der Natrium-Kalium-Pumpe. Da sich die Schleusen für die Kaliumionen zuerst, also am Anfang des Aktionspotential, öffnen und dann die Kanäle für Natriumionen, wird ein ständig wiederkehrendes Aktionspotential erzeugt. In der folgenden Grafik wird die Verbindung von den Aktionspotentialen und den Kanälen sichtbar.



Und die dazugehörigen Phasenräume mit ihren Endloszyklen:



4.4 Implementation in Matlab

Für dieses Modell wird angenommen, dass das Membranpotential bis zur Zeit $t = 0$ konstant auf -70 millivolt gehalten wird. Wir nennen das in unserem Code *vhold*. Zum Zeitpunkt $t = 0$ gehen wir von einem Reiz aus, der v auf einen von uns bestimmten Wert hebt. Wir nennen diesen Wert *vstart*. Ist dieser unter einer gewissen Reizschwelle, reicht der Unterschied zwischen *vhold* und *vstart* nicht aus, um ein Aktionspotential auszulösen.

Es ist weiters möglich, zusätzliche Reize zu späteren Zeitpunkten zu simulieren. In unserem Code nennen wir diese $izero(t)$. Wir implementieren das Hodgkin-Huxley Modell in mehreren verschiedenen Dateien, die mit Querverweisen verbunden sind. Die Hauptdatei, die all diese Dateien miteinander verbindet und ausführt, nennen wir HH.m, kurz für Hodgkin-Huxley.

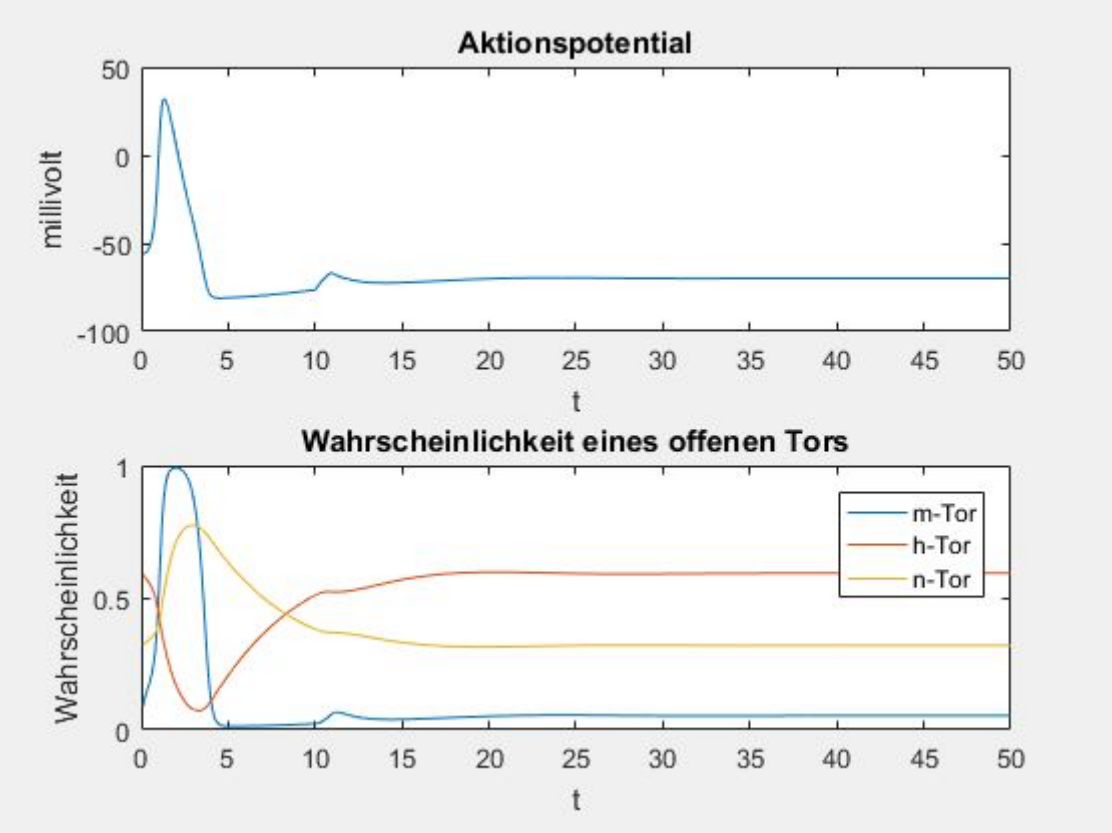
Hier ist der Code der Datei HH.m:

```
% numerical solution of the space-clamped Hodgkin-Huxley equations

clear all
```

```
clf
global t1p t2p ip

% initialization
in_HH
in_mhnv

for klok=1:klokmax
    t = klok*dt;
% update gate variables
    m = snew(m,alpham(v),betam(v),dt);
    h = snew(h,alphah(v),betah(v),dt);
    n = snew(n,alphan(v),betan(v),dt);
% update conductances
    gNa = gNabar*(m^3)*h;
    gK  = gKbar*(n^4);
    g   = gNa + gK + gLbar;
    gE  = gNa*ENa + gK*EK + gLbar*EL;
% update v
    v = (v + (dt/C)*(gE + izero(t)))/(1 + (dt/C)*g);
% store results for plotting at the end
    mhn_plot(:,klok) = [m h n]';
    v_plot(klok) = v;
    t_plot(klok) = t;
end
subplot(2,1,1)
plot(t_plot,v_plot)
subplot(2,1,2)
plot(t_plot,mhn_plot)
```

Der erste Graph, der aus diesem Code resultiert, zeigt ein Aktionspotential und einen vergeblichen Versuch, ein Aktionspotential auszulösen. Dieser entsteht durch einen Reiz, der die Reizschwelle nicht erreicht.

Der zweite Graph zeigt die Wahrscheinlichkeiten eines offenen n, m oder h Tor zum Zeitpunkt t.



5 Pumpverhältnis (1:1) & (3:2)

Wie bereits erwähnt wurde, wird, um zu vereinfachen, im Hodgkin-Huxley-Modell von einem 1:1 Verhältnis der Natriumionen und Kaliumionen ausgegangen. Da menschliche Nervenzellen aber ein 3:2 Verhältnis aufweisen, adaptieren wir das Hodgkin-Huxley-Modell folgendermaßen:

5.1 1:1 Verhältnis

Im Hodgkin-Huxley-Modell wird davon ausgegangen, dass die Ionenpumpe jeweils ein Natriumion aus der Zelle hinaus und ein Kaliumion in die Zelle hinein transportiert. Infolgedessen sind die diversen Berechnungen, Ansätze und Gleichungen für die Ionenströme, Gleichgewichtspotential und die Differentialgleichung für das Membranpotential v auf diese Bedingung ausgelegt.

5.1.1 Ionenströme

Aufgrund der Annahmen kommt man auf die Gleichungen für die Ionenströme der einzelnen Ionen.

$$I_{\mathrm{Na}} = g_{\mathrm{Na}}(v - (kT/q)\ln([\mathrm{Na}^+]_o/[\mathrm{Na}^+]_i)) + pq \tag{11}$$
$$I_{\mathrm{K}} = g_{\mathrm{K}}(v - (kT/q)\ln([\mathrm{K}^+]_o/[K^+]_i)) - pq \tag{12}$$
$$I_{\mathrm{Cl}} = g_{\mathrm{Cl}}(v - (kT/q)\ln([\mathrm{Cl}^+]_o/[Cl^+]_i)) \tag{13}$$

Es wird angenommen, dass die Werte $I_{\mathrm{Na}}, I_{\mathrm{K}} und I_{\mathrm{Cl}}$ null sind. Dadurch lassen sich Werte für die Quotienten der inneren Konzentration durch die äußere Konzentration der einzelnen Ionen berechnen. Folglich lassen sich zahlreiche Variablen berechnen, die für die spätere Berechnung von v von Nöten sind. Unter anderem ergibt sich dann für das 1:1 Verhältnis

$$p_{\mathrm{opt}} = (kT/q^2)(g_{\mathrm{Na}}g_{\mathrm{K}}/(g_{\mathrm{Na}} + g_{\mathrm{K}}))ln([\mathrm{Na}^+]_{\mathrm{o}}\mathrm{g}_{\mathrm{K}}/(\mathrm{g}_{\mathrm{Na}}[\mathrm{K}^+]_{\mathrm{o}})). \tag{14}$$

5.1.2 Gleichgewichtspotential und Differentialgleichung

Durch weitere Rechenschritte lässt sich mit der Differentialgleichung von $C\dot{v}$ nach der Zeit abgeleitet ein Gleichgewichtspotential definieren. Und zwar gilt dann

$$C\dot{v} + g(v - E) = 0. \tag{15}$$

Außerdem lässt sich $v(t)$ bestimmen.

$$v(t) = \frac{C * v(t - \Delta t) + \Delta t(g(t)E(t) + i_o(t))}{C + \frac{\Delta t}{C} g(t)} \tag{16}$$

5.2 3:2 Verhältnis

Da in den menschlichen Nervenzellen anstatt des 1:1 Verhältnisses ein 3:2 Verhältnis gilt, müssen zahlreiche Berechnungen neu ausgeführt werden. Das heißt, dass jetzt nicht mehr jeweils ein Natriumion hinaus und ein Kaliumion hinein gepumpt wird, sondern, dass 3 Natriumionen hinaus und 2 Kaliumionen hinein gepumpt werden.

5.2.1 Ionenströme

Wenn die Gleichungen (11),(12) und (13) herangezogen werden, müssen die Ausdrücke $+pq$ und $-pq$ aufgrund der 3:2 Pumpe durch $+3pq$ und $-2pq$ ersetzt werden.

$$I_{\mathrm{Na}} = g_{\mathrm{Na}}(v - (kT/q)\ln([\mathrm{Na}^+]_o/[\mathrm{Na}^+]_i)) + 3pq \tag{17}$$

$$I_{\mathrm{K}} = g_{\mathrm{K}}(v - (kT/q)\ln([\mathrm{K}^+]_o/[K^+]_i)) - 2pq \tag{18}$$

$$I_{\mathrm{Cl}} = g_{\mathrm{Cl}}(v - (kT/q)\ln([\mathrm{Cl}^+]_o/[Cl^+]_i)) \tag{19}$$

Diese Gleichungen werden nun wieder null gesetzt und die Quotienten der inneren Konzentration durch die äußere Konzentration der einzelnen Ionen berechnet. Bereits hier ergeben sich die ersten Unterschiede zum 1:1 Verhältnis. Durch weitere Berechnungen entsteht ein neues p /eine neue Pumprate. Da es für die Berechnungen und die spätere Implementation in den Matlab-Code einfacher ist, wird gleich eine neue Formel für pq bestimmt.

$$pq = ln(\frac{[\mathrm{Na}^+]_o * 3g_{\mathrm{K}}}{[\mathrm{K}^+]_o * 2g_{\mathrm{Na}}}) * \frac{g_{\mathrm{K}} * g_{\mathrm{Na}} * kT}{q * (2g_{\mathrm{Na}} + 3g_{\mathrm{K}})} \tag{20}$$

Wie man nun sehen kann, ist pq dynamisch und hängt von den Leitfähigkeiten der Membran gegenüber Natrium, Kalium und Chlor.

5.2.2 Differentialgleichung

Aufgrund des neuen Verhältnis muss auch die Differentialgleichung für v angepasst werden. Man weiß,

$$C\dot{v} = \frac{d}{dt}(Vq[\mathrm{Na}^+]_i) + \frac{d}{dt}(vq[\mathrm{K}^+]_i + \frac{d}{dt}(V(-q)[\mathrm{Cl}^-]_i. \tag{21}$$

Hier können die einzelnen Summanden durch $-I_{\mathrm{Na}}$, $-I_{\mathrm{K}}$ und $-I_{\mathrm{Cl}}$, die bereits in (17),(18) und (19) für das neue 3:2 Verhältnis berechnet wurden, ersetzt werden. Daraus ergibt sich durch Vereinfachung und Einsetzen ein neues Gleichgewichtspotential

$$C\dot{v} + g(v - E) + pq = 0, \tag{22}$$

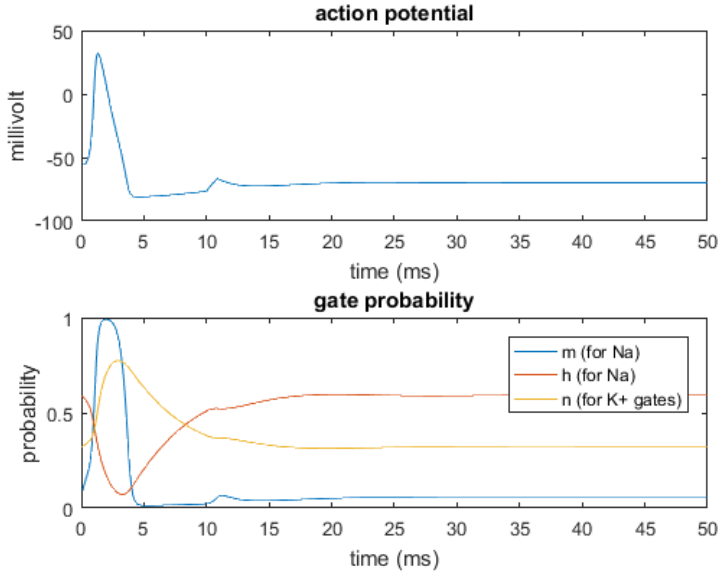
welches nach $v(t)$ umgeformt wird.

$$v(t) = \frac{C * v(t - \Delta t) + \Delta t(g(t)E(t) + i_o(t) - pq)}{C + \frac{\Delta t}{C} g(t)} \tag{23}$$

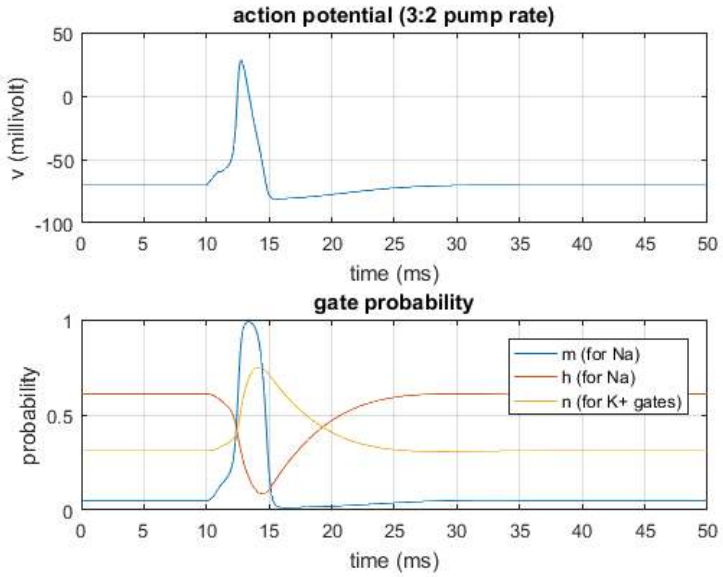
Wenn man nun diese erhaltene Gleichung (23) mit $v(t)$ (16) für das 1:1 Verhältnis vergleicht, erkennt man, dass in der zweiten Klammer des Zählers ein $-pq$ für das 3:2 Verhältnis dazugekommen ist. Dieser Ausdruck muss nun im Matlab-Code des 1:1 Verhältnis in der Datei `HH.m` implementiert werden. Des Weiteren muss natürlich auch pq in `HH.m` definiert werden. Da man aber für pq die neuen Variablen kT/q und $[\mathrm{Na}^+]_o/[\mathrm{K}^+]_o$ benötigt, müssen auch diese im Code in der Datei `in_HH.m` implementiert werden.

5.2.3 Grafiken-Vergleich

Nun folgt eine Gegenüberstellung der Grafiken für ein 1:1 und 3:2 Verhältnis. Zuerst das 1:1 Verhältnis:



Nun folgt die Grafik für das 3:2 Verhältnis:



Man kann sehen, dass das 3:2 Verhältnis naturgetreu aussieht im Gegensatz zum 1:1 Verhältnis, wo es scheinbar keine Kontrolle über den Nerv gibt und es zu ständigen Erregungen kommt. Vor allem lässt sich erkennen, dass bei 3:2 die Rückkehr nach dem Aktionspotential in das Ruhepotential geschmeidiger erfolgt. Des Weiteren hat die v-Kurve zu Beginn eine moderate Steigung, bis sie den Schwellenwert erreicht. Danach schnellt die Kurve ganz schnell nach oben und es entsteht ein Aktionspotential.

5.2.4 Code

Anpassung des Codes in `HH.m`.

```
% update conductances
    gNa = gNabar*(m^3)*h;
    gK  = gKbar*(n^4);
    g   = gNa + gK + gLbar;
    gE  = gNa*ENa + gK*EK + gLbar*EL;
    pq  =log((NaK*3*gK)/(2*gNa))*kTq*gK*gNa/(2*gNa+3*gK);
% update v
    v = (v + (dt/C)*(gE + izero(t)-pq))/(1 + (dt/C)*g);
```

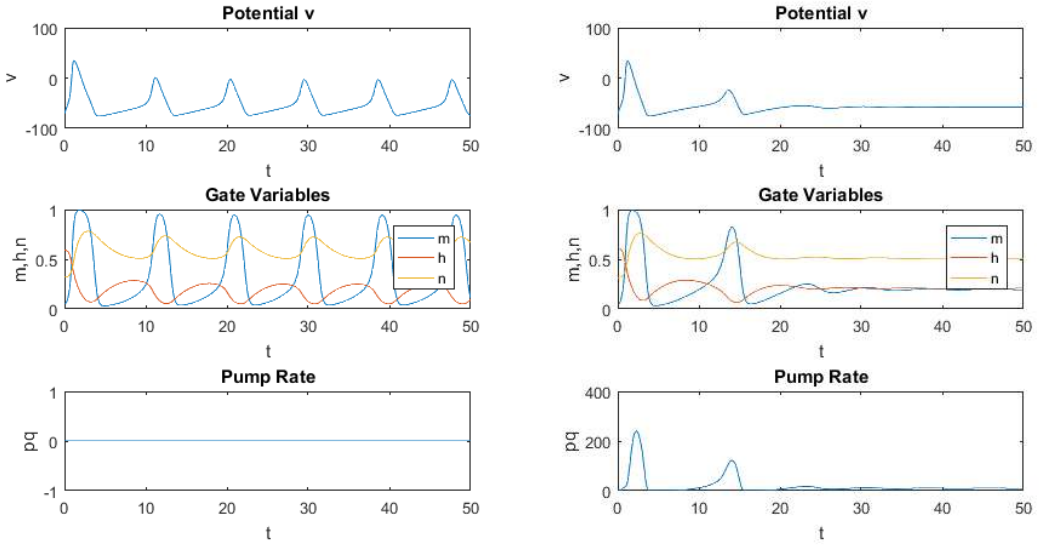
Anpassung des Codes in `in_HH.m`.

```
% voltage prior to t=0 is
vhold = -70;      % mV
    vhold=-70.4090;
% voltage just after t=0
vstart = -55;     % mV
    vstart=vhold;

% (change in v is result of current shock applied at t=0)

% initialize parameters of subsequent current pulse
t1p = 10;         % starting time, ms
t2p = 11;         % stopping time, ms
ip = 15;          % current applied, muA


NaK=145/4;
kTq=25;
```

Damit man die Änderungen und neuen Definitionen im Code verstehen kann, muss beachtet werden, dass NaK für $[\mathrm{Na}^+]_\mathrm{o}/[\mathrm{K}^+]_\mathrm{o}$ und kTq für kT/q steht. Des Weiteren muss `vhold` von -70 auf -70.4090 abgeändert werden.

5.2.5 Gleichgewichte

Wie bereits erkannt wurde, gibt es einen ausschlaggebenden Unterschied zwischen dem 1:1 und 3:2 Verhältnis, was das Gleichgewichtspotential betrifft.

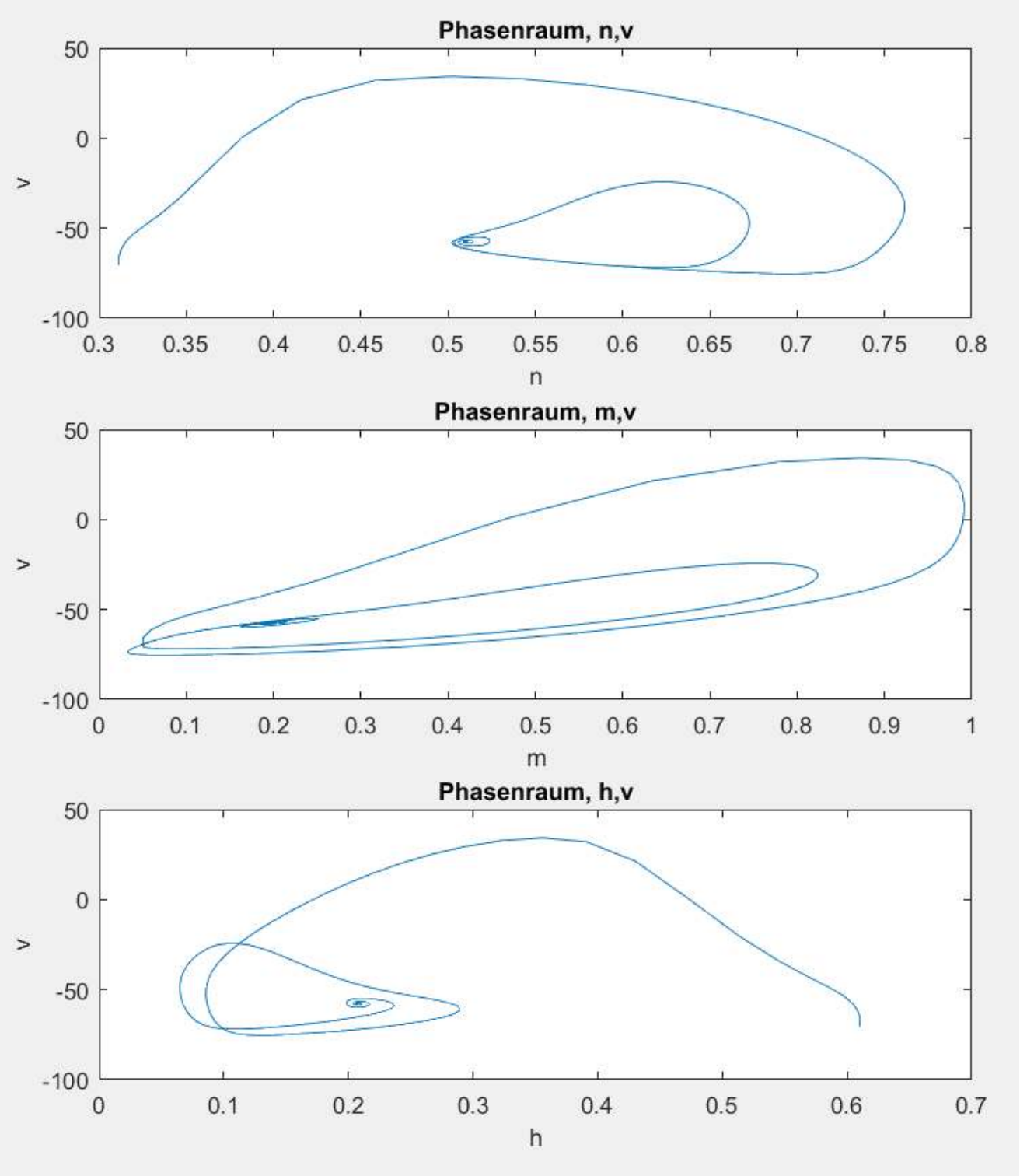


In der linken Grafikspalte wird das 1:1 Verhältnis angenommen, bei dem die Pumprate null ist. Es ist zu erkennen, dass das Membranpotenial nach dem ersten größten Ausschlag und auch später nicht in das Ruhepotential zurückkehrt. Nach dem ersten Aktionspotential werden die weiteren zwar abgeschwächt, aber die abgeschwächten Aktionspotentiale kehren weiterhin periodisch wieder. Diese Membranpotential ist sozusagen instabil.

In der rechten Grafikspalte wird das 3:2 Verhältnis angenommen, bei dem die Pumprate von den abhängigen Leitfähigkeiten der Membran abhängig ist. Das Membranpotential hat nach dem ersten Aktionspotential zwar weitere Anstieg, die aber schlussendlich im Ruhepotential (Gleichgewicht) enden. Der Grund für dieses Phänomen ist die dynamische Pumprate, welche beim 3:2 Verhältnis die Fortsetzung der Aktionspotentiale verhindert, ohne jemals null zu werden. Das ganze ist somit stabil. Durch diese Gegenüberstellung lässt sich erkennen, weshalb Menschen ein 3:2 Verhältnis haben. Wenn wir Menschen ein 1:1 Verhältnis hätten, würde es zu einem Dauerfeuer der Nerven kommen. Infolgedessen ist das 3:2 Verhältnis mit der Pumprate für Menschen überlebenswichtig.

5.2.6 Phasenraum

Der Phasenraum verhaltet sich bei 3:2-Pumprate anders als bei 1:1-Pumprate. Da sich bei einer ungleichen Pumprate das Membranpotential einem konstanten Wert nähert, stellt sich auch im Phasenraum von v gegen alle drei Schleusen ein Gleichgewicht ein, wie es in den folgenden Grafiken sichtbar wird.



6 Räumliche Diffusion

6.1 Das Modell mit Diffusion

6.1.1 Theorie

Wie bereits erwähnt, kann der Impuls, der bisher immer unabhängig von der Ortskomponenten war und somit das ganze Axon gleichzeitig erregt hat, auch nur einen bestimmten räumlichen

Bereich oder Punkt des Axons erregen. Diese abschnittsweise oder punktuelle Erregung führt zu einer Diffusion zwischen den Zellen des ganzen Axons. Das heißt, dass aufgrund der partiellen Erregung ein Potentialunterschied zwischen den gedachten Subzellen entsteht. Jeder Potentialunterschied führt jedoch zu einer Spannung und im Weiteren zu einer Kraft, die stets ausgeglichen werden will. Dadurch erhöht sich die Konzentration der geladenen Teilchen in den weiteren Subzellen, indem Natrium- und Kaliumionen durch die äußere Membran einströmen und die Ionenkonzentration, also das Potential, läuft das Axon entlang bis zu dessen Ende, da sich die Spannung sozusagen immer weiter bewegt.

6.1.2 Code

Im Folgenden werden wir unsere Umsetzung in MATLAB präsentieren und erklären. Die Einheiten der Konstanten und Variablen werden jeweils am Ende der Zeile als Kommentar angegeben. Falls nötig werden die skalaren Werte zu Matrizen umgewandelt.

Zuerst definieren wir die Kapazität der Membran (C), die maximal möglichen Leitwerte für Na^+ und K^+ ($gNabar$, $gKbar$; durch Kanäle) und den Leitwert für Leckströme ($gLbar$; Diffusion von Ionen direkt durch Membran), jeweils pro Flächeneinheit,

```
C = 1;              % muF/cm^2

gNabar = 120;       % (muA/mV)/cm^2
gKbar = 36;         % (muA/mV)/cm^2
gLbar = 0.3;        % (muA/mV)/cm^2
```

die Gleichgewichtspotentiale für Na^+ und K^+ (ENa, EK)

```
ENa = 45;           % mV
EK = -82;           % mV
```

und das Gleichgewichtspotential für sonstige Ionen, die nicht aktiv gepumpt werden (Diffusion durch Membran; z.B. Cl^-)

```
EL = -59;           % mV
```

Wir legen die Anzahl und Dauer der einzelnen Zeitintervalle (Nt, dt)

```
Nt   = 500;
dt   = 0.1;         % ms
tmax = Nt*dt;       % ms
```

und der räumlichen (gedachten) Subzellen (Nx, dx) fest

```
% initialize space step and length of axon
Nx   = 100;
dx   = 0.01;        % cm
xmax = Nx*dx;       % cm
```

Die Gatevariablen (m, h, n) sind wie folgt definiert:

```
m   = alpham(v)./(alpham(v)+betam(v));
h   = alphah(v)./(alphah(v)+betah(v));
n   = alphan(v)./(alphan(v)+betan(v));
gNa = gNabar*m.^3.*h;
gK  = gKbar*n.^4;
```

Mit α und β abhängig vom Potential v

```
alpham = @(v) (v~=-45).*((v+45)/10)./((v==-45)+1-exp(-((v+45)/10)))+1.0*(v==-45);
alphah = @(v) 0.07*exp(-((v+70)/20));
alphan = @(v) 0.1*(v~=-60).*((v+60)/10)./((v==-60)+1-exp(-((v+60)/10)))+1.0*(v==-60);
betam  = @(v) 4*exp(-(v+70)/18);
betah  = @(v) 1./(1 + exp(-(v+40)/10));
betan  = @(v) 0.125*exp(-(v+70)/80);
```

Das Potential vor dem Zeitpunkt t=0:

```
vhold = -70.4090; % mV
v = vhold*ones(Nx+1,1);
```

und direkt nach Start des Programms:

```
vstart = -55;     % mV
vstart = vhold;
```

externe Anregung mit Strom $i0$:

```
i0 = @(t) 40*(x_plot' < 0.1);
```

Als Letztes definieren wir noch die Einheitsmatrix I:

```
I = speye(Nx+1);
```

Um sie später plotten zu können speichern wir die Anfangswerte für die Variablen

```
m_plot(:,1)  = m;
h_plot(:,1)  = h;
n_plot(:,1)  = n;
v_plot(:,1)  = v;
t_plot(1)    = 0;
```

Um die aktuellen Werte für die jeweiligen Variablen zum Zeitpunkt t zu erhalten aktualisieren wir sie innerhalb einer for-Schleife

```
for k=1:Nt

    t = k*dt;

    m = (m + dt*alpham(v))./(1 + dt*(alpham(v) + betam(v)));
    h = (h + dt*alphah(v))./(1 + dt*(alphah(v) + betah(v)));
    n = (n + dt*alphan(v))./(1 + dt*(alphan(v) + betan(v)));


    gNa = gNabar*m.^3.*h;
```

```
    gK  = gKbar*n.^4;
    g   = gNa + gK + gLbar; g = g.*nR;
    gI  = spdiags(g(:),0,Nx+1,Nx+1);
    gE  = gNa*ENa + gK*EK + gLbar*EL; gE = gE.*nR;

    v = (C*I + dt*gI - dt*A) \ (C*v + dt*(gE + i0(t) - pq));
```

und speichern sie zum Plotten.

```
    m_plot(:,k+1)  = m;
    h_plot(:,k+1)  = h;
    n_plot(:,k+1)  = n;
    v_plot(:,k+1)  = v;
    t_plot(:,k+1)  = t;
   end
```

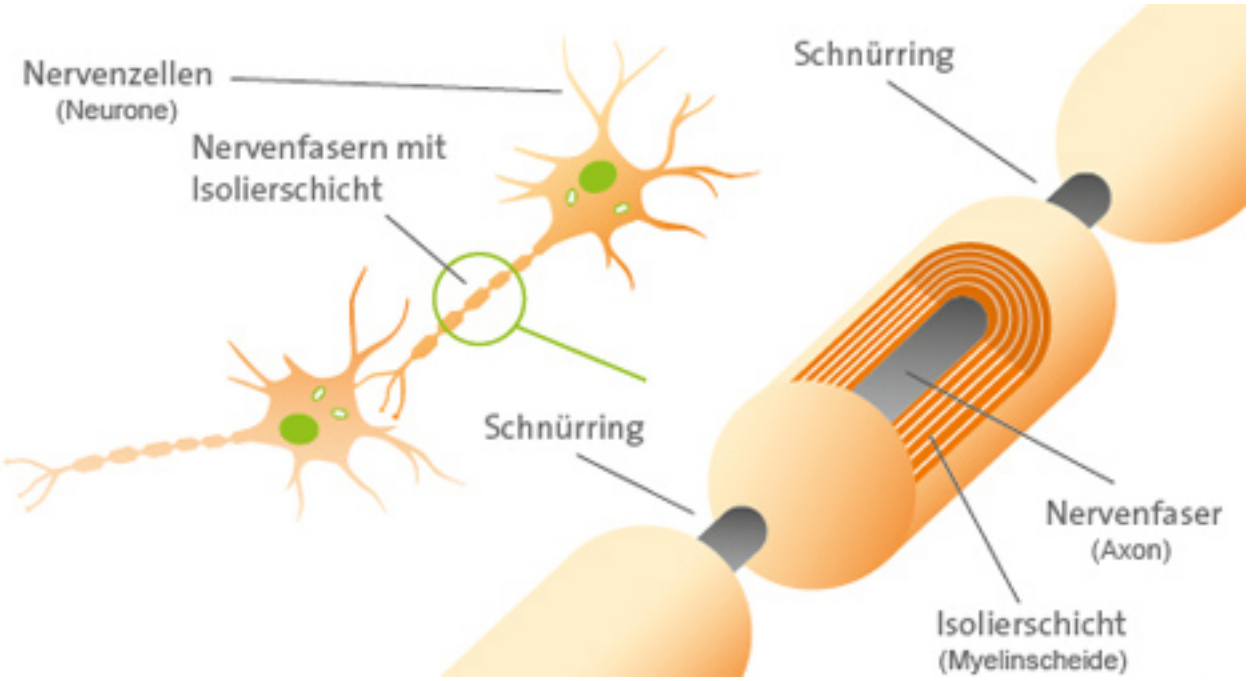
Wir stellen das Potential und die Gatevariablen nun in drei Dimensionen (Zeit t, Länge x, Stärke v, m, h, n) dar und bechriften die Achsen entsprechend.

```
subplot(2,3,1)
waterfall(t_plot,x_plot,v_plot)
xlabel('t')
ylabel('x')
zlabel('v')
title('Potential v')

subplot(2,3,2)
waterfall(t_plot,x_plot,m_plot)
title('Gate Variable m')
xlabel('t')
ylabel('x')
zlabel('m')

subplot(2,3,3)
waterfall(t_plot,x_plot,h_plot)
title('Gate Variable h')
xlabel('t')
ylabel('x')
zlabel('h')

subplot(2,3,5)
waterfall(t_plot,x_plot,n_plot)
title('Gate Variable n')
xlabel('t')
ylabel('x')
zlabel('n')
```

7 Myelinscheide

Die Struktur eines Nerven wurde anfangs bereits erklärt. Manche Nerven verfügen über eine zusätzliche Schicht, die das Axon umhüllt. Diese wird Myelinscheide genannt und führt zu einer weitaus schnelleren Bewegung des Nervimpulses. Die Myelinscheide bedeckt Abschnitte eines Axons. Die Teile des Nerven, die nicht durch eine solche Schicht bedeckt sind, nennt man Ranviersche Schnürringe. In diesen Abschnitten ist die Leitfähigkeit gleich stark wie gewöhnlich,

verglichen mit der bis zu hundertfachen Leitfähigkeit der Abschnitte mit Isolation.



Um das Modell wahrheitsgetreu zu halten, simulieren wir Myelinscheiden durch Nervabschnitte mit weitaus höheren Leitfähigkeiten. Diese Abschnitte werden in regelmäßigen Abständen durch Ranviersche Schnürringe unterbrochen.

Diese Gegebenheiten mathematisch darzustellen ist mithilfe von einigen Modifizierungen zu unseren bisherigen Berechnungen möglich. Anstatt einer konstanten Leitfähigkeit (in unserem Code d genannt) machen wir diese variabel. Wir stellen die Nervenzelle mithilfe von Vektoren dar. Jeder Wert dieses Nx1 Vektors (N ist die Anzahl der Unterteilungen, die wir zur numerischen Lösung des Problems benötigen) steht für eine simulierte Unterteilung der Nervenzelle. An jeder Schnittstelle zwischen zwei solcher Unterteilungen ist eine unterschiedliche Leitfähigkeit möglich. Der Realität entsprechend implementieren wir zwei unterschiedliche Leitfähigkeiten, die eines Abschnittes mit Myelinscheide und die der Ranvierschen Schnürringe.

Daher besteht der Vektor nR (Abkürzung für node of Ranvier = Ranvierscher Schnürring) aus dem Element 1 an jedem Ranvierschen Schnürring und aus dem Element 0 an jeder anderen Stelle. Diese Werte können als logische Argumente $0 = false$ und $1 = true$ aufgefasst werden, was die Implementation auf Matlab erleichtert. Der Vektor mS (MyelinScheide) ist definiert als $1 - nR$, also ein Vektor mit Element 1 an Stellen mit Myelinscheide.

Im Code steht der Wert $a1$ für die Diffusionsrate in Bereichen mit Myelinscheiden, während $a2$ die Diffusionsrate in Ranvierschen Schnürringen angibt.

Die Implementation in Matlab benötigt nur einige zusätzliche Zeilen:

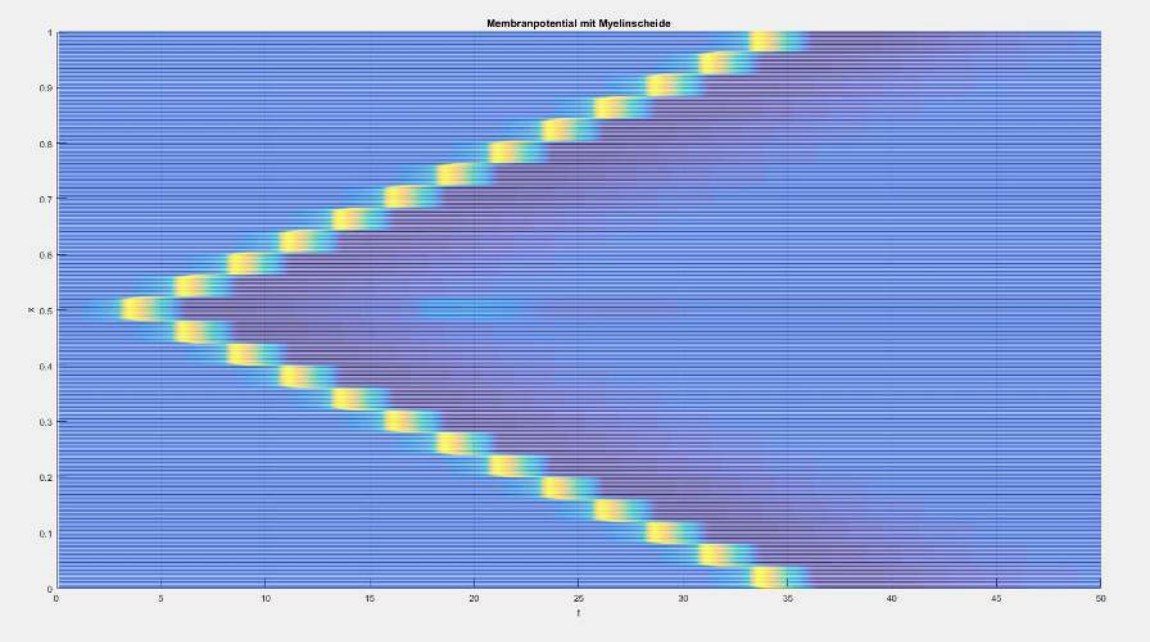
```
nR = zeros(N,1);
nR(1:20:end) = 1;
nR(2:20:end) = 1;
mS = 1-nR;
```

```
a1 = 0.0001;
a2 = 0.01;
d = ((nR(1:N-1)+nR(2:N)) > 0);
d = a1*(d>0) + a2*(d==0);
```

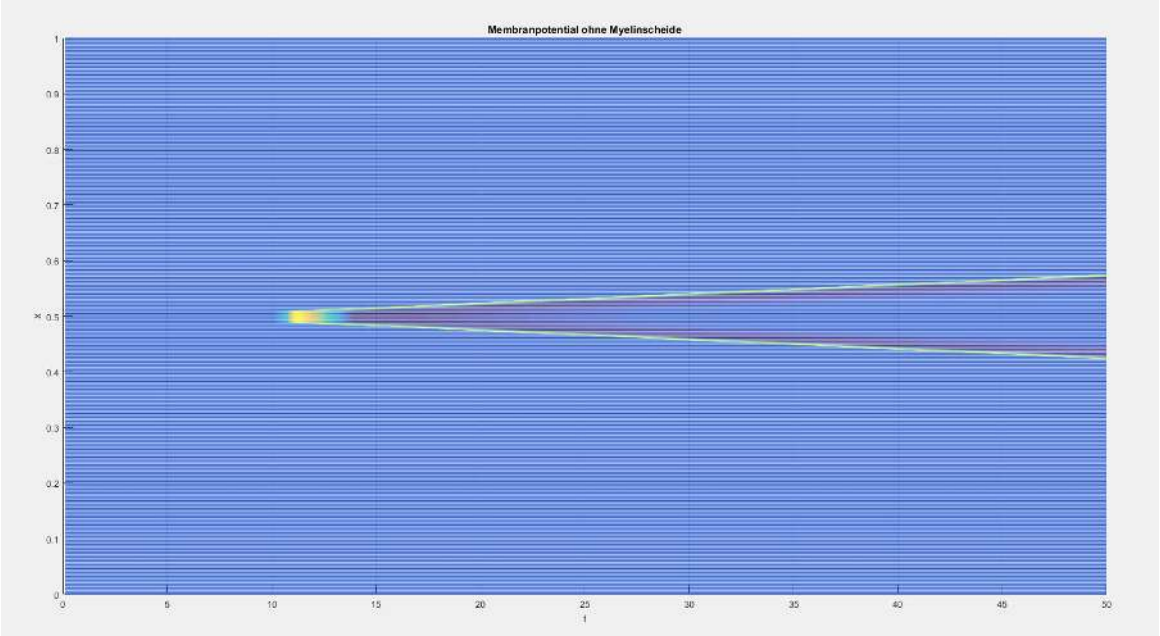
Die ersten drei Zeilen definieren hier den Vektor nR. Es ist natürlich auch möglich, die Stellen, an denen sich die Ranvierschen Schnürringe befinden, zu verändern.

In den letzten beiden Zeilen verwenden wir hier logische Argumente, die als 0 oder 1 ausgewertet werden. Diese Implementation erspart uns die Verwendung einer if-Schleife, die mehr Speicher in Anspruch nimmt.

Die folgenden Graphen vergleichen die räumliche Ausbreitung des Nervimpulses mit Myelinscheide:



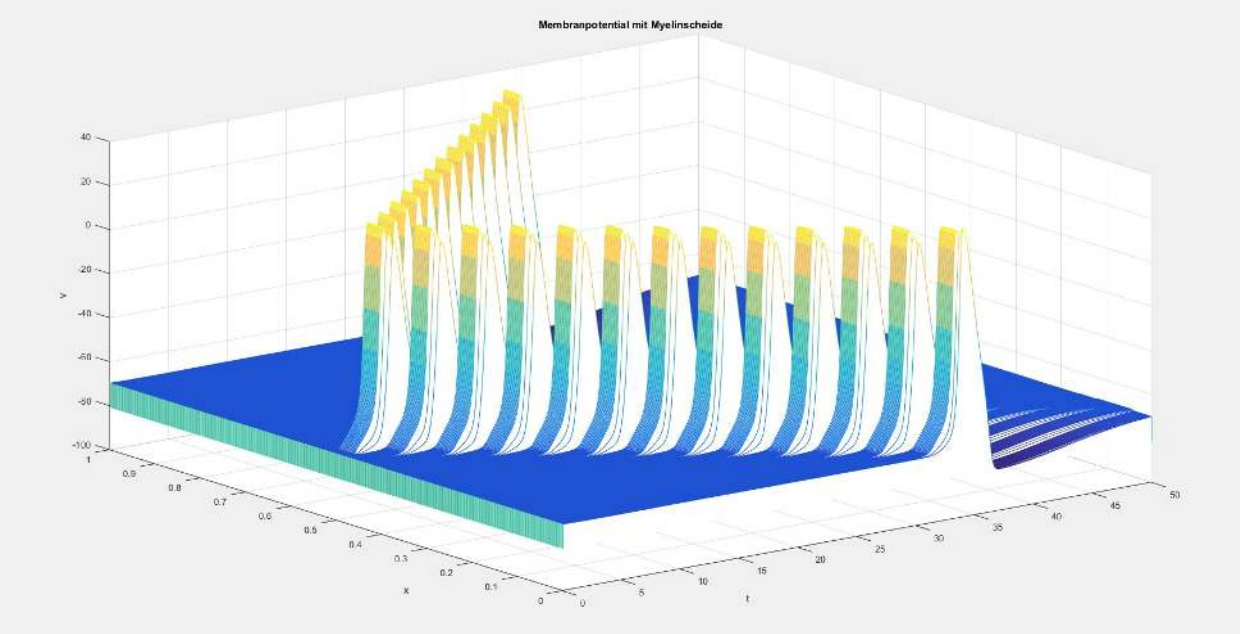
und die ohne Myelinscheide:



Beide Graphen sind gleich skaliert. Die x-Achse zeigt die Zeit, während die y-Achse die räumliche Ausbreitung darstellt. Der Startimpuls beider Grafiken hat dieselbe Stärke. Wie man sieht, braucht der Impuls in der unteren Grafik deutlich mehr Zeit, um sich räumlich auszubreiten. Das liegt an der geringeren Diffusionsrate des Nerven ohne Myelinscheide.

Die dickeren, gelben Stellen des oberen Graphen sind die Stellen mit Myelinscheide, die zu einer deutlich schnelleren Diffusion entlang des Nerven führen.

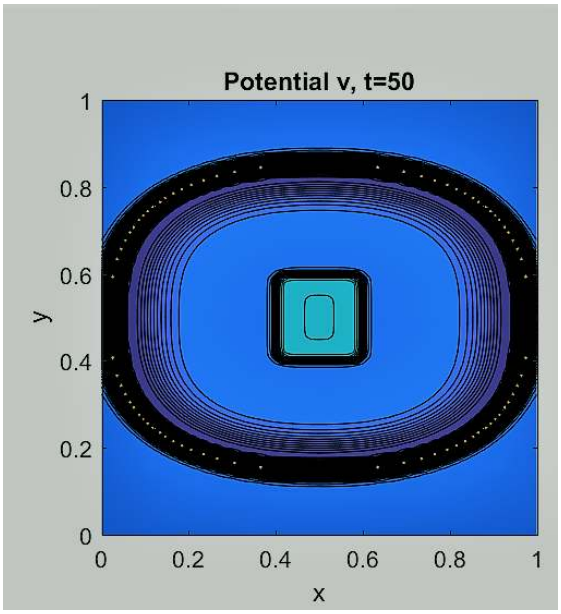
Die folgende Grafik zeigt zusätzlich zur Zeit und der räumlichen Ausbreitung auch das Membranpotential v, also das Aktionspotential:



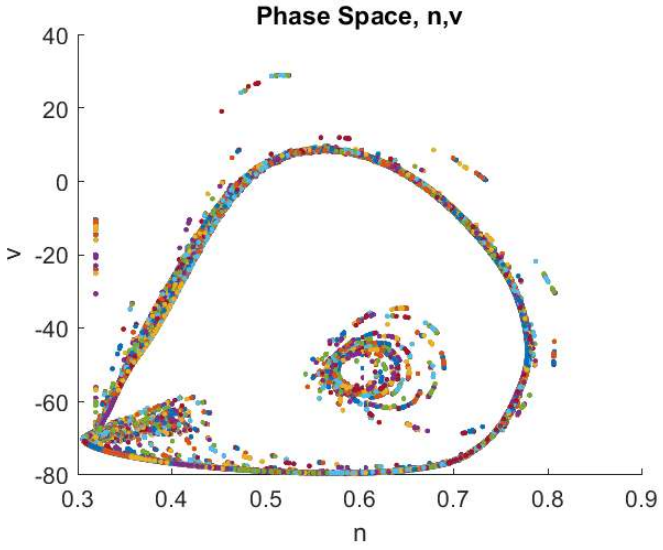
In diesem Graphen sehen wir deutlich die räumliche Ausbreitung wie auch die charakteristische Form des Aktionspotentials.

7.1 Modell mit Diffusion in 2 Dimensionen

Am Ende beschäftigten wir uns mit der Diffusion in 2 Dimensionen. Es stellte sich heraus, dass wir nur einige wenige Veränderungen an unserem Code machen mussten, um zum erwünschten Ergebnis zu kommen. Außerdem erstellten wir, um es grafisch darstellen zu können, eine Animation, die sich mit der Zeit veränderte und nur sich in x-Richtung und y-Richtung ausbreitete. Dass wir die Zeit nicht als Achse aufgetragen haben, hatte den Grund, dass Matlab mit der Grafik überfordert gewesen wäre, da diese Version deutlich mehr Speicherplatz benötigt hätte. Unsere fertige Simulation nach abgelaufener Zeit schaute schließlich folgendermaßen aus.



Oben sieht man das Membranpotential in die 2 Ausbreitungsrichtungen.



Oben sieht man den Phasenraum vom Membranpotential gegen einer Schleuse. Es dauert deutlich länger bis sich das Potential in einen Gleichgewichtszustand begibt, da wir aufrgund der 2-dimensionale Ausbreitung eine größere Interaktionsfläche mit dem Umfeld haben.

8 Fazit

Am Anfang versuchten wir einen möglichst einfachen Code für eine Funktion zu schreiben, die dem Aktionspotential und der dazugehörigen Refraktärzeit entsprechen. Nachdem wir uns anfänglich mit dem Code vertraut machen mussten, konnten wir uns dann der Problemstellung widmen. In dieser Woche konnten wir jedoch die ursprüngliche Problemstellung, das Hodgkin-Huxley-Modell in Matlab zu programmieren, sehr viel weiter ausbauen. Wir haben einerseits die Vereinfachung der Pumprate naturgemäß in eine 3:2-Pumpe umwandeln können und somit ein realistisches Modell für die menschliche Nervenübertragung geschaffen. Andererseits konnten wir die Myelinscheide in den Code einbauen. Das Gleichgewicht spielte ein zentrale Rolle, da

wir mithilfe der 3:2-Pumprate ein stabiles Potential nach einem ausgeführten Nervenimpuls bekamen.