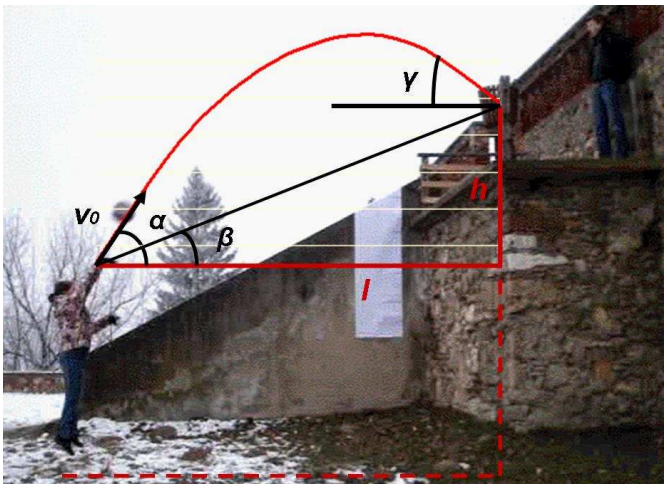
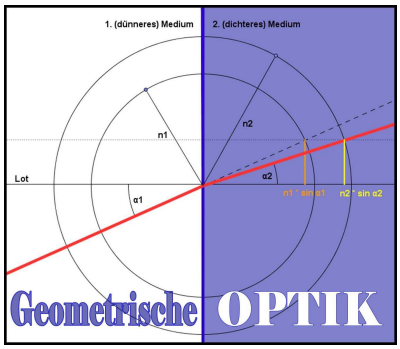
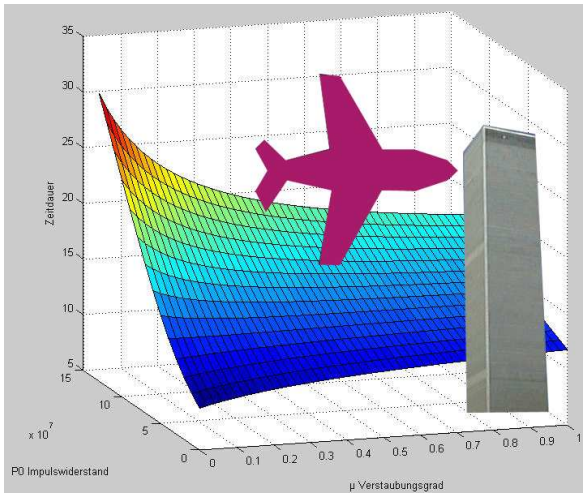
WOCHE DER MODELLIERUNG MIT MATHEMATIK

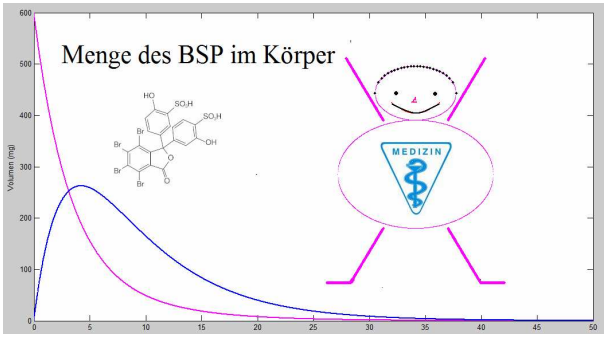
Dokumentationsbroschüre

11. – 17. Jänner 2009









Woche der Modellierung mit Mathematik

Schloss Seggau, 11.01. – 17.01.2009

Weitere Informationen:
http://math.uni-graz.at/modellwoche/2009/

Das Land Steiermark

UNI GRAZ MATHEMATIK

Vorwort

Die Idee zu der in der Steiermark durchgeführten „Modellierungswoche“ für Schüler der 7. und 8. Klasse der AHS wurde schon längere Zeit am Institut für Mathematik und Wissenschaftliches Rechnen der Universität Graz diskutiert. Vorbild waren ähnliche Vorhaben, die bereits in Kaiserslautern, in Bozen und auch in Linz durchgeführt wurden. Mitglieder des Institutes haben bereits Erfahrungen mit ähnlichen Veranstaltungen für Studierende und angehende Wissenschafter. Im Jahre 2005 wurde vom Institut für Mathematik und Wissenschaftliches Rechnen der Karl-Franzens-Universität erstmals eine Modellierungswoche durchgeführt, die bei allen teilnehmenden Schülern und Schülerinnen großen Anklang gefunden hat. Ermutigt durch diesen durchschlagenden Erfolg haben wir seither jedes Jahr wieder eine Modellierungswoche angeboten. Hauptziel der Modellierungswoche ist es, Schüler und Schülerinnen mit einem Aspekt der Mathematik zu befassen, der unserer Meinung nach im Unterricht an den AHS unterrepräsentiert ist: Die Rolle der Mathematik als Werkzeug zum Verständnis der Welt, die uns in Alltag und Wissenschaft umgibt. Während ihrer gesamten Geschichte stand die Mathematik immer in Wechselwirkung mit angewandten Bereichen. Viele mathematische Theorien entstanden in Reaktion auf Anforderungen aus den verschiedensten Anwendungsbereichen. Die Verfügbarkeit immer leistungsfähigerer Computer hat neue Möglichkeiten für die mathematische Behandlung verschiedenster komplexer Probleme eröffnet. Quantitative Resultate statt qualitativer Aussagen sind immer wichtiger und erfordern zu ihrer Bewältigung die mathematische Modellierung komplexer Systeme in interdisziplinärer Zusammenarbeit.

Den an der Modellierungswoche teilnehmenden Schülern und Schülerinnen sollte an Hand sorgfältig ausgewählter Projektaufgaben Gelegenheit gegeben werden, den angewandten Aspekt der Mathematik durch Teamarbeit in Projektgruppen zu erleben. Es wurde versucht, den Teilnehmenden die wesentlichen Phasen eines Modellierungsprozesses nahe zu bringen: Einarbeiten in das Anwendungsgebiet, Wahl der Modellstruktur in Hinblick auf die Aufgabenstellung, Einsatz numerischer Methoden, Interpretation der Ergebnisse, Präsentation der Resultate.

Treibende Kraft für die Realisierung der Modellierungswoche ist Dr. Stephen Keeling, dem hier für seinen großen Einsatz gedankt sei. Besonderer Dank gebührt dem Landesschulrat für Steiermark, und hier insbesondere Frau Landesschulinspektorin Hofrat Mag. Marlies Liebscher. Sie hat die Idee einer gemeinsamen Veranstaltung sofort sehr positiv aufgenommen und tatkräftig unterstützt. Ohne den großen Einsatz der direkten Projektbetreuer, Dr. Sigrid Thaller – Institut für Sportwissenschaft, Dr. Peter Schöpf, Dr. Wolfgang Ring und Dr. Stephen Keeling – alle Institut für Mathematik und Wissenschaftliches Rechnen, und von Herrn Markus Müller, einem Doktoranden unseres Institutes, der die ganze Veranstaltung betreut hat und auch die Gestaltung dieses Berichtes übernommen hat, wäre die Modellierungswoche nicht durchführbar gewesen. Eine wesentliche Rolle im Organisationsteam der Modellierungswoche spielten Gerlinde Krois und Dr. Georg Propst vom Institut für Mathematik und Wissenschaftliches Rechnen. Wir danken der Abteilung für Wissenschaft und Forschung der Landesregierung Steiermark für ihre Subvention.

Wir danken Frau Landesschulinspektorin Hofrat Mag. Marlies Liebscher und Herrn Präsident Wolfgang Erlitz vom Landesschulrat für Steiermark und Vizerektor Martin Polaschek, Vizerektor Ralph Zettl und Dekan Karl Crailsheim von der Universität Graz für die finanzielle Unterstützung.

Schloss Seggau, am 17. 1. 2009

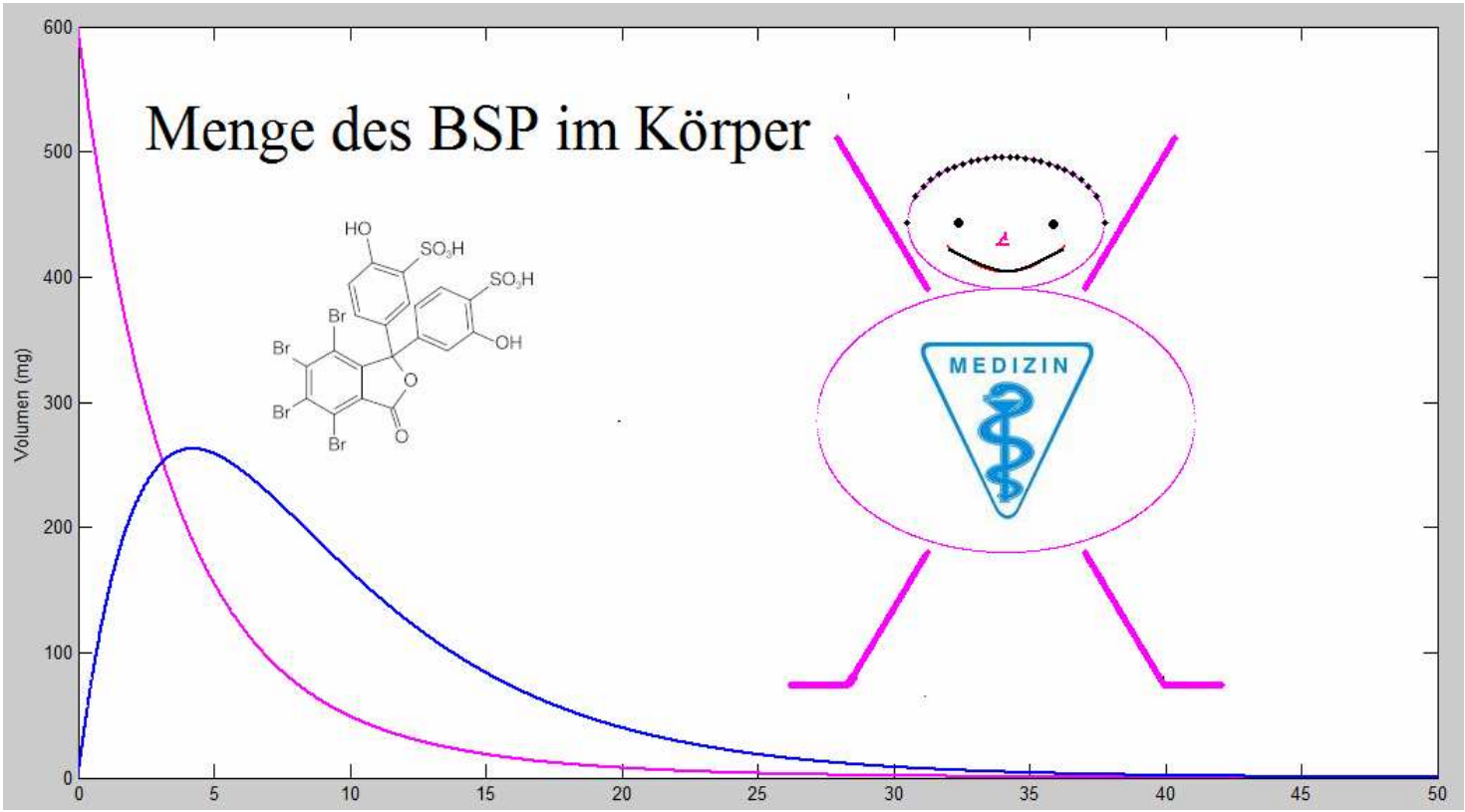
F. Kappel
(Institut für Mathematik und Wissenschaftliches Rechnen)

Bromsulphalein-Test

Projekt: Medizin

Betreuer: Prof. Franz Kappel

Modellierer: Lukas Andritsch, Nasifa Ghazi,
Kerstin Klampfer, Balint Kovacs, Stefan Meighen-Berger,
Alexander Pfleger



Inhaltsverzeichnis

Problemstellung:

Der Bromsulphalein-Test ist ein Leberfunktionstest, bei dem eine Dosis (5mg/kg Körpergewicht) des Farbstoffes Bromsulphalein intravenös verabreicht wird. Dieser Stoff wird in der Leber abgebaut, wodurch man mithilfe der Abbaurate auf die Funktion der Leber schließen kann. Zur Bestimmung der Abbaurate müsste man die Konzentration der Substanz im Lebergewebe alle 3 min messen. Da dies für die Patienten äußerst unangenehm wäre, wird zur Ermittlung der Abbaurate die Konzentration von BSP in Blutproben unmittelbar nach der Verabreichung zum Zeitpunkt t = 0 und dann in regelmäßigen Abständen gemessen.

Das Ziel unseres Projekts ist daher, ein einfaches mathematisches Modell zu entwickeln. Anhand realer Messdaten soll die Problemlösung durch das Modell dargestellt werden.

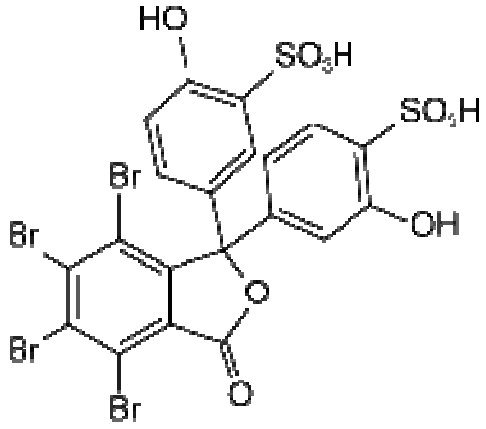
Einführung:

Bromsulphalein:

Bromsulfophthalein auch Bromsulphalein oder BSP ist ein Diagnostikum zur Überprüfung der Leberfunktion. Es ist ein bromiertes Sulfonsäurederivat des Phenolphthaleins.

Summenformel: $C_{20}H_{10}Br_4O_{10}S_2$

Strukturformel:



BSP ist in alkalischen, also basischen Lösungen violett, in sauren Lösungen farblos und nur in der Leber in nennenswerten Mengen abgebaut wird.

BSP wird im Bromsulphalein-Test eingesetzt, um die exkretorische Leberfunktion zu untersuchen, der 1938 etabliert wurde.

Bromsulphalein-Test:

Um 13 Uhr trafen wir in unserem Seminarraum zusammen und diskutierten über unsere Problemstellung. Dabei wurde uns erklärt, dass der Bromsulphalein-Test, unser Projektthema, ein Leberfunktionstest ist, bei dem eine variierbare Dosis (5mg/kg Körpergewicht) Bromsulphalein intravenös verabreicht wird.

Bromsulphalein selbst ist lediglich ein Farbstoff, der nur über die Leber in die Galle in nennenswerten Mengen ausgeschieden wird. Die Zerfallsrate von Bromsulphalein gibt an, wie gut bzw. schlecht die Ausscheidungsfunktion der Leber ist. Eine Konzentration von bis zu 0,4 mg pro 100 ml bzw. bis zu 4% der ursprünglich injizierten Menge nach 45 Minuten ist normal.

Anwendungen fand der Bromsulphalein-Test im 1. Irak-Krieg, in dem urangehärtete Granaten zum Einsatz kamen. Beim Aufprall der Granaten auf Panzer etc. zerstäubte das Uran, verbreitete sich in der Luft, wurde eingeatmet und so auch die Leber der Soldaten geschädigt. Außerdem wurde auch die Dialyse erwähnt, die mit Heparin, einem Stoff aus Schweinemägen, durchgeführt wird. Man sucht nämlich eine alternative Substanz, welche zum Beispiel Zitrat wäre, das in der Leber vorkommt.

Der Test wird wegen der damit verbundenen Gefahren kaum noch angewendet und nur aus historischen Gründen erwähnt. Wiederholt werden auch anaphylaktische Schocks beschrieben. Die paravenöse Injektion führt zu schweren Gewebsnekrosen. Vorteilhaft sind die leichte Durchführung und der Preis, aber der Farbstoff BSP ist heute nur mehr schwer erhältlich.

Heute wird der Test hauptsächlich in Tierversuchen und der Veterinärmedizin angewandt.

Zu den Tests, mit deren Hilfe die Ausscheidungsfunktion der Leber überprüft werden kann, gehört die Bestimmung des Gallenfarbstoffes „Bilirubin" im Blut. Bilirubin ist ein Abbauprodukt des roten Blutfarbstoffes Hämoglobin, das normalerweise über die Galle in den Darm ausgeschieden wird. Dazu ist ein regelrechtes Funktionieren der Ausscheidungsleistung der Leber erforderlich. Ein Ansteigen des Bilirubins im Blut zeigt demnach eine Störung an, die entweder in der Leber selbst oder in den galleabführenden Wegen liegt.

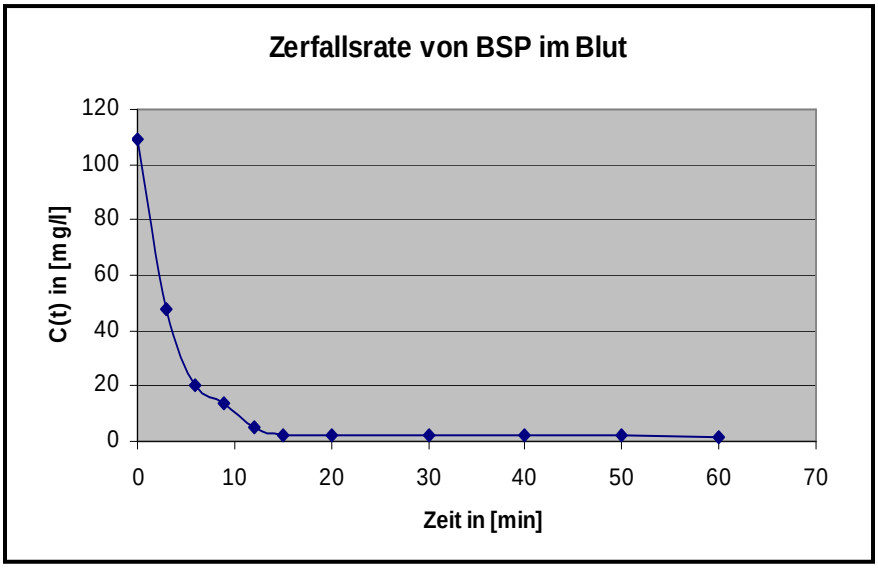
Problem:

Zur Bestimmung der Zerfallsrate könnte man einerseits die Bromsulphaleinkonzentration im Lebergewebe in einem regelmäßigen Abstand von 3 Minuten nach der Injektion messen, was aber für den Patienten sehr unangenehm wäre und daher undurchführbar ist. Deshalb wird die Konzentration in Blutproben vom Zeitpunkt 0 in 3 Minuten-, später in 10 Minutenabständen gemessen.

Das Problem der Aufgabe ist daher, anhand der Konzentrationsmessungen im Blut auf die Zerfallsrate von Bromsulphalein in der Leber zu kommen.

Die Messwerte des Patienten:

$C_B(t)$ in [mg/l]	Zeit in [min]
109	0
48	3
20	6
14	9
5	12
2,4	15
2	20
2	30
2	40
2	50
1,80	60



Die Formel von Du Bois:

Der deutsche Physiologe Du Bois stellte eine empirische Formel auf, also eine Formel, die lediglich auf Beobachtungen beruht, um die Körperoberfläche eines Menschen zu berechnen.

$$s = 0{,}007184 \cdot w^{0{,}425} \cdot h^{0.725}$$

Diese Formel ist unerlässlich zur Bestimmung der Konzentration des Bromsulphaleins im Blut, da wir für die Berechnung der Konzentration das Gesamtblutvolumen benötigen. Dieses wiederum ist nur durch das Wissen der Körperoberfläche zu erhalten.

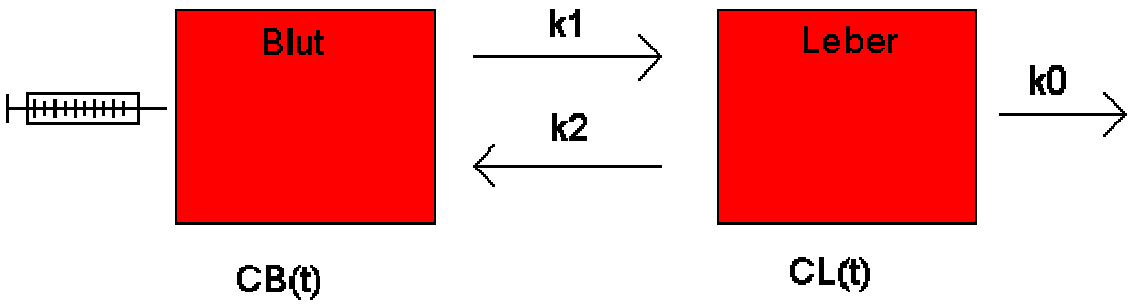
$$V = 3{,}290 \cdot s - 1{,}229$$

s = Körperoberfläche in m^2
w = Gewicht in kg
h = Körpergröße in cm
V = Volumen in l

Kompartment-Modell:

Ein Multi-Kompartment-Modell ist ein mathematisches Modell zur Beschreibung der Art und Weise, wie bestimmte Materialien oder Energien in Abteilungen eines Systems übertragen werden. Die Multi-Kompartment-Modellierung ist die Annahme von mehreren Behauptungen, dass Systeme in physischer Existenz mathematisch modelliert werden können. Multi-Kompartment-Modelle werden in vielen Bereichen einschließlich der Pharmakokinetik, Biomedizin, Epidemiologie, Technik, Physik, etc. verwendet.

Kompartment-Modell



$C_B(t)$= Die Konzentration des Bromsulphaleins im Blut

$C_L(t)$= Die Konzentration des Bromsulphaleins in der Leber

k_1= Die Rate des Bromsulphaleins vom Blut in die Leber

k_2= Die Rate des Bromsulphaleins von der Leber zurück ins Blut

k_0= Die Rate des Bromsulphaleins, die in der Leber abgebaut wird.

Mit diesem Kompartment-Modell konnten wir unsere Problemstellung veranschaulichen und uns unsere kommenden Rechenschritte besser vorstellen. Durch dieses Schema sieht man, wie durch die Spritze eine Dosis Bromsulphalein injiziert wird und so in das Blut gelangt. Von dort weg wird der Farbstoff weiter in die Leber geleitet, um dort letztendlich abgebaut zu werden.
Je mehr Bromsulphalein in der Leber zerfällt, desto besser ist die Ausscheidungsfunktion des Organs, wodurch man anhand der Abbaurate eine Diagnose über den Gesundheitszustand der Testperson stellen kann.

Mathematische Aspekte des Modells:

Parameteridentifizierung:

Aufgrund des bereits beschriebenen Kompartment-Modells konnten wir folgende zwei Formeln erstellen, die die Veränderung der Bromsulphaleinmenge im Blut (m_B) bzw. in der Leber (m_L) zum Zeitpunkt t beschreiben:

$$\frac{d}{dt} m_B(t) = -k_1 m_B(t) + k_2 m_L(t)$$

Diese Formel stellt die Veränderung des Bromsulphaleins im Blut dar: Einerseits fließt der Farbstoff mit dem Faktor k_1 vom Blut in die Leber, andererseits gelangt auch wieder ein Teil der Menge des Stoffes mit dem Faktor k_2 multipliziert aus der Leber zurück ins Blut, da er aufgrund der großen Konzentration nicht abgebaut werden konnte.

$$\frac{d}{dt} m_L(t) = k_1 m_B(t) - (k_0 + k_2) m_L(t)$$

Auch hier wird die Veränderung gezeigt, diesmal in der Leber: Bromsulphalein gelangt mit dem Faktor k_1 aus dem Blut, doch wird es hier abgebaut (k_0) beziehungsweise (bei gesunder Leber in geringfügigen Mengen) mit dem Faktor k_2 in den Blutkreislauf zurückgeschickt.

Da wir nur die Konzentration C_B und somit auch die Masse des Farbstoffes im Blut kennen, ersetzten wir nach Umformungen m_L durch m_B, wodurch wir eine Funktion in den Parametern k_0, k_1 und k_2 erhalten.

$$m_B''(t) + m_B'(t) \cdot (k_0 + k_1 + k_2) + m_B(t) \cdot (k_0 \cdot k_1) = 0$$

Da es sich bei m_B um einen biologischen Abbau handelt, können wir davon ausgehen, dass es durch eine Exponentialfunktion beschrieben wird:

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

Durch das Einsetzen erhalten wir eine quadratische Gleichung und somit zwei Lösungen für λ.

$$\lambda_{1/2} = \frac{-k_1 - k_0 - k_2 \pm \sqrt{(k_0 + k_1 + k_2)^2 - 4k_0k_1}}{2}$$

Beide λ nehmen negativ reelle Werte an, was Voraussetzung für den Abbau ist. Beide Eigenschaften sollen im Folgenden gezeigt werden:
Behauptung 1: $\lambda_{1,2}<0$:
Beweis: Alle k_1, k_2 und k_0 Werte sind positiv, da sie Zuflüsse bzw. Abbau eines Stoffes beschreiben. Wir betrachten daher nur den Zähler des Bruches. Ist dieser stets negativ, so ist dies auch für $\lambda_{1,2}$ der Fall. Zieht man die Wurzel ab, so besteht die rechte Seite der Gleichung aus ausschließlich negativen Werten, λ_2 ist also negativ. Es genügt daher zu zeigen, dass auch λ_1 negativ ist, was gleichbedeutend mit folgendem Ausdruck ist:

$$(k_0 + k_1 + k_2) > \sqrt{(k_0 + k_1 + k_2)^2 - 4k_0k_1}$$

$$\Leftrightarrow -4k_1k_0 < 0$$

.
Dies ist offensichtlich richtig, womit Behauptung 1 gezeigt ist.

Behauptung 2: $\lambda_2 \in$ R
Beweis: Die Zahlen sind genau dann reell, wenn die Diskriminante größer oder gleich 0 ist.
z.z:

$$(k_0 + k_1 + k_2)^2 - 4k_0k_1 > 0$$

$$\Leftrightarrow {k_0}^2 + {k_1}^2 + {k_2}^2 - 2k_0k_1 + 2k_1k_2 + 2k_0k_2 > 0$$

$$\Leftrightarrow (k_0 - k_1)^2 + {k_2}^2 + 2k_1k_2 + 2k_0k_2 > 0$$

Da alle Werte mindestens 0 sind, und keines der k_0, k_1, k_2 selbst den Wert 0 annehmen kann ist auch die 2.Behauptung bewiesen.

Somit kann die Funktion $m_B(t)$ durch die Summe von zwei Exponentialfunktionen dargestellt werden, mithilfe der Exponenten λ_1 und λ_2 und zweier Koeffizienten α und β:

$$m_B(t) = \alpha \cdot e^{\lambda_1 \cdot t} + \beta \cdot e^{\lambda_2 \cdot t}$$

Die nächste Aufgabe besteht darin, die beiden Koeffizienten α und β zu bestimmen, um die Funktion auf die drei Parameter k und die Zeit t zu beschränken. Dazu verwenden wir die Werte für $m_B(0)$ und $m'_B(0)$, die sich aus den beiden Differentialgleichungen ergeben (da das Bromsulphalein bei t=0 gerade injiziert wurde, befindet es sich ausschließlich im Blut: $m_B(0)=D$, $m_L(0)=0$).

Wir erhalten:

$$m_B(0) = D = \alpha + \beta$$
$$m'_B(0) = -k_1 D = \lambda_1 \alpha + \lambda_2 \beta$$

Aus diesem Gleichungssystem können wir beide Koeffizienten errechnen, da alle weiteren Größen bereits bekannt sind: Durch Einsetzen erhält man die gesuchte Funktion:

$$m_B(t) = \frac{D \cdot (\lambda_2 + k_1) \cdot e^{\lambda_1 \cdot t}}{\lambda_2 - \lambda_1} + \frac{D \cdot (\lambda_1 + k_1) \cdot e^{\lambda_2 \cdot t}}{\lambda_1 - \lambda_2}$$

Anzumerken ist noch, dass die beiden Parameter α und β immer positiv sind, da die Menge des Bromsulphalein im Blut logischerweise nicht negativ sein kann.
Behauptung: α >0
Beweis: Da der Nenner des ersten Terms negativ ist, müssen wir nachweisen, dass auch der Zähler negativ ist:

$$\lambda_2 + k_1 < 0$$
$$\Leftrightarrow -k_0 - k_1 - k_2 - \sqrt{(k_0 + k_1 + k_2)^2 - 4k_0 k_1} < -2k_1$$
$$\Leftrightarrow -1 < 1$$

Analog muss nachgewiesen werden, dass β positiv ist:
Beweis: Der Nenner des zweiten Terms ist positiv, folglich muss auch der Zähler positiv sein:

$$\lambda_1 + k_1 > 0$$
$$\Leftrightarrow -k_0 - k_1 - k_2 + \sqrt{(k_0 + k_1 + k_2)^2 - 4k_0 k_1} > -2k_1$$
$$\Leftrightarrow 1 > -1$$

Da wir die Menge im Blut durch die drei Parameter ersetzt haben, ergibt sich sofort die Konzentration im Blut und zwar aufgrund der Division durch das Blutvolumen V_B:

$$C(t) = \frac{m_B(t)}{V_B}$$

Damit haben wir das Messbare, die Konzentration des Bromsulphaleins im Blut, auf unsere 3 Parameter und zusätzlich das Blutvolumen reduziert. Es ist uns nun möglich, nur aufgrund der Konzentration des Farbstoffes im Blut auf die Abbaurate der Leber zu schließen und damit eine Verbindung zu den Messwerten herzustellen.

<u>Annäherung an die Messdaten</u>:

Als ersten Schritt wollen wir eine ungefähre Einschätzung für die drei Parameter erhalten. Hierfür helfen uns die Messdaten des Patienten, da diese dem Wert C_B zum Zeitpunkt t entsprechen:

$$C_B(t) = \alpha \cdot e^{\lambda_1 t} + \beta \cdot e^{\lambda_2 \cdot t}$$

Da wir vorerst noch nicht die exakten Werte erhalten wollen, können wir uns auf den stärker fallenden Teil der Funktion $C_B(t)$ beschränken, der den Abnahme der Kurve um ein Vielfaches stärker beeinflusst als der andere Teil der Summe mit dem größeren Exponenten λ_1. Weiters kennen wir den Startwert der Funktion zum Zeitpunkt t=0, da die Konzentration im Blut des Patienten 109 annimmt. Wir definieren nun deshalb die vereinfachte Exponentialfunktion:

$$\overset{*}{C}_B(t) := 109 \cdot e^{\lambda_2 \cdot t}$$

Betrachtet man die Messung nach drei Minuten, so erhält man durch Logarithmieren:

$$\frac{-k_0 - k_1 - k_2 - \sqrt{(k_0 + k_1 + k_2)^2 - 4k_0k_1}}{2} = \frac{\ln 48 - \ln 109}{-3}$$

Wir vereinfachen die Gleichung ein weiteres Mal, indem wir den Faktor $-4k_0k_1=0$ setzten. Damit haben wir den Abnahme im ersten Vereinfachungsschritt zwar verlangsamt, jetzt verkleinern wir damit den Exponenten λ_2, womit wir den Abnahme wiederum verschnellern. Nun können wir die Wurzel ziehen, dadurch ergibt sich für die Summe der drei Parameter:

$$k_0 + k_1 + k_2 \approx 0{,}2734$$

Wiederholt man diesen Vorgang bei mehreren Messwerten, so erhält man ähnliche Werte für die Summe der drei Parameter, womit sich eine grobe Abschätzung für die Summe ergibt:

$$k_0 + k_1 + k_2 = 0{,}3 \pm 0{,}1$$

Die große Streuung des Wertes der Summe folgt logischerweise aus der Vereinfachung der Funktion.

Um die Werte genau bestimmen zu können, benötigen wir nun eine Funktion C(t), die sich bestmöglich an die realen, gegebenen Messdaten anpasst. Diese nennen wir $\tilde{J}(t)$. Die effektivste Möglichkeit eine Kurve mit bestimmten Punkten anzugleichen besteht darin, den Abstand zwischen Kurve und Punkt zu messen und diesen zu minimieren. Die Messpunkte definierten wir als Werte der Menge X_i, wobei die Messwerte vom ersten bis zum letzten durchlaufend nummeriert werden:

$$\tilde{J}(k_0, k_1, k_2) = \sum_{i=1}^{N} \left| C_B(t, k_1, k_2, k_o) - X_i \right|$$

Wir müssen die Summe der Beträge der Differenzen minimieren, da der Abstand eines Punktes zu einer Kurve stets positiv ist. Da die Betragsfunktion nicht für alle Werte differenzierbar ist, erstellen wir eine zweite Funktion J(t), die die Quadrate summiert:

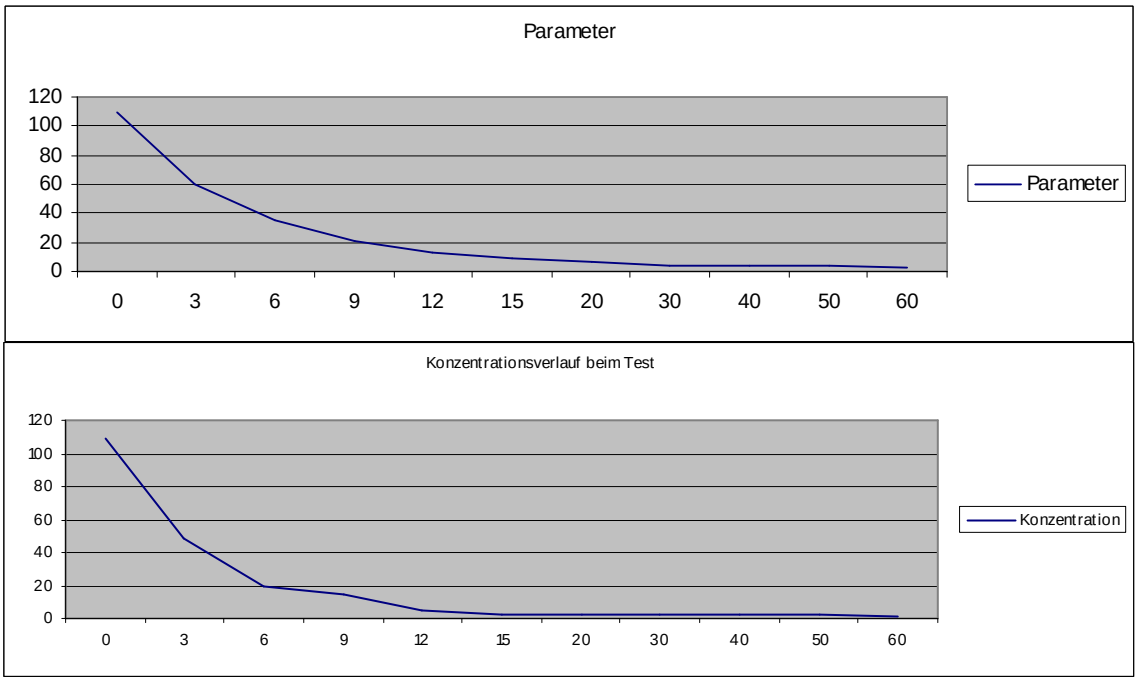
$$J(k_1, k_2, k_0) = \sum_{i=1}^{N} (C_B(t, k_1, k_2, k_0) - X_i)^2$$

Im Idealfall entspricht die Kurve exakt den Messdaten, dann wäre die Summe optimal, also 0.

Das Modell

Erste Schritte: Excel

Unsere erste Idee der graphischen Darstellung verwirklichten wir in Excel. Wir wollten uns einen Überblick von der Formel und möglichen Problemen schaffen, bevor wir mit der schwierigeren, wenn auch viel genaueren, Programmierung in MATLAB anfingen. Schnell wurde aber klar, dass Excel bei weitem nicht ausreicht, um ein Modell für den BSP-Test zu erstellen, da es nicht Befehle wie „fminsearch“ oder „If - Schleifen“ enthält.



MatLab

Da wir die riesigen anfallenden Datenmengen nicht zu Fuß ausrechnen wollten, suchten wir ein Programm, das einfach zu bedienen ist und auch mehrere Funktionen in Diagrammen darstellen kann. Wir entschieden uns letztendlich dann für MATLAB. Um uns mit diesem Programm vertraut zu machen, bekamen wir eine kurze Einführung zu MATLAB, einerseits von Herrn Univ.Prof. Franz Kappel, andererseits durch das Lesen der programmeigenen Demofunktionen. Wir versuchten erstmal nur den Graphen der echten Messwerte darzustellen; und siehe da, es funktionierte auf Anhieb und wir bekamen für jeden Wert einen Punkt in das Diagramm eingezeichnet. Um mit MATLAB mehr in Einklang zu kommen, experimentierten wir noch ein bisschen mit dem Graphen herum und änderten Farben und Einheiten.

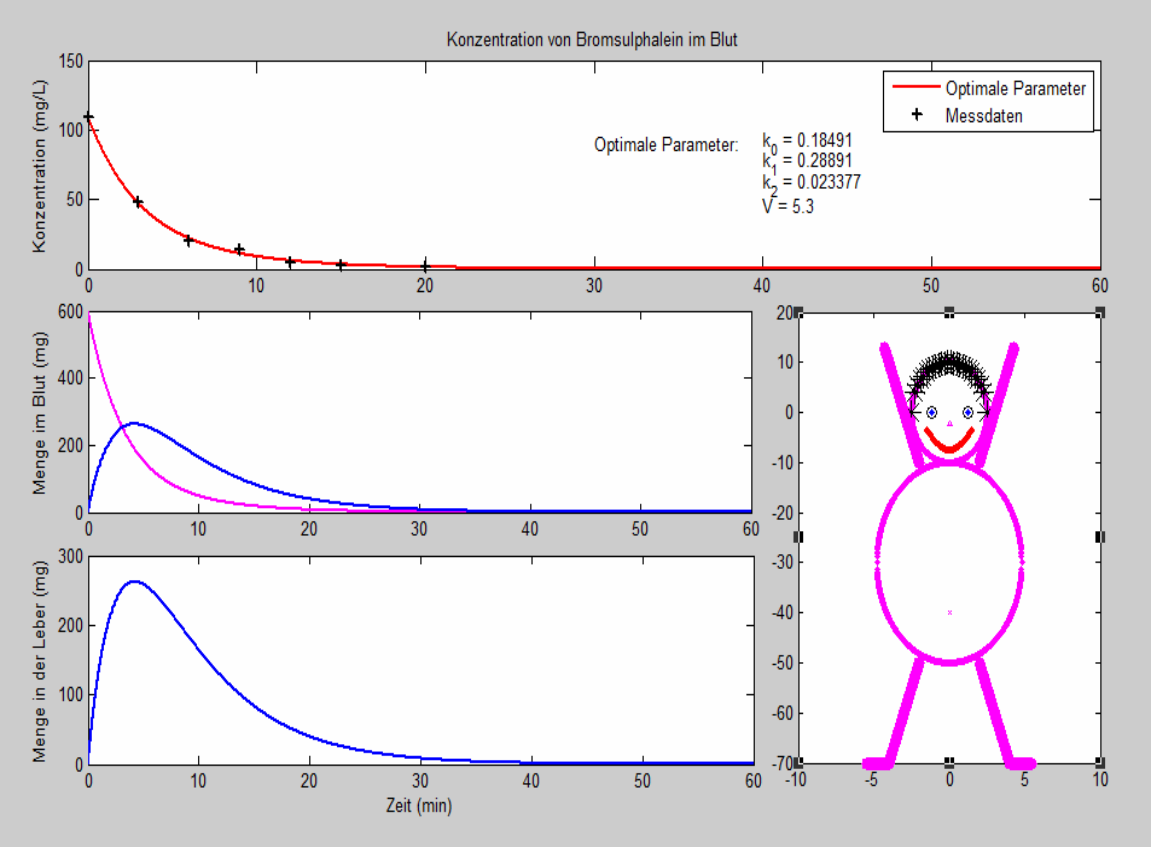
Unser nächster Schritt war der wichtigste im ganzen Projekt, denn hier begann erst das eigentliche Modellieren. Wir fassten alle Formeln zusammen und versuchten ein zusammenhängendes Modell zu erstellen, in dem man jeweils Graphen des Bromsulphaleingehalts in der Leber und im Blut mit Abhängigkeit von der Zeit darstellt.
Unsere ersten Schritte waren noch nicht so schwer, und es dauerte auch nicht außergewöhnlich lange, bis wir die ersten Graphen auf dem Bildschirm hatten, doch zu diesem Zeitpunkt fingen unsere Probleme erst an.

Zunächst beschäftigen uns wir uns einmal mit dem Programm MATLAB selbst. MATLAB ist eine plattformunabhängige Software, was den Linuxuser unter uns sehr freute. MATLAB ist hauptsächlich zum Arbeiten mit Matrizen erzeugt worden. Der Unterschied zu anderen Programmen dieser Größenordnung ist, dass MATLAB fast ausschließlich mit numerischen Lösungen arbeitet.

Die gesunde Testperson:

Aus dem Quelltext:

```
function kmin = American_French_Fry03(x0, f)
Xi = [109, 48, 20, 14, 5, 2.4, 2]; %Messwerte
T = [0, 3, 6, 9, 12, 15, 20]; %Messzeitpunkte
D= 600; %Dosis
kmin(1) = 0.2889 %BSP-Abbaurate in der Leber
kmin(2) = 0.0233 %BSP-Zuflussrate in die Leber
kmin(3) = 0.1849 %BSP-Rückflussrate von der Leber ins Blut
V = 5.3 %Blutvolumen
```



1. Diagramm:
Die vorhandenen Messwerte werden als Kreuze in das Diagramm eingezeichnet und der rote Graph repräsentiert die mit den Messwerten errechnete Kurve für die BSP-Konzentration ($C_B(t)$) im Blut.

2. Diagramm:
Der rote Graph zeigt wie viel BSP zu jedem Zeitpunkt im Blut vorhanden ist. Als Vergleich dazu entspricht der blaue Graph den Werten in der Leber zum gleichen Zeitpunkt.

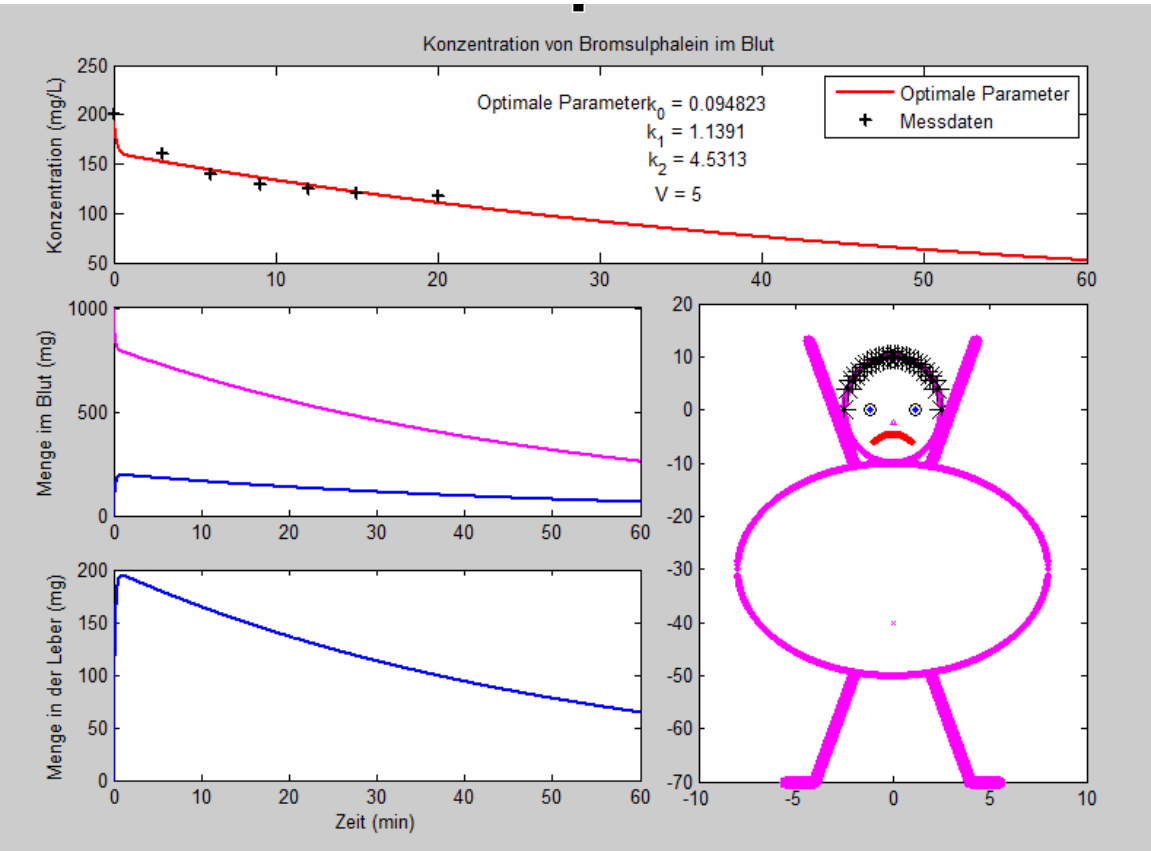
3. Diagramm:
Die BSP-Werte in der Leber werden gezeigt.

4. Diagramm:
Das Männchen lächelt, wenn es gesund ist und weint, wenn es krank ist. Außerdem verändert sich die Breite seines Körpers in Bezug auf das Gewicht.

Die kranke Testperson :
Aus dem Quelltext:

```
function kmin = American_French_Fry03(x0, f)
Xi = [200, 160, 140, 130, 125, 120, 117];%Messwerte
T = [0, 3, 6, 9, 12, 15, 20];%Messzeitpunkte
```

```
D= 1000; %Dosis
kmin(1) = 0.0948 %BSP-Abbaurate in der Leber
kmin(2) = 4.5313 %BSP-Zuflussrate in die Leber
kmin(3) = 1.1391 %BSP-Rückflussrate von der Leber ins Blut
V = 5 %Blutvolumen
```



Quelltext:

```
function c= conc1 %BSP-Konzentration
  k0= kmin(1); %BSP-Abbaurate in der Leber
  k1=kmin(2); %BSP-Zuflussrate in die Leber
  k2=kmin(3); %BSP-Rückflussrate von der Leber ins Blut
  lambda1= (-k1-k2-k0+sqrt((k1+k2+k0)^2-4*k0*k1))/2;
  lambda2= (-k1-k2-k0-sqrt((k1+k2+k0)^2-4*k0*k1))/2;
  beta = D*(lambda1+k1)/(lambda1-lambda2);
  alpha = D*(lambda2+k1)/(lambda2-lambda1);
  mB = alpha*exp(lambda1*tf)+beta*exp(lambda2*tf); %BSP-Menge im Blut
  mL = (alpha*exp(lambda1*tf)*(lambda1-k1)+beta*exp(lambda2*tf)*(lambda2-
k1))/k2; %BSP-Menge in der Leber
  c = mB/V; %BSP-Konzentration
end
tf = 0:0.01:60; %Zeitwerte für den roten Graphen im 1. Diagramm
subplot(4,1,1); plot(tf, c, 'r','LineWidth',2); %1. Diagramm – roter Graph
hold on
plot(T, Xi, 'k +', 'MarkerSize',6, 'LineWidth', 2); %1. Diagramm - Kreuze
subplot(4,1,2); plot(tf, mB, 'm', 'LineWidth', 2) %2. Diagramm – rosa Graph
hold on
subplot(4,1,2); plot(tf, mL1, 'LineWidth', 2) %2. Diagramm – blauer Graph
ylabel ('Menge im Blut (mg)')
hold on
subplot(4,1,3); plot(tf, mL1, 'LineWidth', 2) %3. Diagramm – blauer Graph
```

<u>Um die Werte für k_1, k_2 und k_0 zu finden verwendeten wir zwei Optimierungsverfahren</u>:

1. Mit der Summe der Quadrate der Abstände der Messwerte zur errechneten Kurve:

```
kmin = fminsearch(@Jqu, x0); %Findet ein Minimum
function y = Jqu(x)
C=conc(x);
y= sum((C-Xi).^2);  %Die Summe der Quadrate der Abstände der Messwerte zur
errechneten Kurve
P=0;
Q=0;
R=100000;
if  x(1)<0; %Wenn kmin(1) negativ ist, wird es mit R multipliziert und von
0
    P=P-R*x(1); % abgezogen
end
if x(2)<0;  %Wenn kmin(2) negativ ist, wird es mit R multipliziert und von
0
    P=P-R*x(2); % abgezogen
end
if x(3)<0;  %Wenn kmin(3) negativ ist, wird es mit R multipliziert und von
0
    P=P-R*x(3); % abgezogen
end
if x(4)<5;  %Wenn das Blutvolumen kleiner als 5 ist, wird es mit 7
    P=P-7*x(4); % multipliziert und von 0 abgezogen
if q>-5; % Wenn das Volumen kleiner als -5 ist, wird es mit 7 multipliziert
    x(4)=7*x(4);
end
y=y+P;
end
```

2. Mit der Summe der Absolutbeträge der Abstände der Messwerte zur errechneten Kurve:

```
kmin = fminsearch(@Jabs, x0); %Findet ein Minimum
function y = Jabs(x)
C=conc(x);
y=sum(abs(C-Xi)); %Die Summe der Absolutbeträge der Abstände der Messwerte
zur errechneten Kurve
P=0;
q=0;
R=100000;
if  x(1)<0; %Wenn kmin(1) negativ ist, wird es mit R multipliziert und von
0
    P=P-R*x(1); % abgezogen
end
if x(2)<0;  %Wenn kmin(2) negativ ist, wird es mit R multipliziert und von
0
    P=P-R*x(2); % abgezogen
end
if x(3)<0;  %Wenn kmin(3) negativ ist, wird es mit R multipliziert und von
0
    P=P-R*x(3); % abgezogen
end
if x(4)<5;  %Wenn das Blutvolumen kleiner als 5 ist, wird es mit 7
```

```
    q=q-7*x(4); % multipliziert und von 0 abgezogen
if q>-5; % Wenn das Volumen kleiner als -5 ist, wird es mit 7 multipliziert
    x(4)=7*x(4);
end
y=y+P;
end
```

Wir suchten die Minima der Funktion mit „fminsearch“ in MATLAB. Damit die Werte für k_1, k_2 und k_0 nicht negativ und das Blutvolumen nicht weniger als 5 ist.

Schwierigkeiten bei der Programmierung mit MATLAB

Negative Parameter:

MATLAB bot uns nicht nur eine Hilfe mit der Modellierung, sondern auch ein paar Hürden. Zuerst stellte sich das Problem, dass die Konstanten k_1, k_2, k_0 öfters negativ waren. Dies hätte zu bedeuten, dass der Körper BSP erzeugt und nicht abbaut. Um dem vorzubeugen fügten wir den Funktionen
„function y = Jqu(x)“ und „function y = Jabs(x)“

```
P = 0;
R=100000;

If x (1) < 0;
P = P-R*x (1);
End

If x (2) < 0;
P = P-R*x (2);
End

If x (3) < 0;
P = P-R* x (3);
End

y =y + P;
```

hinzu. Die Überlegung ist, dass negative Optima, die das Programm unter Umständen finden könnte, mit einem großen negativen Faktor P multipliziert werden und dadurch nicht mehr als bestmöglicher Wert in Frage kommen. Graphisch bedeutet das für den Punkt, dass er entlang einer Geraden in einen für das Programm uninteressanten Wertebereich verschoben wird.

Einführung eines vierten Parameters

Als nächstes wollen wir die empirische Formel für das Blutvolumen entfernen und behandeln deshalb V als unbekannte Konstante. Folglich bestimmen wir mittels

```
a=5;
b=7;
V = a+(b-a)*rand (1);
```

den ungefähren Bereich für das Blutvolumen, wobei a und b die anfänglichen Grenzwerte für das Blutvolumen sind und der Befehl „rand“ eine Funktion von MATLAB ist die eine Zufallszahl zwischen 0 und 1 bestimmt. Nun fügen wir den beiden Funktionen
„function y = Jqu(x)“ und „function y = Jabs(x)“

```
q=0;
if x (4) < 5;
q = q-7*x (4);

if q > -5
x (4) = 7*x (4);
end
end
```

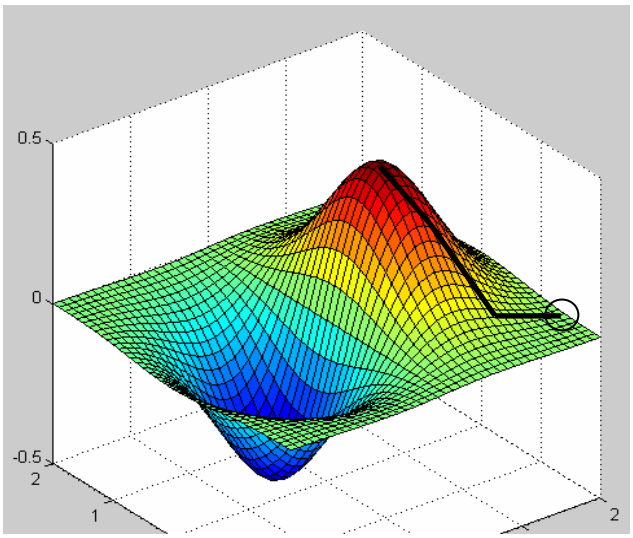
hinzu, um „fminsearch“ vom Annähern an Zahlen unter 5 abzuhalten. Der Rest des Codes bleibt unverändert, nach wie vor bestimmen wir k_1, k_2, k_0 und definieren V als vierte Koordinate x(4) des Vektors X_0. Das Übrige erledigt das Programm, indem es den Idealwert für das Blutvolumen sucht.

Ungenauigkeit durch Überbestimmtheit:

Leider ist es nicht unbedingt hilfreich, viele Messwerte für die Bestimmung der Parameter zu haben. In unserem Fall nimmt die Güte der Parameterschätzung durch Berücksichtigung der letzten 5 Messungen deutlich ab. Die sogenannte Sensitivitätsanalyse erlaubt es die Empfindlichkeit des Systems gegenüber Änderungen in den Parametern zu bestimmen. Mittels dieses Verfahrens stellen wir fest, dass sich die letzten fünf Angaben negativ auf die Parameterschätzung auswirken: Da das Programm versucht sich auch an diese fehlerhaften Daten (die Konzentration müsste stärker gegen 0 konvergieren) anzupassen entstanden Fehler bei den übrigen Annäherungen an die Messwerte. Deshalb ignorierten wir die Werte, die nach zwanzig Minuten gemessen wurden und verringerten so die Differenz zwischen Funktion und Angabe, ohne das Ergebnis zu verfälschen.

Lokal statt Global

Weiters erhält man mit verschiedenen Eingaben teilweise nicht dieselben Konstanten. Der Grund dafür ist, dass der Befehl „fminsearch“ meistens ein lokales Minimum findet, das sich in der Nähe der Startwerte befindet, anstatt des von uns gesuchten globalen Minimums.
Vorstellen kann man sich den Vorgang anhand dieser Grafik:



Hier sieht man, dass das Programm zwar ein Minimum gefunden hat, doch ist es nur ein Lokales im Gegensatz zum Globalen. Dieser Fehler kommt zustande, da „fminsearch“ vom angegebenen Startwert ausgeht und den kleinsten Punkt in der Nähe absucht. Von dort aus sucht das Programm den nächsten kleinsten Punkt und wiederholt diesen Prozess, bis sich kein tiefer gelegener Punkt in der Nähe finden lässt. Wie man aber in der oberen Grafik sehen kann, ist das lokale Minimum, in dem das Programm seine Suche beendet hat, umgeben von mehreren Maxima. Deshalb hört das Programm auf, obwohl es noch kleinere Punkte geben würde.

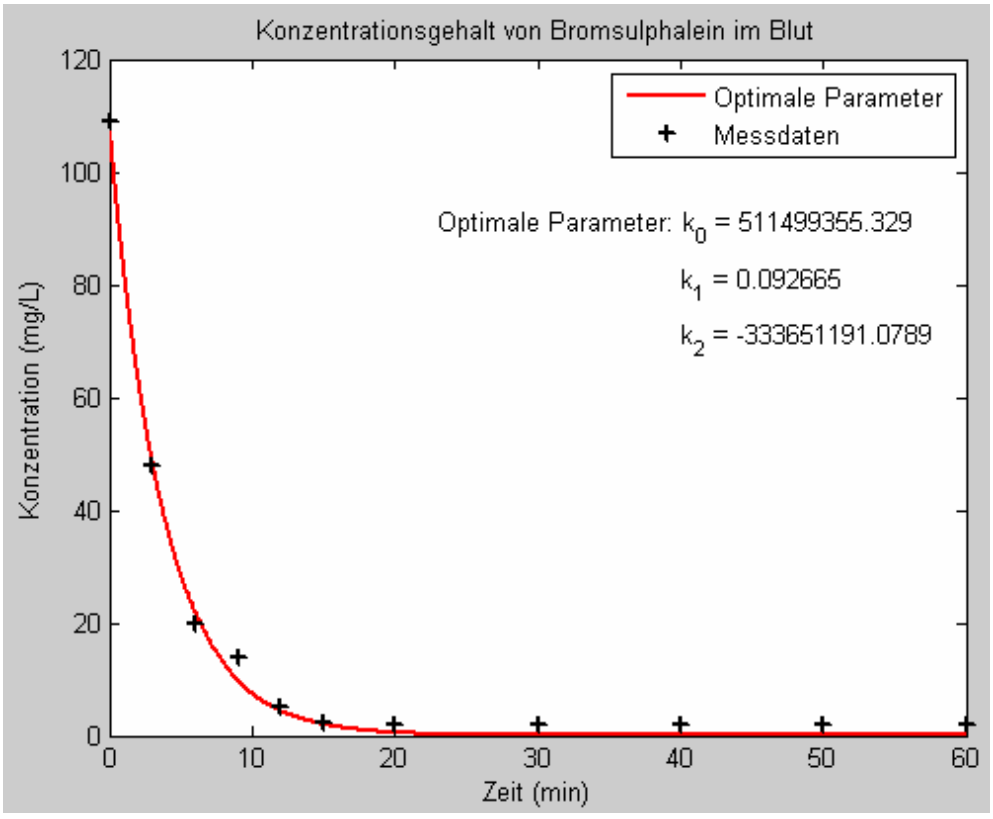
Wegen dieses Problems findet das Programm auch bei der Summe der Absolutbeträge und den Quadraten unterschiedliche Werte.

Eine Möglichkeit das Globale Minimum immer zu finden, wäre unendlich viele Ebenen parallel zueinander zu schichten und dem Programm aufzutragen, den Wert anzugeben, wenn es nur einen Schnittpunkt, oder Berührpunkt gibt. Ein Problem dabei ist, dass man unendlich viele Ebenen bräuchte, da der Unterschied der tiefsten Gebiete nahezu identisch sein könnte, was gleichzeitig eine unendlich große Rechenleistung eines Computers erfordern würde, um globale Minima zu finden.

Kritik an der verwendeten Quelle

Die unten erwähnte Kritik bezieht sich auf das Buch: „Lecture Notes in Biomathematics, Mathematical models in medicine: Mainz, March 1976 / Berger, Jürgen. - Berlin : Springer, Seite 250-253, Bromsulphalein Test

Während wir uns mit den Messdaten der wissenschaftlichen Publikation auseinandersetzten, kamen wir dahinter, dass diese auch nicht ohne Mangel waren. An erster Stelle sei zu erwähnen, dass die Autoren ihre Funktion wahrscheinlich ohne mathematischen Hintergrund an die Messwerte angepasst hatten. Auf diese Schlussfolgerung kamen wir indem wir versuchten die gleiche Funktion zu ermitteln wie im Buch angegeben wurde. Als wir dann nach langen Mühen eine beinahe identische Funktion erzeugt hatten, mussten wir mit Bedauern feststellen, dass die Veröffentlicher des Buches wahrscheinlich negative k Werte verwendet hatten um die Messwerte so gut zu treffen und die Funktion so gut aussehen zu lassen. Aber negative Werte für unsere Konstanten würden biologisch unsinnige Ergebnisse liefern wie zum Beispiel eine Leber die Bromsulphalein produziert oder ein Blutkreislauf der von einer leeren Leber BSP abzieht.



Aus diesem Grund mussten wir dann eine Sensitivitätsanalyse durchführen und feststellen, dass jene Werte die hinter der Sensitivitätsschwelle (t=20) liegen die Genauigkeit der Kurve nicht verbessern sondern deren Qualität verschlechtern.
In der Graphik, die mit dem frühen Programm Kartoffel02 erzeugt wurde, ist erkennbar, dass sich die Funktion sehr gut an die Werte annähert aber automatisch einen negativen Wert k_2 als Optimum berechnet, was eigentlich unmöglich ist.

Danksagung

Um unseren Bericht zu beenden, wollen wir uns bei allen Organisatoren, Sponsoren und Mitwirkenden bedanken, durch deren Einsatz und Unterstützung uns die unvergessliche Modellierungswoche auf dem Schloss Seggau ermöglicht wurde!

Ein großes Dankeschön auch an die Angestellten im Schloss, die sich ständig für unser Wohlbefinden einsetzten, die Unterkunft sauber hielten, uns mit ausgezeichnetem Essen versorgten und wir uns dadurch wie zu Hause fühlten.

Danksagung verdient auch unser Gruppenbetreuer und Koordinator,

Herr Prof. Franz Kappel, der uns in der Woche mit viel mathematischen Wissen, Geduld und Freundlichkeit zur Seite stand! Danke Herr Professor!!

Vielen Dank auch an Markus Müller, der sich besonders um unser Wohlergehen untereinander, sowohl in der Projektgruppe als auch bei der Gestaltung unserer Freizeit, kümmerte.

Natürlich sind wir auch den anderen Schülern, die an der Modellierungswoche teilgenommen haben, dankbar, dass sie immerzu freundlich und hilfsbereit waren.

Treffsicherheit im Sport

Projekt:
Sportwissenschaften

Betreuerin:
Dr. Sigrid Thaller

Modellierer/innen:
Jakob Bloder, Eva Groß, Sebastian Hofer, Nadja Koch, Amanda Schalk, Nina Waldner

Inhaltsverzeichnis

Problemstellung:

In diesem Projekt sollen mathematische Modelle für Bewegungen entwickelt werden, mit denen man beschreiben kann, wie man trotz kleinerer oder größerer Abweichungen einen Treffer erzielen kann. Die Ergebnisse werden mit realen sportlichen Bewegungen verglichen. Wir konzentrierten uns hauptsächlich auf den Wurf in den Basketballkorb.

- *Wo ist der optimale Abwurfpunkt?*
- *Was sind die möglichen Abwurfwinkel, um den Korb zu treffen?*
- *Welche Geschwindigkeit sollte man am besten wählen?*
- *Wie verhält sich der Ball, wenn er am Brett abprallt?*

Freier Fall:

Für den senkrechten Freien Fall ist die Erdanziehungskraft (Gravitation) verantwortlich. Dabei bewegt sich ein Körper ohne die Wirkung von anderen Kräften dem Boden entgegen.

Ort, Geschwindigkeit und Beschleunigung des Freien Falls können mit folgenden Formeln berechnet werden:

- Senkrechte Ortskomponente *y(t)* , abhängig von der Zeit *t*:

$$y(t) = -g \cdot \frac{t^2}{2} + v_0 \cdot t + y_0$$

- Die 1. Ableitung der Ortskomponente beschreibt die Geschwindigkeitskomponente $v_y(t)$:

$$y'(t) = v_y(t)$$

$$v_y(t) = -gt + v_{y_0}$$

- Durch die 1. Ableitung der Geschwindigkeitskomponente ergibt sich die Beschleunigung *a(t)*:

$$v_y'(t) = a(t)$$

$$a(t) = -g$$

$$g = 9{,}81 m/s^2$$

Modelle

1.Modell: Flugkurve eines Balles in die Korbmitte

Ein Basketballspieler steht vier Meter vom Korb entfernt und versucht den Ball gezielt in die Korbmitte zu werfen.

Dabei geht man von folgenden Daten aus:

Abstand *l* zum Korb	4m
Höhe *H* des Korbes	3,05m
Größe des Basketballers mit gestreckten Armen	2,05m
Abstand *h* Arme-Korb	1m
Korbdurchmesser	45,7cm
Balldurchmesser	24cm
Gravitation	9,81m/s²

Um das Ganze zu veranschaulichen wurde diese Skizze erstellt.

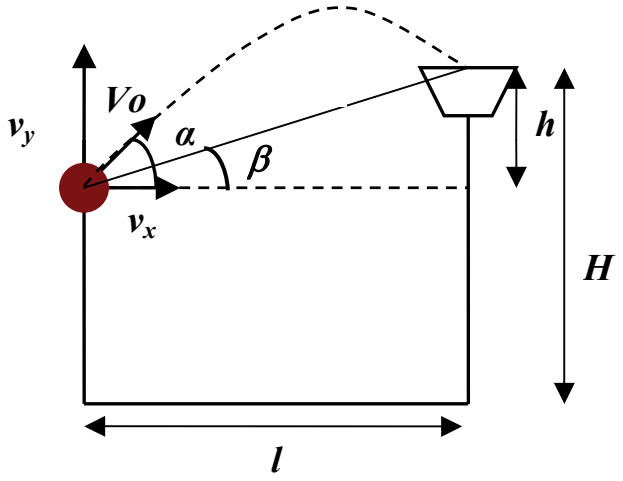


Abbildung 1

- Die Flugbahn des Balles wird durch seine Anfangsgeschwindigkeit v_0 und den Abwurfwinkel α bestimmt.
- Die Geschwindigkeit v_0 kann in eine X-Komponente v_x und in eine Y-Komponente v_y zerlegt werden.
- Während des ganzen Fluges bleibt die X-Komponente gleich.
- Die Y-Komponente ändert sich jedoch im Laufe der Zeit.

$$v_x(t) = v_0 \cdot \cos\alpha$$

$$v_y(t) = v_0 \cdot \sin\alpha - g \cdot t$$

Dabei bezeichnet *g* die Gravitationsbeschleunigung und *t* die Zeit.

Der Ursprung des Koordinatensystems wird in den Abwurfpunkt gesetzt. Die Ortskoordinaten der Flugbahn erhält man durch:

$$x(t) = v_0 \cdot \cos\alpha \cdot t$$

$$y(t) = v_0 \cdot \sin\alpha \cdot t - g \cdot \frac{t^2}{2}$$

Aus der Gleichung für die X-Koordinate ergibt sich: $t = \frac{x}{v_0 \cdot \cos\alpha}$.

Das setzt man in die Gleichung der Y-Koordinate ein und man erhält die Gleichung der Flugbahn des Balles:

$$y = \tan\alpha \cdot x - \frac{g}{2} \cdot \frac{x^2}{{v_0}^2 \cdot \cos^2\alpha}$$

Wenn der Ball in die Mitte des Korbes trifft, hat er die Koordinaten *l* und *h*.

Wegen $\frac{1}{\cos^2\alpha} = \tan^2\alpha + 1$ ergibt sich für diesen Punkt:

$$h = \tan\alpha \cdot l - \frac{g}{2} \cdot \frac{l^2}{{v_0}^2} \cdot (\tan^2\alpha + 1)\,.$$

α...Abwurfwinkel
v_0…Abwurfgeschwindigkeit
h… Höhenunterschied zwischen Korb und Hand
l…Entfernung des Spielers zum Korb

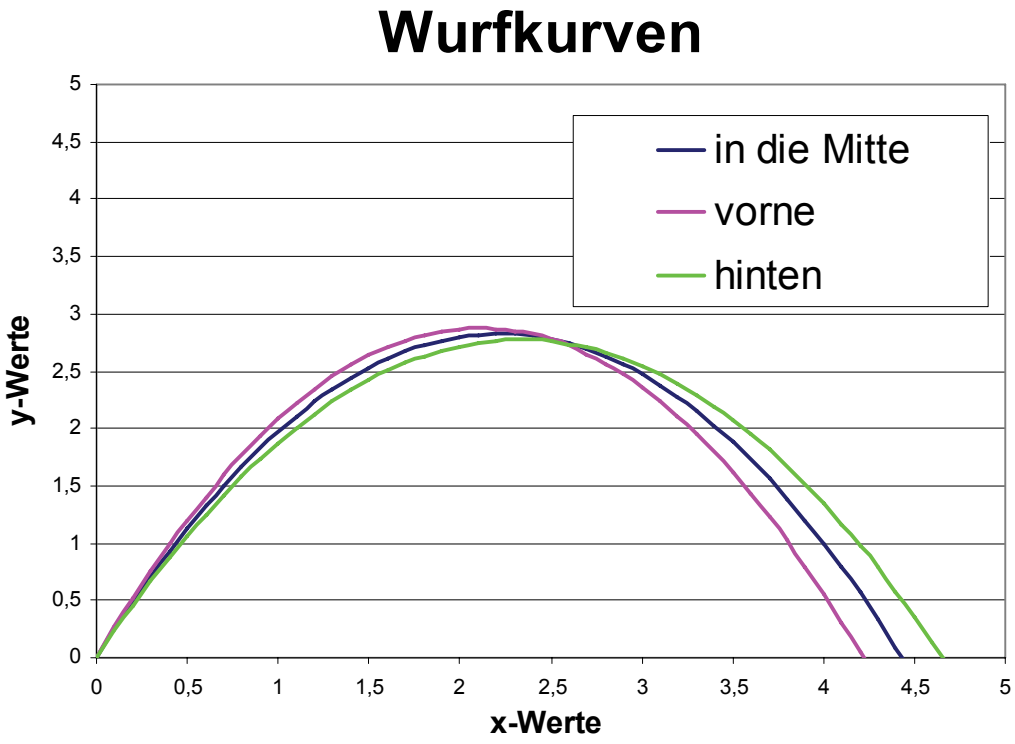


Abbildung 2

Diese Gleichung kann man nach tan α auflösen. Man erhält

$$\tan\alpha = \frac{v_0^2 \pm \sqrt{v_0^4 - g^2l^2 - 2ghv_0^2}}{gl}.$$

Der Ausdruck unter der Wurzel muss positiv sein, das heißt,

$$v_0^4 - g^2l^2 - 2ghv_0^2 \geq 0.$$

Daraus ergibt sich $v_0^2 \geq g(h + \sqrt{h^2 + l^2})$.

Es gibt also eine minimale Geschwindigkeit. Durch Einsetzen dieser Geschwindigkeit in die Formel für tan α erhält man den dazugehörigen Abwurfwinkel: $\tan\alpha = \frac{h + \sqrt{h^2 + l^2}}{l}$.

Wegen $\tan\beta = h/l$ ergibt sich $\tan\alpha = \tan\beta + \sqrt{\tan^2\beta + 1} = \tan(45°) + \beta/2$ und daher $\alpha = 45° + \beta/2$.

In dem bisherigen Modell war der Ball allerdings nur ein Punkt. Da es sich aber um einen Körper und nicht um einen Punkt handelt, muss beachtet werden, dass dieser auch eine gewisse Fläche einnimmt. Außerdem ist die Fläche des Korbes größer als die des Balles und somit kann dieser weiter vorne oder hinten in den Korb geworfen werden.

2.Modell: Mögliche Variationen der Flugkurve

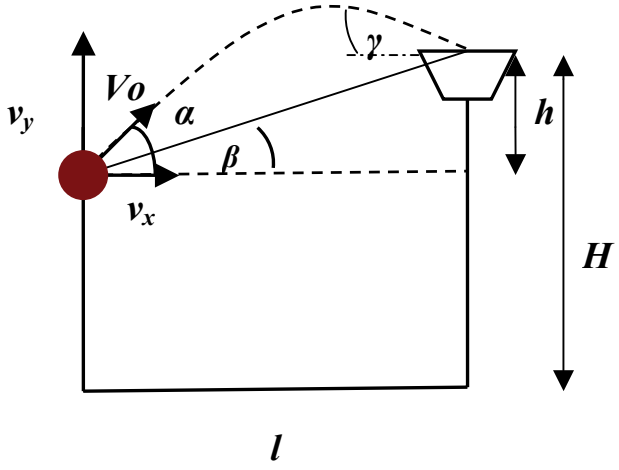


Abbildung 3

In dieser Abbildung sieht man alle wichtigen Parameter des Wurfes.
Der Winkel γ bezeichnet den Winkel, mit dem der Ball in den Korb eintrifft. Man kann den Zusammenhang zwischen den Winkeln α, β und γ berechnen.

Der Winkel β lässt sich aus dem Verhältnis vom Abstand des Spielers zum Korb (l) und der Höhe des Korbes über der Abwurfhöhe (h) berechnen: $\tan\beta = \frac{h}{l}$. Setzt man für h und l die Gleichungen der Koordinaten ein, ergibt sich:

$$\tan\beta = \tan\alpha - \frac{g \cdot t}{2 \cdot v_0 \cdot \cos\alpha}.$$

Aus dem Verhältnis der Geschwindigkeitskomponenten kann man den Winkel γ berechnen:

$$\tan\gamma = \frac{v_y}{v_x}.$$

Die Geschwindigkeitskomponenten der Wurfparabel können aus der Gleichung des Wurfes ausgerechnet werden. Daher erhält man:

$$\tan\gamma = -\tan\alpha + \frac{g \cdot t}{v_0 \cdot \cos\alpha}.$$

Fügt man die Gleichungen für $\tan\beta$ und $\tan\gamma$ zusammen, erhält man den Zusammenhang zwischen den Winkeln α, β und γ:

$$\tan\gamma = \tan\alpha - 2\tan\beta$$

Ein Basketball hat einen Durchmesser d_B = 24cm. Damit er ohne den Rand zu berühren in den Korb gelangt, darf der Eintrittswinkel nicht zu klein sein. Den minimalen Eintrittswinkel γ_{min} kann man aus dem Korbdurchmesser (d_K = 45,7cm) und der Größe des Balles berechnen:

$$\sin \gamma_{\min} = \frac{d_B}{d_K}$$

Der minimale Eintrittswinkel beträgt also ca. 32°.

Wenn der Ball die Mitte des Korbes nicht genau trifft, kann er je nach Eintrittswinkel ein bisschen abweichen und trotzdem noch durch den Korb fallen ohne den Rand zu berühren. Den Spielraum Δx, den der Ball hat, kann man aus dem Eintrittswinkel γ berechnen.

$$\Delta x = d_K - (\frac{d_B}{\sin \gamma}) \cdot \frac{1}{2}$$

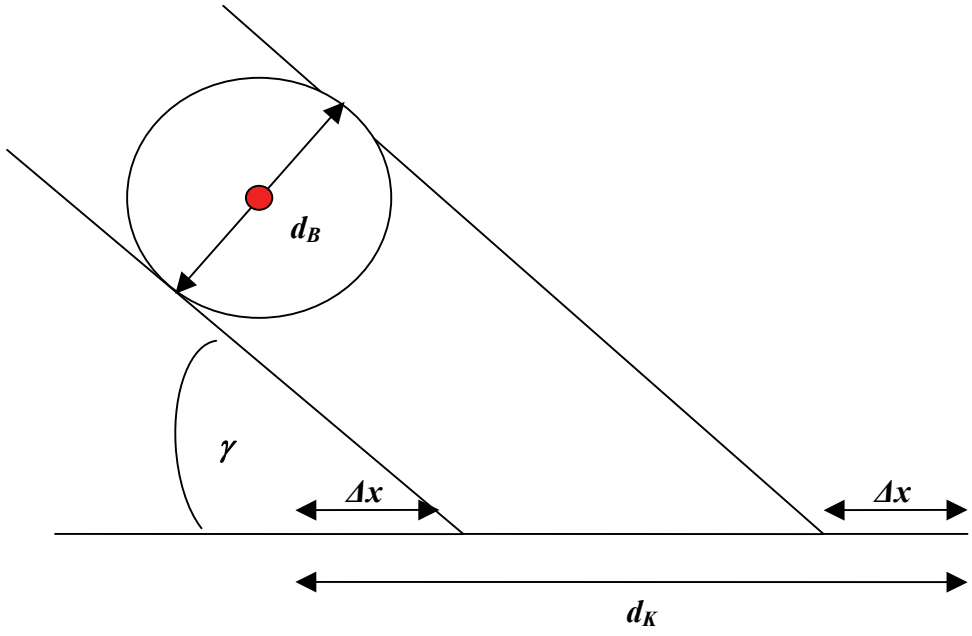


Abbildung 4

Um den Winkel γ auszudrücken, wandelt man die Formel $\tan \gamma = \tan \alpha - 2 \tan \beta$ in

$\gamma = \arctan\left(\tan \alpha - 2 \frac{h}{l} \right)$

um.

Um nun zu berechnen, wie sich Abwurfwinkel α und Geschwindigkeitsänderungen auswirken, nimmt man zuerst fixe Werte für die Entfernung l zum Korb und die Höhe h an: $l = 4\text{m}$; $h = 1\text{m}$
Daraus ergibt sich der minimale Abwurfwinkel α von 48,17°.
Den Eintrittswinkel γ kann man nach der obigen Formel aus dem Abwurfwinkel α beschreiben. Der Winkel β ist durch h und l eindeutig gegeben.

Dadurch lässt sich nun der Spielraum Δx berechnen. Zu jedem möglichen Abwurfwinkel α kann also die Geschwindigkeit v_0 variiert werden, sodass der Ball den vorderen oder hinteren Rand des Korbes gerade noch berührt. Daraus ergibt sich für jeden Winkel α ein ganzes Intervall an möglichen Abwurfgeschwindigkeiten.

Das folgende Diagramm stellt die möglichen Schwankungen von der Abwurfgeschwindigkeit v_0 bei verschiedenen Abwurfwinkeln α dar. Man kann erkennen, dass die Schwankungsbreite der Anfangsgeschwindigkeit v_0 bei größer werdendem Startwinkel α steigt.

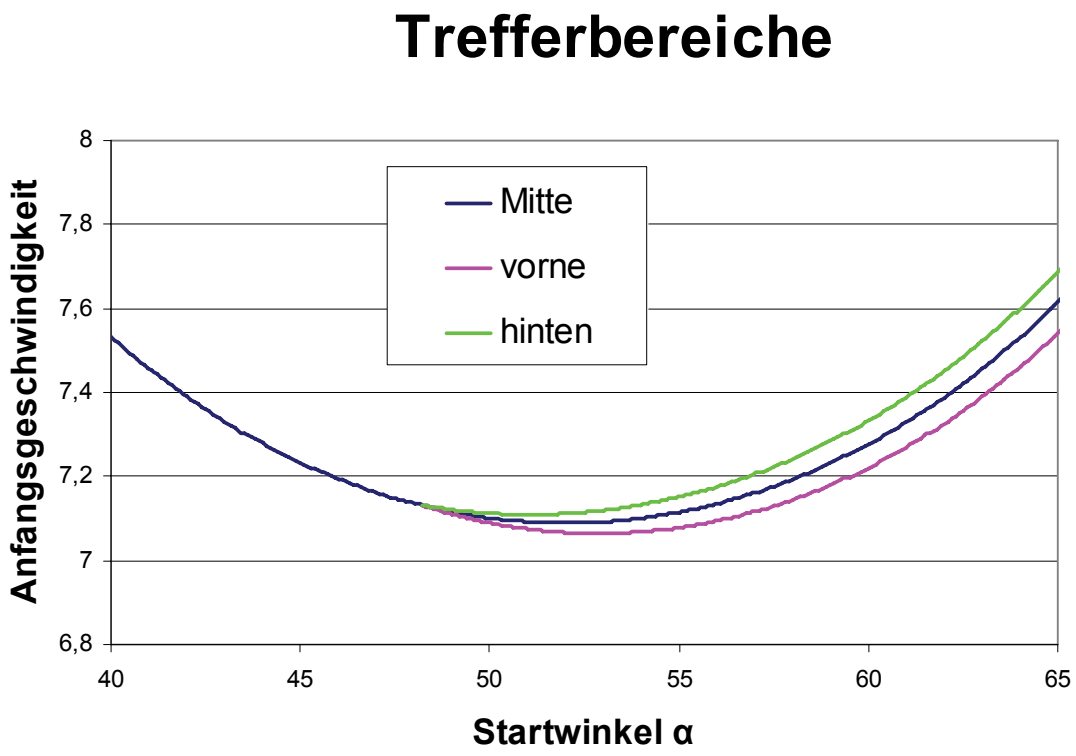


Abbildung 5

Trefferbereiche bei h = 1m und l = 4m

Betrachtet man eine konstante Anfangsgeschwindigkeit v_0 (horizontale Linien im Diagramm), so erkennt man aus der Grafik die Schwankungsbreite des Abwurfwinkels. Je kleiner die Abwurfgeschwindigkeit v_0 desto größer ist der Spielraum für den Abwurfwinkel α.
Die blaue Linie in der Mitte beschreibt jene Würfe, die genau in die Mitte des Korbes treffen.
Die Fläche zwischen den äußeren Kurven zeigt den Schwankungsbereich.
Die Treffsicherheit ist bei jenen Werten am größten, wo die meisten Schwankungen erlaubt sind.

Nimmt man für den Abstand vom Korb statt 4m nun 5m an, dann verschiebt sich der Bereich der möglichen Abwurfwinkel und Abwurfgeschwindigkeiten (siehe Abbildung 6).

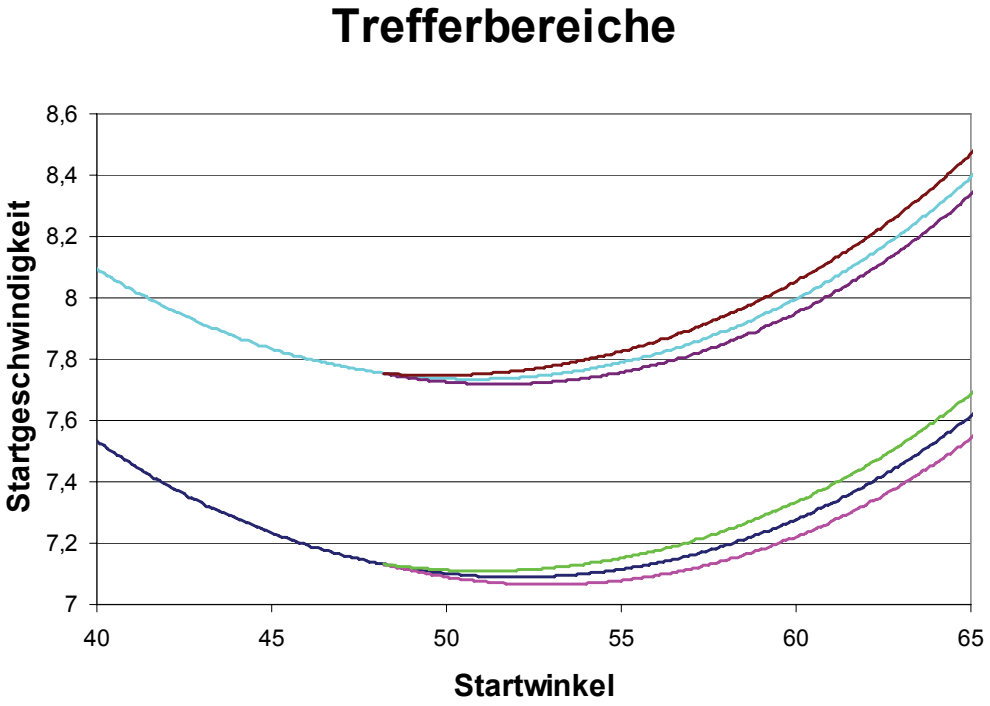


Abbildung 6

Um die Winkelabweichungen beim Abwurf nach rechts und links zu berechnen, müssen der Balldurchmesser d_B, der Korbdurchmesser d_K und der Abstand zwischen Abwurfpunkt und Korbmittelpunkt berücksichtigt werden.

Bei l = 4m und h = 1m beträgt der Abstand zwischen Abwurfpunkt und Korb 4,12m.
Der Ball hat einen seitlichen Spielraum von 12cm auf jeder Seite; daraus ergibt sich ein Spielraum für den seitlichen Winkel von $\delta = 3{,}3°$.

Diese Winkelabweichung ist aber nur erlaubt, wenn der Abwurfwinkel α exakt eingehalten wird. Kommt es bei α zu Ungenauigkeiten, d.h. dass der Ball weiter vorne oder hinten in den Korb gelangt, wird der seitliche Spielraum kleiner. So wie Variationen in v_0 und α nicht von einander unabhängig sind, so sind auch Variationen im Abwurfwinkel α und im seitlichen Winkel δ abhängig voneinander.

3.Modell: einfache Reflexion

Um das Modell noch realistischer zu gestalten, muss man die Rückwand des Basketballkorbes in die Berechnungen mit einbeziehen. Jedoch werden bei dem einfachen Reflexionsmodell die spezifischen Eigenschaften des Balles (Elastizitätskoeffizient *e* und Winkelgeschwindigkeit ω) noch weitestgehend vernachlässigt.

Für das einfache Modell gilt:

$$\alpha = \beta$$

$$v_{vx} = v_{nx}$$
$$v_{vy} = -v_{ny}$$

$$v = \sqrt{v_{vx}^{\,2} + v_{vy}^{\,2}}$$
$$v' = \sqrt{v_{vx}^{\,2} + v_{vy}^{\,2}}$$

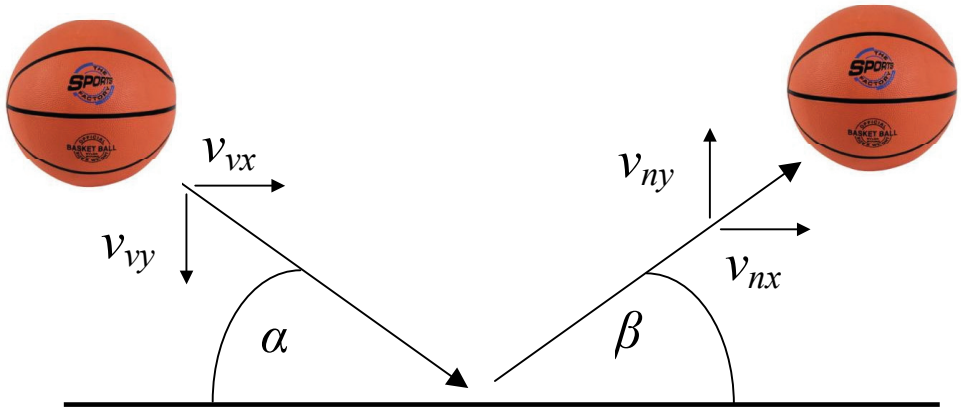


Abbildung 7

Wie man aus der Grafik bereits erkennen kann, sind im einfachen Reflexionsmodell Eintrittswinkel und Ausfallswinkel genau gleich groß. Weiters ändert sich weder die X-Komponente der Geschwindigkeiten (v_{vx} und v_{nx}), noch die Y-Komponente (v_{vy} und v_{ny}), welche jedoch nach dem Kontakt mit der Reflexionswand das Vorzeichen ändert. Da die eigentlichen Geschwindigkeiten direkt über den pythagoräischen Lehrsatz ($a^2 = b^2 + c^2$) in Verbindung mit den jeweiligen *v*-Komponenten stehen, ergeben sich auch für diese identische Werte.

Gleichzeitig kann man das Reflexionsproblem in der Theorie umgehen, indem man die Wurfkurve einfach weiterlaufen lässt, ohne die Reflexionswand zu berücksichtigen. Um trotzdem zu richtigen Eintreffpunkten zu gelangen, stellt man sich einfach einen fiktiven Korb vor, der durch Spiegelung am Brett entsteht.
Trifft nun die künstlich verlängerte Parabel in den fiktiven Korb, so kann man durch erneutes Spiegeln feststellen, wo der Ball in den realen Korb eintritt.

Allerdings muss man beachten, dass dieses Reflexionsmodell in der Praxis kaum Anwendung finden wird, da folgende Voraussetzungen exakt erfüllt sein müssen:

- Die Winkelgeschwindigkeit (ω_v) vor dem Kontakt muss genau das negative Äquivalent der reflektierten Winkelgeschwindigkeit (ω_n) sein.

- Der Elastizitätskoeffizient muss annähernd 1 sein, d.h. es darf zu (fast) keiner Verzerrung des Balles kommen.

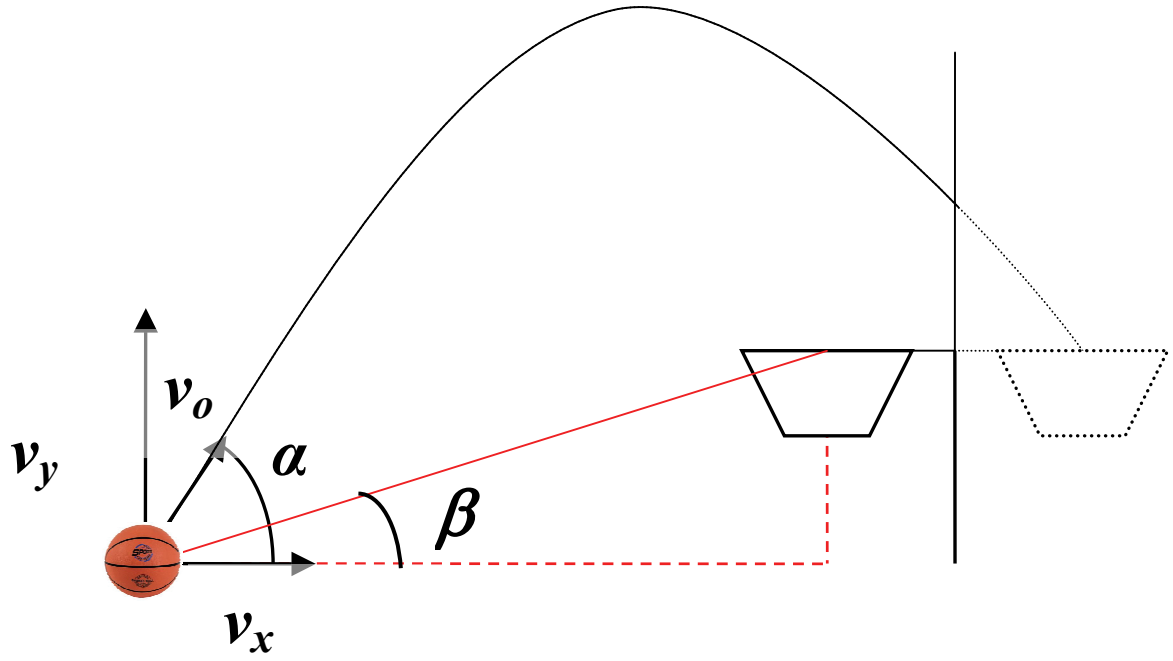


Abbildung 8

Folgendes Diagramm beschreibt die Abhängigkeit vom Startwinkel α zu den Geschwindigkeiten und umgekehrt. Die unterschiedlichen Funktionen entstehen durch den zusätzlichen Weg über die Reflexionswand in den Korb:

- Die dunkelblaue Kurve beschreibt den direkten Weg in den Mittelpunkt des Korbs.
  Die umliegenden grünen und pinken Kurven ergeben sich aus der Berechnung der zugehörigen Varianzen und drücken die Trefferbereiche aus.

- Die auf der Y-Achse verschobenen Funktionen stellen die reflektierten Wurfkurven und deren Varianzen dar.

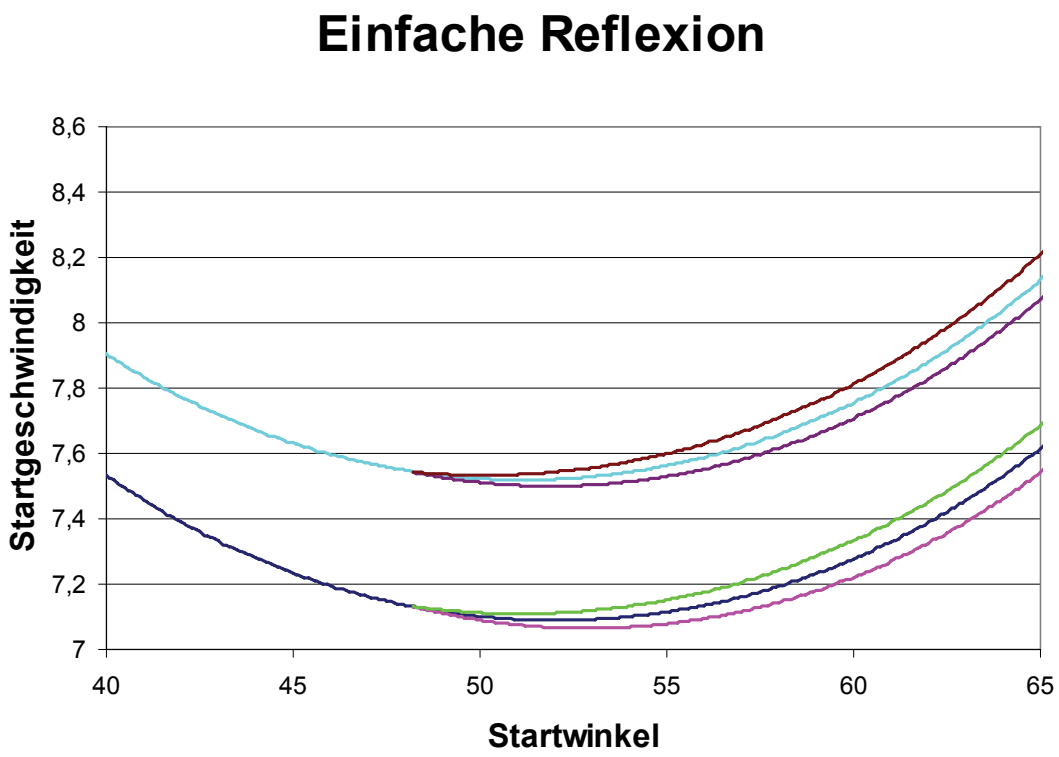


Abbildung 9

4.Modell: Reflexion mit Rotation

Die Weiterentwicklung des einfachen Reflexionsmodells berücksichtigt bereits mehr der relevanten Parameter, die den Flug des Balls in der Realität beeinflussen.

Elastizitätskoeffizient e

Das Quadrat des Elastizitätskoeffizient *e* kann über Experimente berechnet werden, indem man das Objekt fallen lässt und die Rücksprunghöhe durch die Abwurfhöhe dividiert. Für *e* ist ein Wert zwischen 0 und 1 anzunehmen, wobei die Elastizität des Balles bei 1 am höchsten ist.

$$e = \sqrt{\frac{Rücksprunghöhe}{Abwurfhöhe}}$$

Winkelgeschwindigkeit ω

Damit man die Geschwindigkeitsveränderung beim Kontakt mit der Reflexionswand exakter berechnen kann, muss man über die Winkelgeschwindigkeit des geworfenen Balles Bescheid wissen. Diese definiert, wie schnell sich der Ball pro Sekunde dreht (Einheit: rad/s).

Vorzugsweise wird dem Basketball beim Abwurf ein starker *Backspin* mitgegeben. Dies erfolgt intuitiv vor allem in der letzten Phase der Wurfbewegung. In diesem Fall bedeutet ein *Backspin* in Relation zur senkrechten Reflexionswand eine Drehung gegen den Uhrzeigersinn.

Der auftretende Effekt bewirkt eine Erhöhung der senkrechten Komponente der Geschwindigkeit nach dem Kontakt, was zu einer Verkleinerung des Reflexionswinkels führt. Der Vorteil besteht in der größeren Fehlertoleranz, mit der man nun Abwurfwinkel und Startgeschwindigkeit wählen kann.

Weiters müssen zur Modellierung der Reflexion verschiedenste Werte einkalkuliert werden, um sicherzustellen, dass der berechnete Eintrittspunkt auch mit dem realen übereinstimmt. Hierzu muss man genaue Kenntnis über die X- und Y-Komponenten der Abwurfgeschwindigkeit, sowie alle wesentlichen Winkeln haben.

Länge zum Korbmittelpunkt	4m
Höhe	1m
Korbdurchmesser	45,7cm
Höhe der Reflexionswand	1,05m
Abstand zur Reflexionswand	4,3285m

Reflexionspunkt:

Zuallererst muss man jenen Punkt berechnen, an dem der Ball das Brett berührt. Hierzu muss man die Parabelfunktion mit der Gleichung der Geraden, auf der das Brett liegt, schneiden.

Wurfkurve:

$$y = \tan\alpha \cdot x - \frac{g}{2} \cdot \frac{x^2}{{v_0}^2} \cdot (\tan^2\alpha + 1)$$

Brett:

$$x = 4{,}3285$$

Reflexionspunkt:

$$y = \tan\alpha \cdot 4{,}3285 - \frac{9{,}81}{2} \cdot \frac{4{,}3285^2}{{v_0}^2} \cdot (\tan^2\alpha + 1)$$

Die Veränderung der Y-Koordinate bei gleich bleibender Anfangsgeschwindigkeit von 8m/s aber einem variablen Alpha-Winkel wird in folgendem Diagramm deutlich veranschaulicht. Zu erkennen sind bereits jene Winkel, mit denen die Wurfkurve das Brett erst gar nicht berührt ($y < 1$).

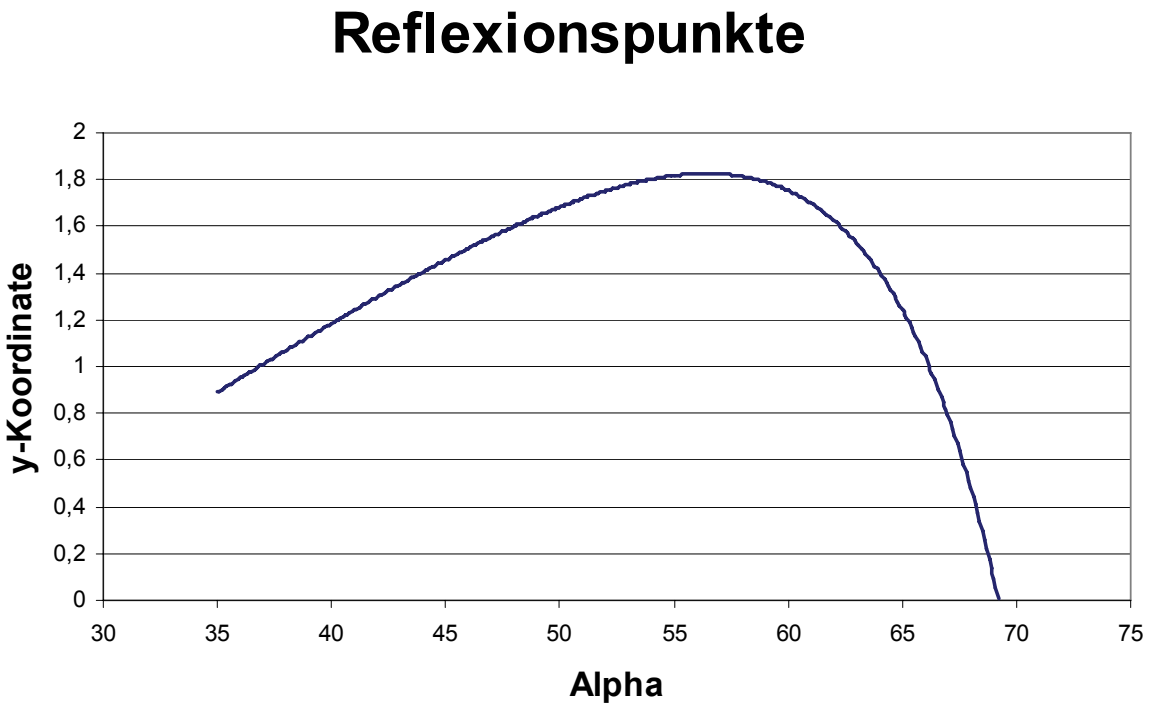


Abbildung 10

Reflexionszeitpunkt:

Hat man den Schnittpunkt der Parabel und der Gerade gefunden, muss man noch den genauen Zeitpunkt des Kontaktes berechnen. Dies erfolgt über eine Umformung der zeitabhängigen Ortskomponente *x(t)*.

Ortskomponente:

$$x(t) = v_0 \cdot \cos\alpha \cdot t$$

Zeitpunkt:

$$t = \frac{x(t)}{v_0 \cdot \cos\alpha}$$

Für $x = 4{,}3285$ (m) und $v_0 = 8$ (m/s) ist $t = 0{,}54/\cos\ \alpha$ (s).

Die Geschwindigkeit am Reflexionspunkt beträgt vor der Reflexion

$$v_x = v_0 \cos\ \alpha,\ v_y = v_0 \sin\ \alpha - 4{,}3285\ g\ /(v_0 \cos\ \alpha).$$

Geschwindigkeit nach dem Abprall

Die X-Komponente (senkrecht, siehe Abbildung 7) der Geschwindigkeit kann man mittels der Winkelgeschwindigkeit und v_{vx} berechnen:

$$v_{nx} = \frac{5}{7} \cdot v_{vx} - \frac{2}{7} r \cdot \omega_v$$

Für die dazugehörige Y-Komponente benötigt man noch zusätzlich den Elastizitätskoeffizienten:

$$v_{ny} = -e \cdot v_{vy}$$

Winkelgeschwindigkeit nach dem Abprall

Die Winkelgeschwindigkeit ω_n muss sich beim Kontakt mit dem Brett automatisch verändern. Hier kommt es zu einer Übertragung kinetischer Translationsenergie (v_{vx}) zu Rotationsenergie.

$$\omega_n = \frac{5}{7} \cdot \frac{v_{vx}}{r} + \frac{2\omega_v}{7}$$

Aufstellen der Reflexionsparabel

Um schlussendlich den gesuchten Eintrittspunkt des Balles zu erhalten, muss man eine völlig neue Parabelfunktion aufstellen.

Neuer Anfangspunkt ist der Schnittpunkt der Wurfparabel mit der Reflexionswand.

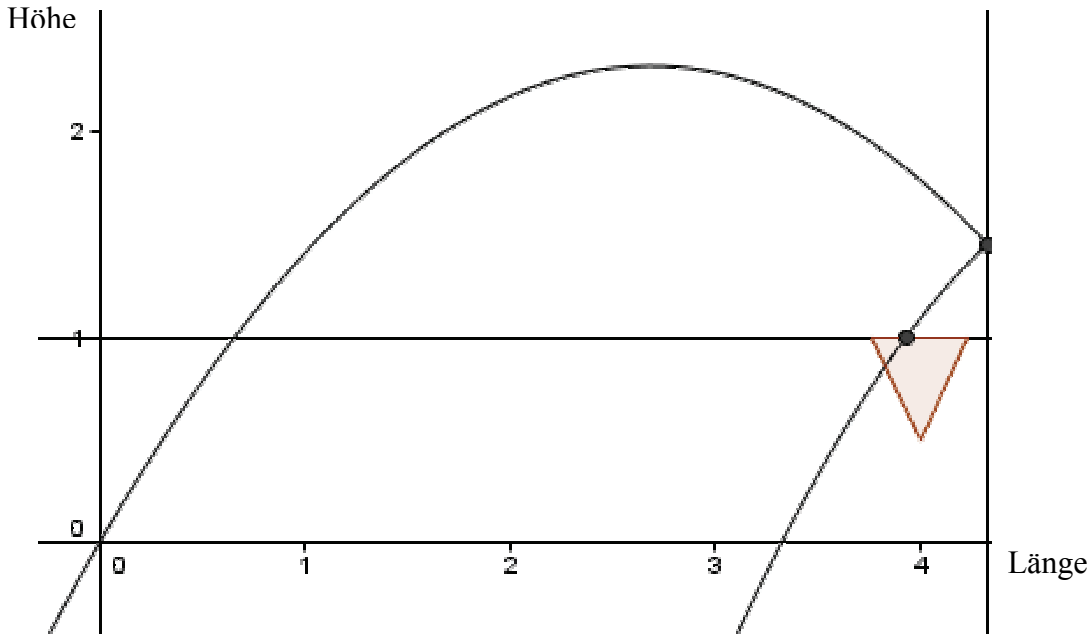


Abbildung 11

Neue Anfangsgeschwindigkeit:

Um die Anfangsgeschwindigkeit v_0 der neuen Parabel zu berechnen, kommt wieder der pythagoreische Lehrsatz zum Einsatz:

$$v_{0n} = \sqrt{v_{nx}{}^2 + v_{ny}{}^2}$$

Neuer Abwurfwinkel

Der Tangens des Startwinkels ergibt sich aus dem Quotienten von v_{ny} und v_{nx}.

$$\alpha' = -\arctan(\frac{v_{ny}}{v_{nx}})$$

Schneidet man die neue Parabel mit der waagrechten Gerade $y = 1$, so kann man den Punkt berechnen, in dem der Basketball die Höhe des Korbes erreicht. Nun muss wieder der Eintrittswinkel beachtet werden, um zu berechnen, ob der Ball in den Korb trifft.

Die folgende Abbildungen zeigt die x-Koordinate in Höhe des Korbes in Abhängigkeit des Abwurfwinkels (bei konstanter Abwurfgeschwindigkeit $v_0 = 8$ m/s)) für verschiedene Rotationsgeschwindigkeiten.

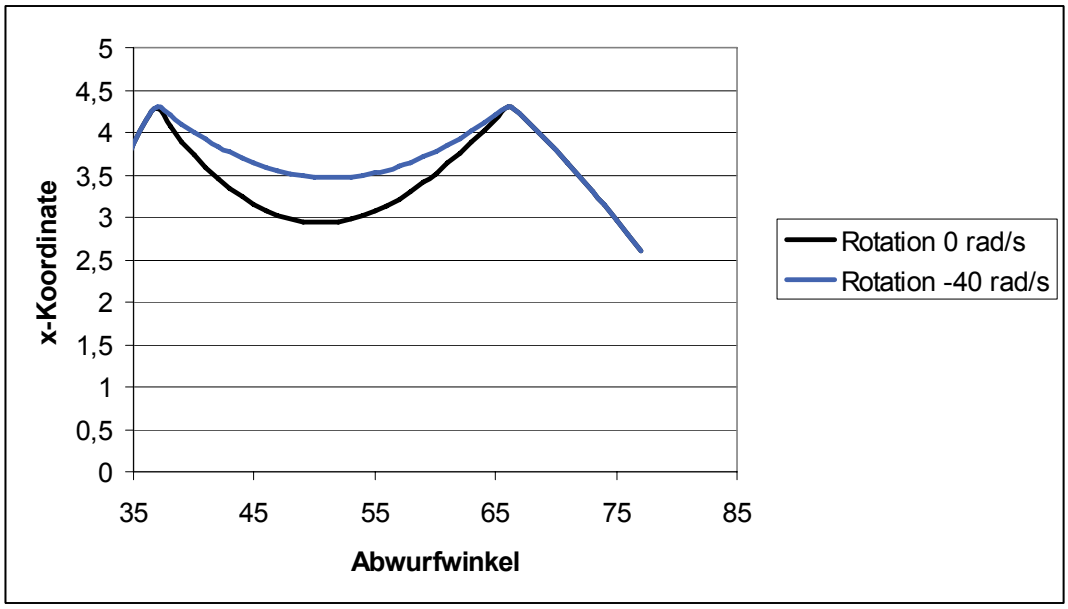


Abbildung 12

Deutlich zu erkennen ist der Unterschied zwischen beiden Kurven, der nur durch die unterschiedliche Rotationsgeschwindigkeit vor der Reflektion auftritt. Der *Backspin* führt zu einer wesentlich flacher verlaufenden Kurve, welche die größere Fehlertoleranz erklärt. Der Bereich, in dem sich die x-Koordinaten um 4 (= Korbmittelpunkt) bewegen, ist wesentlich größer. Der Anfang und das Ende beider Kurven verlaufen gleich, weil es bei diesen Abwurfwinkeln zu keiner Reflexion kommt.

Experimente:

Um zu sehen ob man diese Überlegungen auch in die Praxis umsetzen kann, gingen wir mit unserem Basketball ins Freie. Da wir keinen richtigen Basketballkorb zur Verfügung hatten, mussten wir improvisieren. So diente nach langem Suchen ein Mistkübel, den wir auf eine Mauer stellten, als geeignetes Ziel. Außerdem befestigten wir einen Maßstab um die Höhen genau messen zu können.

Um das Ganze zu veranschaulichen, kann man hier ein Bild davon sehen (Abbildung 14).

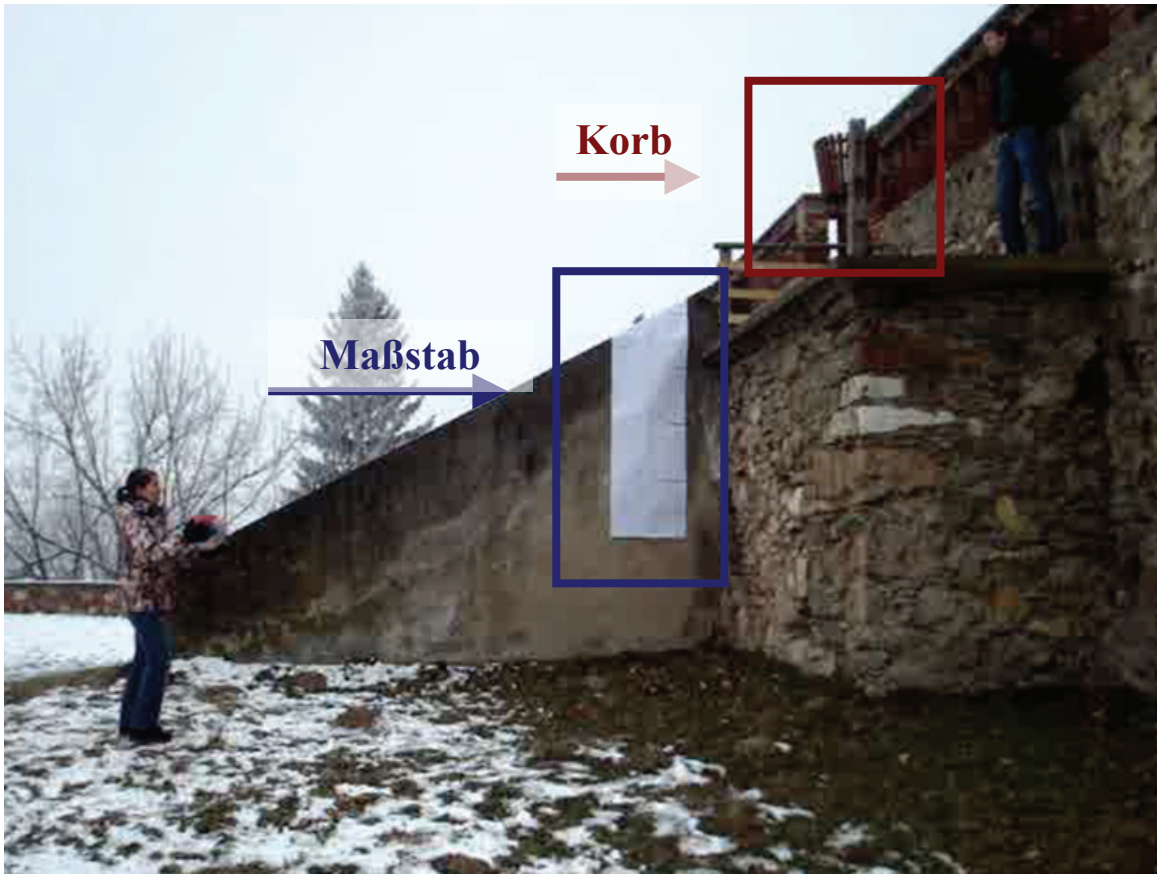


Abbildung 13

Zwei unserer Gruppenmitglieder versuchten nun ihr Glück, wobei nicht jeder Wurf ein Treffer war.
Nach mehreren Versuchen gelang es überraschenderweise doch, was wir natürlich mit unserer Kamera aufgezeichnet haben.

Um die Kurve zeichnen zu können, mussten wir zuerst die Höhe des Korbes und die Entfernung zur Mauer messen. Der schwierigste Teil war jedoch die Schätzung des Abwurfwinkels α.
Wir kamen zu folgenden Werten:

Für Abbildung 14:

Abstand vom Korb (gemessen)	4,2 m
Höhenunterschied Arme-Korb (gemessen)	1,8 m
Winkel α (geschätzt)	60°
Winkel β (berechnet)	23,2°
Winkel γ (berechnet)	41,2°
Anfangsgeschwindigkeit (berechnet)	7,95 m/s

Damit man sich das Ganze besser vorstellen kann, haben wir unsere Daten in das Bild eingezeichnet (Abbildung 4).

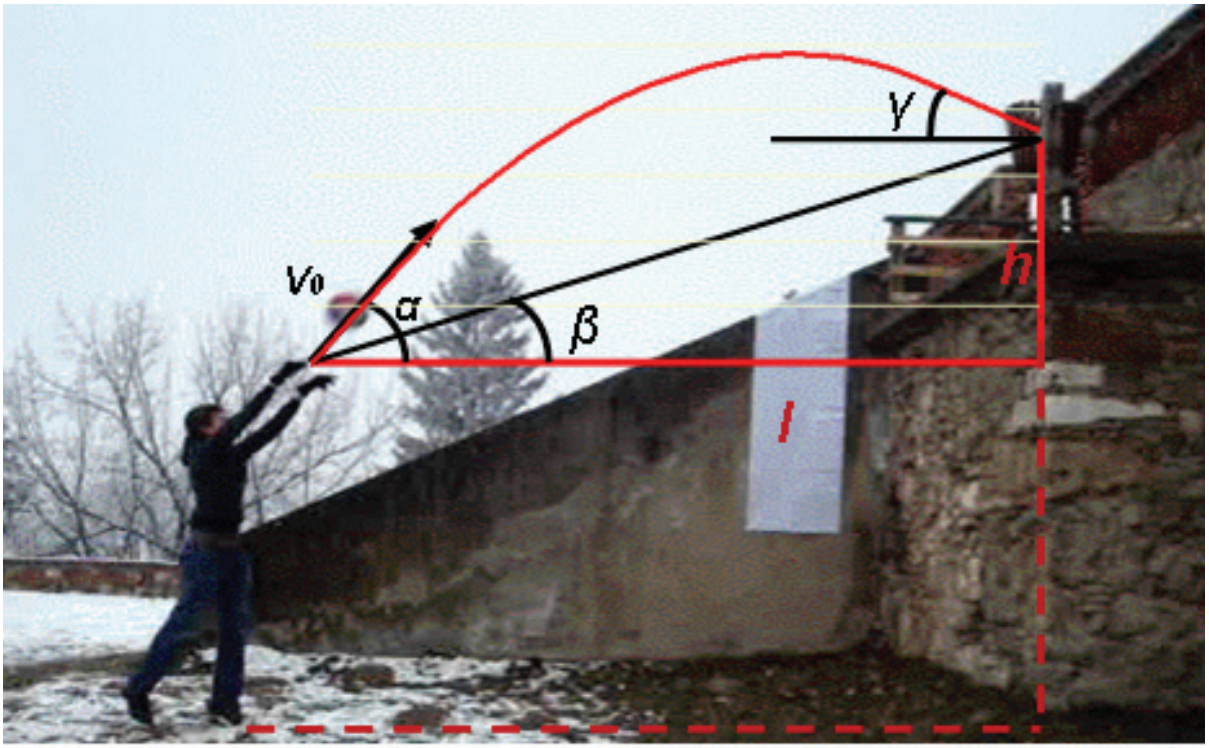


Abbildung 14

Für Abbildung 15:

Abstand vom Korb	4,2 m
Höhenunterschied Arme-Korb	2,0 m
Winkel α	69°
Winkel β	25,46°
Winkel γ	58,82°
Anfangsgeschwindigkeit	8,68 m/s

Auch zu dieser Tabelle gibt es wieder eine Veranschaulichung (Abbildung 14):

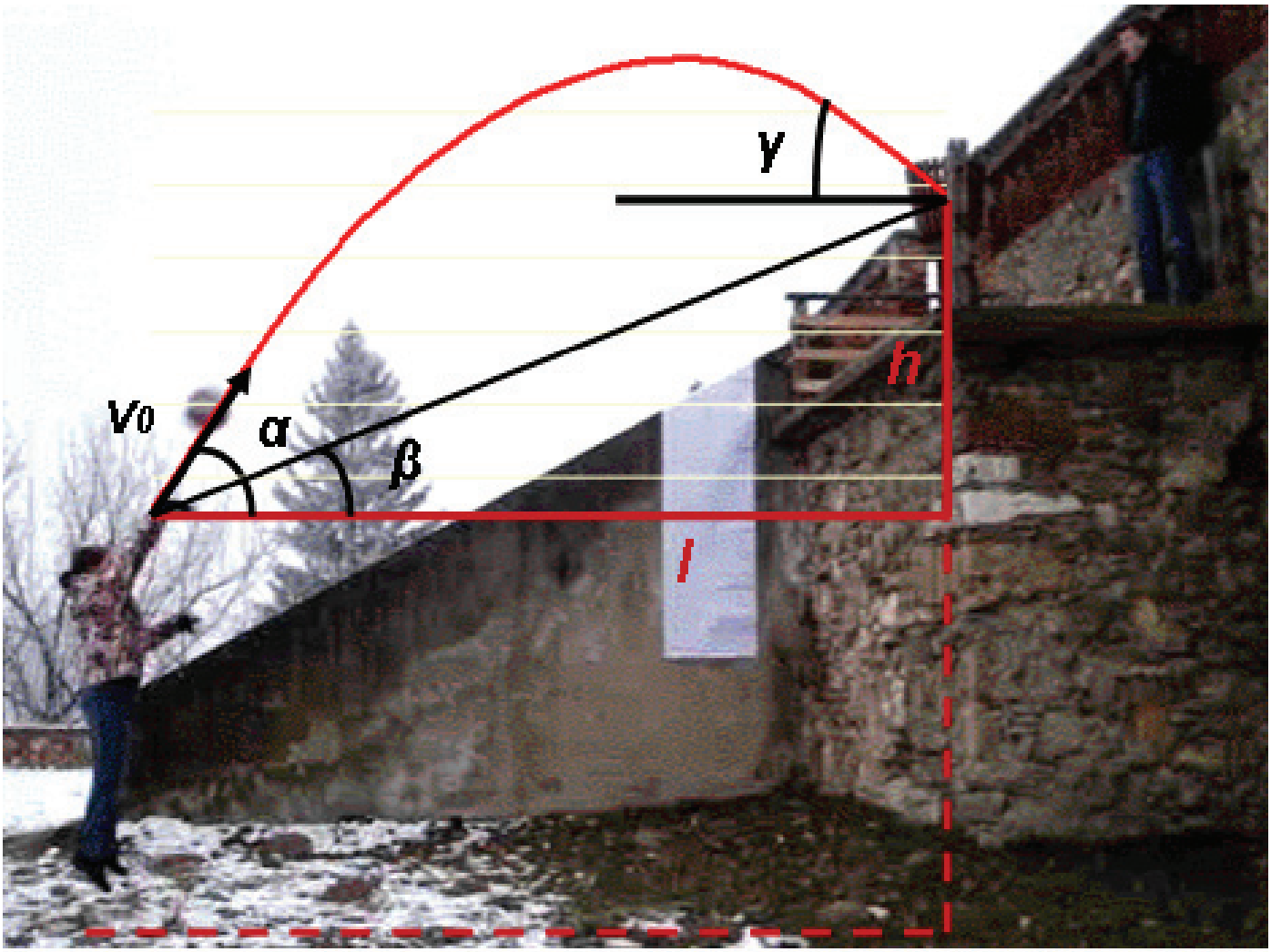
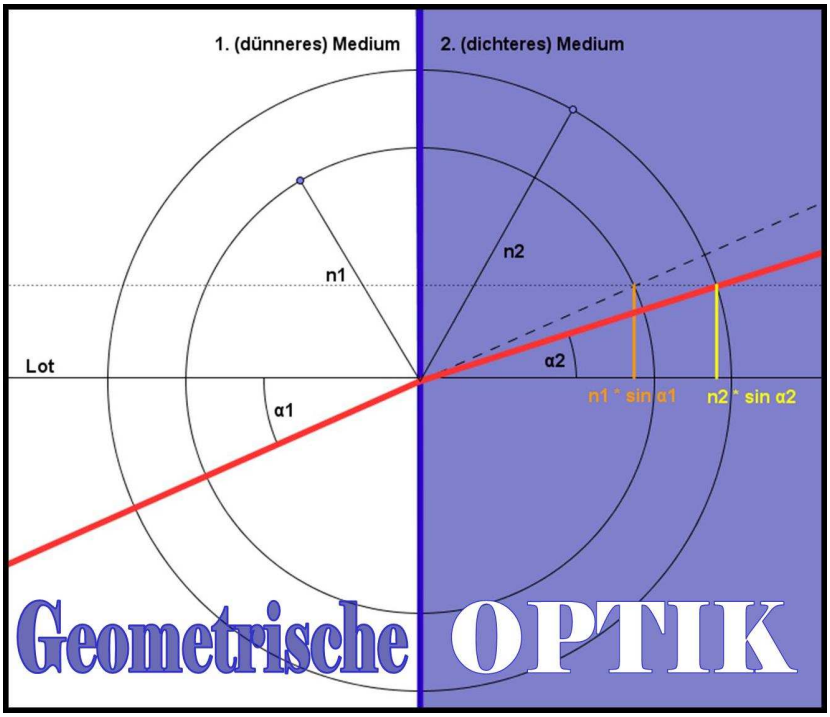


Abbildung 15



Verfolgung des Strahlenganges (ray tracing) in der geometrischen Optik

Prof. Peter Schöpf

Modellierer:
Petra Erdely
Susanne Guidassoni
Dario Kaylani
Dino Mehic
Julia Nitsch
Mattias Schönberger

INHALT

PROBLEMSTELLUNG

Das Wissen um den Strahlengang ist ein entscheidendes Element für den Bau von optischen Systemen wie zum Beispiel Brillen, Kontaktlinsen, Teleskopen oder Kameras. Aber auch bei Computerspielen und computeranimierten Filmen ist die Berücksichtigung des Strahlenganges von großer Bedeutung (ray tracing).

Im Rahmen der Modellierungswoche setzen wir uns nun zum Ziel, die Mathematik mit der Physik zu verbinden und in die Welt der Strahlen einzutauchen. Wir wollen die Reflexion bzw. Brechung von Lichtstrahlen an verschiedenen Grenzflächen geometrisch darstellen, und exakt und näherungsweise berechnen.

Die Aufgabe zerfällt in drei Teilaufgaben:

~ Berechnung des Auftreffpunktes eines Strahles auf eine Oberfläche.

~ Festlegung des geeigneten Normalenvektors an die Oberfläche im Auftreffpunkt.

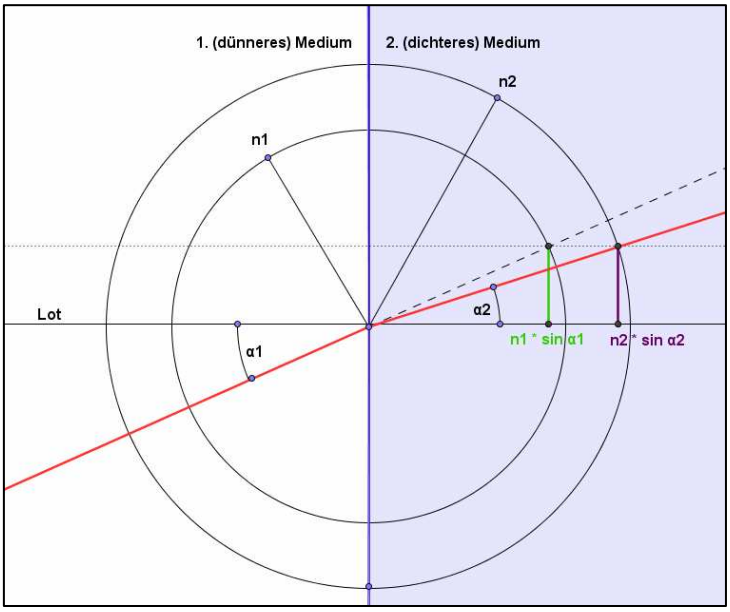
~ Berechnung des Richtungsvektors des reflektierten bzw. gebrochenen Strahls.

ERGEBNISSE

BRECHUNG
eines Lichtstrahls

1 Darstellung der Brechung eines Lichtstrahls mit GeoGebra

1.1 Das Konstruktionsprinzip



In der nebenstehenden Grafik ist die Brechung eines von links kommenden Lichtstrahls an einer Ebene dargestellt. Der Lichtstrahl fällt von einem optisch dünneren Medium mit dem Brechungsindex n_1 in ein optisch dichteres Medium mit dem Brechungsindex n_2 ein.

Zur Konstruktion verwenden wir das Brechungsgesetz nach Snellius, welches Folgendes besagt:

$$\frac{\sin \alpha_1}{\sin \alpha_2} = \frac{n_2}{n_1}$$

α_1, α_2..... Einfalls- bzw. Ausfallswinkel
n_1, n_2..... Brechungsindizes

Wir konstruieren zwei konzentrische Kreise mit den Radien n_1 und n_2 und schneiden den ungebrochenen Strahl mit dem Kreis vom Radius n_1. Unter Anwendung des Brechungsgesetzes ist es uns anschließend möglich, den Ausfallswinkel α_2 exakt zu konstruieren.

Wann wird ein Strahl nun zum Lot gebrochen, wann vom Lot?
Aus dem Physikunterricht wissen wir, dass ein Strahl, der in ein optisch dichteres Medium eintritt, grundsätzlich zum Lot gebrochen wird. Dies wird auch in der obenstehenden Zeichnung ersichtlich. Begründen können wir dies, indem wir das Brechungsgesetz nach Snellius folgendermaßen umformen:

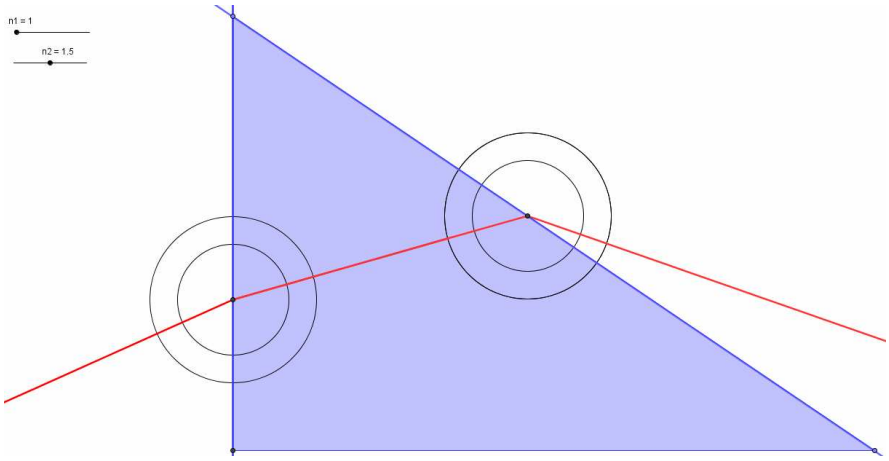
$$\sin \alpha_2 = \frac{n_1}{n_2} \sin \alpha_1$$

Fällt der Lichtstrahl von einem optisch dünneren Medium in ein optisch dichteres ein, dann ist $n_1<n_2$. Es gilt $\frac{n_1}{n_2}<1 \Rightarrow \alpha_2 < \alpha_1$.
Der Strahl wird zum Lot gebrochen.

Fällt der Lichtstrahl von einem optisch dichteren Medium in ein optisch dünneres ein, dann ist $n_1>n_2$. Es gilt $\frac{n_1}{n_2}>1 \Rightarrow \alpha_2 > \alpha_1$.
Der Strahl wird vom Lot gebrochen.

1.2 Brechung am Keil

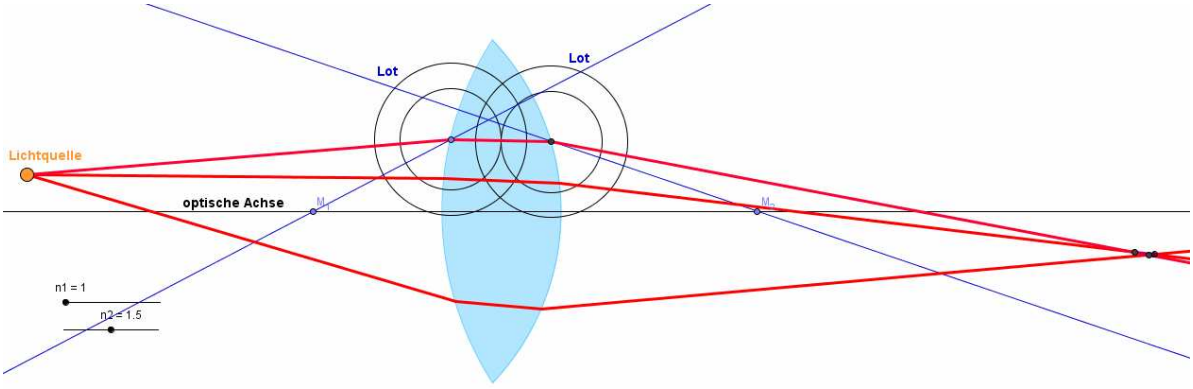
Nach dem oben beschriebenen Konstruktionsprinzip verfolgen wir mit GeoGebra den Strahlengang eines Lichtstrahls durch einen Keil.
Der Lichtstrahl trifft in diesem Beispiel aus der Luft mit dem Brechungsindex n_1=1 auf einen Glaskörper mit dem Brechungsindex n_2=1,5, durchquert ihn und tritt an dessen Rückseite wieder in die Luft ein. Es gibt also zwei Grenzflächen, wir müssen die Konstruktion zweimal ausführen.



1.3 Brechung an einer Bikonvexlinse

Nachdem wir die Konstruktion an ebenen Flächen ausreichend erprobt haben, wagen wir uns nun auch an gebogene Grenzflächen heran. Wir untersuchen die Brechung an einer bikonvexen Linse. Als Medien haben wir wieder Luft und Glas gewählt, der Strahl tritt also von einem optisch dünneren in ein optisch dichteres Medium ein, und anschließend wieder in das optisch dünnere aus.
Bei der Konstruktion müssen wir beachten, dass das Lot im Falle einer Linsenfläche keine Parallele zur optischen Achse mehr ist. Das Lot wird jetzt als Gerade konstruiert, die einerseits durch den Auftreffpunkt des Strahls an der Linse geht, und andererseits durch den Mittelpunkt des Kreises, von dem wir, um unsere Zeichnung zu vereinfachen, einen Teil des Bogens als Grenzfläche gewählt haben.

Sobald wir das Lot konstruiert haben, können wir die zwei Kreise mit den Brechungsindizes als Radien um den Auftreffpunkt zeichnen, und die weitere Konstruktion erfolgt gleich wie an der ebenen Fläche.



Im Physikunterricht wurde uns beigebracht, dass sich Strahlen, welche von einer punktförmigen Lichtquelle ausgehen und auf eine bikonvexe Linse treffen, hinter der Linse in einem Punkt exakt schneiden. Anhand unserer Grafik haben wir allerdings festgestellt, dass dies nur eine Vereinfachung darstellt. Die Strahlen schneiden sich zwar in einem sehr kleinen Bereich, allerdings nicht exakt in einem Punkt.

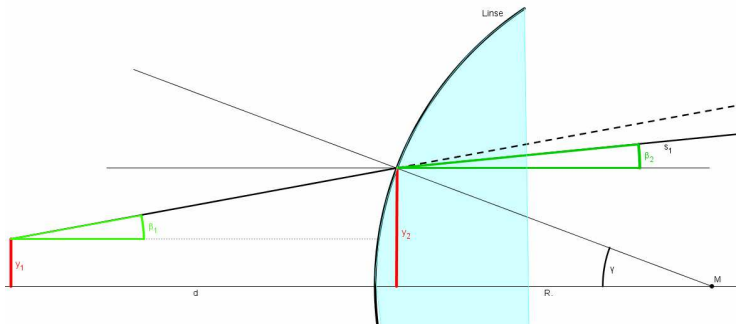
2 Näherungsweise Berechnung des Strahlenganges durch eine Linsenfläche

2.1 Bedingungen für eine näherungsweise Berechnung

Nach der exakten Konstruktion des Strahlenganges durch eine Linsenfläche beschäftigen wir uns nun mit einer genäherten Berechnung. Die Voraussetzungen für die Näherung sind, dass der betrachtete Lichtstrahl annähernd parallel zur optischen Achse verläuft, sowie keine allzu große Distanz zu Selbiger aufweist. Denn die Auftreffhöhe $y_2 = y_1 + d\beta_1$ sollte nicht größer als 1/10 des Krümmungsradius der Linsenfläche sein.

2.2 Was wir berechnen wollen

Wir sind jetzt an einem Verfahren interessiert, mit dem wir die Auftreffhöhe y_2 (rot) und den Anstiegswinkel β_2 (grün) des gebrochenen Strahls näherungsweise berechnen können.

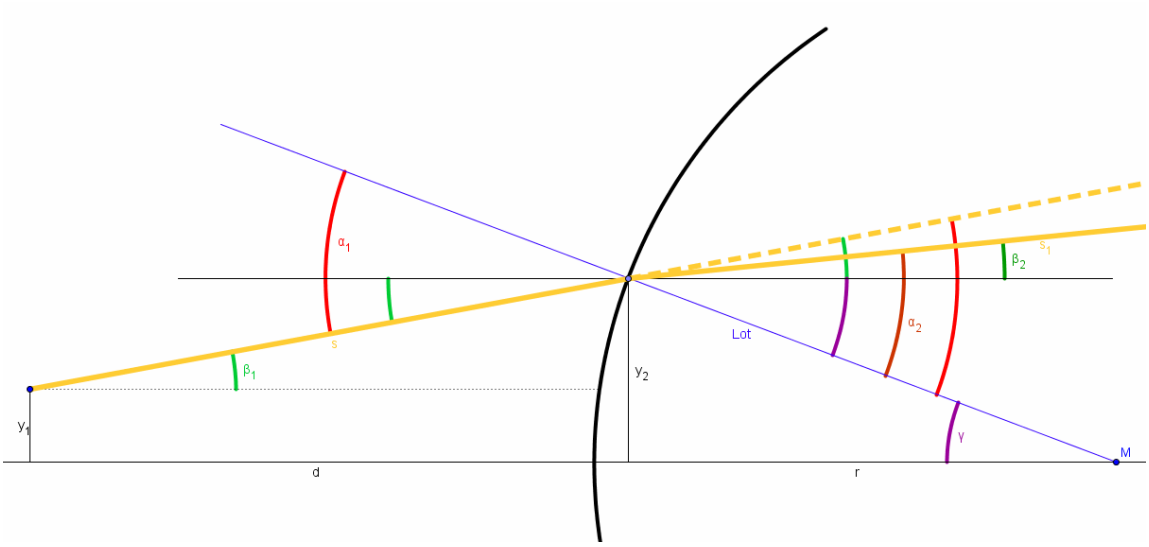


2.3 Unsere Ausgangsdaten

Wir betrachten nun eine Lichtquelle in der Höhe y_1 und in der Distanz d links von der konvexen Linsenfläche. Von ihr gehe ein Lichtstrahl mit dem Anstiegswinkel β_1 in Richtung der Linsenfläche aus. Ferner seien der Krümmungsradius R der Linsenfläche und die Brechungsindizes n_1 und n_2 der Medien links und rechts gegeben.

2.4 Die näherungsweise Berechnung

Wir fertigen eine detailliertere Skizze an:



Als erstes müssen wir die Auftreffhöhe y_2 vom Strahl auf die Linse mit der Querungsmatrix berechnen.

Wie in der Skizze ersichtlich, gilt für $y_2 \approx y_1 + (\tan \beta_1) d$.
Aufgrund der oben beschriebenen Bedingungen für eine näherungsweise Berechnung dürfen wir $\beta_2 \approx \sin \beta_2 \approx \tan \beta_2$ setzen.
Die Höhe y_2 lässt sich nun wie folgt berechnen:

$$y_2 = y_1 + \beta_1 d \, .$$

Da sich am Winkel selbst nichts ändert, bleibt $n_1 \beta_1 = n_1 \beta_1$.

Jetzt müssen wir die zwei Gleichungen auf eine einheitliche Form bringen, damit wir sie in eine Matrixgleichung umwandeln können:

$$n_1 \beta_1 = 1 * n_1 \beta_1 + 0 * y_1$$

$$y_2 = \frac{d}{n_1} * n_1 \beta_1 + 1 * y_1$$

Daraus ergibt sich die Querungsmatrix Q:= $\begin{pmatrix} 1 & 0 \\ \frac{d}{n_1} & 1 \end{pmatrix}$,

und somit

$$\begin{pmatrix} 1 & 0 \\ \frac{d}{n_1} & 1 \end{pmatrix} * \begin{pmatrix} n_1\beta_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} n_1\beta_1 \\ y_2 \end{pmatrix}.$$

Als nächstes berechnen wir den Anstiegswinkel des gebrochenen Strahls mithilfe einer Brechungsmatrix.

Es gilt das Brechungsgesetz: $\frac{\sin\alpha_2}{\sin\alpha_1} = \frac{n_1}{n_2}$; näherungsweise: $\frac{\alpha_2}{\alpha_1} = \frac{n_1}{n_2} \Rightarrow n_2\alpha_2 = n_1\alpha_1$

Aus $\alpha_1 = \beta_1 + \gamma$ ergibt sich: $\alpha_1 n_1 = n_1(\beta_1 + \gamma) = n_1\beta_1 + n_1\gamma = n_1\beta_1 + \frac{n_1 y_2}{R}$

Der Zeichnung kann man entnehmen, dass $\alpha_2 = \gamma + \beta_2$.

Dies ergibt, in das Brechungsgesetz eingesetzt: $\alpha_1 \frac{n_1}{n_2} = \gamma + \beta_2$.

Durch Umformen erhält man: $\beta_2 = \alpha_2 - \gamma = \frac{n_1}{n_2}\left(\beta_1 + \frac{\beta_1 d + y_1}{R}\right) - \frac{\beta_1 d + y_1}{R}$.

Wir ersetzen $y_1 + \beta_1 d$ nun durch y_2: $\beta_2 = \frac{n_1}{n_2}\left(\beta_1 + \frac{y_2}{R}\right) - \frac{y_2}{R}$.

Zur Vereinfachung multiplizieren wir die Formel mit n_2 und lösen die Klammer auf:

$$n_2\beta_2 = n_1\beta_1 + \frac{y_2 n_1}{R} - \frac{y_2 n_2}{R}.$$

Jetzt heben wir y_2 heraus, um wieder auf eine einheitliche Form zu kommen:

$$n_2\beta_2 = n_1\beta_1 + y_2\left(\frac{n_1 - n_2}{R}\right)$$

An der Höhe ändert sich nichts mehr, also: $y_2 = y_2$

$$n_2\beta_2 = 1 * n_1\beta_1 + \frac{n_1 - n_2}{R} * y_2$$
$$y_2 = 0 * n_1\beta_1 + 1 * y_2$$

Hieraus ergibt sich die Brechungsmatrix $B:=\begin{pmatrix} 1 & \frac{n_1-n_2}{R} \\ 0 & 1 \end{pmatrix}$

Wir halten fest:

~ mit der Querungsmatrix Q berechnen wir die neue Höhe y_2:

$$\begin{pmatrix} 1 & 0 \\ \frac{d}{n_1} & 1 \end{pmatrix}\begin{pmatrix} n_1\beta_1 \\ y_1 \end{pmatrix}=\begin{pmatrix} n_1\beta_1 \\ y_2 \end{pmatrix}$$

~ mit der Brechungsmatrix B berechnen wir den neuen Winkel β_1:

$$\begin{pmatrix} 1 & \frac{n_1-n_2}{R} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} n_1\beta_1 \\ y_2 \end{pmatrix}=\begin{pmatrix} n_2\beta_2 \\ y_2 \end{pmatrix}$$

Um den Endvektor E zu erhalten, multiplizieren wir den Ausgangsvektor A zuerst mit der Querungsmatrix Q, und anschließend mit der Brechungsmatrix B:

$$\begin{pmatrix} 1 & \frac{n_1-n_2}{R} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ \frac{d}{n_1} & 1 \end{pmatrix}\begin{pmatrix} n_1\beta_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1+\frac{d(n_1-n_2)}{n_1-R} & n_1-n_2 \\ \frac{d}{n_1} & 1 \end{pmatrix}\begin{pmatrix} n_1\beta_1 \\ y_1 \end{pmatrix}=\begin{pmatrix} n_2\beta_2 \\ y_2 \end{pmatrix}$$

B Q A B*Q A E

Zusammenfassend kann man sagen, dass die Rechnungen durch die 2x2-Matrizen übersichtlicher und dadurch vereinfacht wurden. Außerdem ergibt sich für uns der Vorteil, dass wir jetzt auch mit MatLab arbeiten können, das selbst mit Matrizen rechnet.

3 Näherungsweise Berechnung des Strahlenganges durch eine Linse

3.1 Aufgabenstellung

Nachdem wir uns mit MatLab etwas vertraut gemacht haben, berechnen wir den Strahlengang durch eine Bikonvexlinse. Der Strahl beginnt in der Entfernung d_1 links von der Linse und trifft in der Entfernung d_3 rechts von der Linse auf einen Bildschirm. Der Strahl startet in der Höhe y_1 mit dem Anstiegswinkel β_1 und landet auf dem Bildschirm in der Höhe y_4 und mit dem Anstiegswinkel β_4.

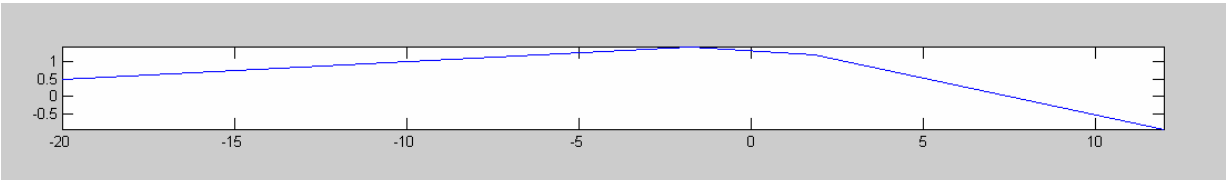
Bevor wir uns MatLab widmen, überlegen wir noch, dass man den Endvektor E aus der Multiplikation aller Matrizen mit dem Ausgangsvektor A erhält:

$$E=(Q_3*B_2*Q_2*B_1*Q_1)*A$$

3.2 Programmcodes in Matlab

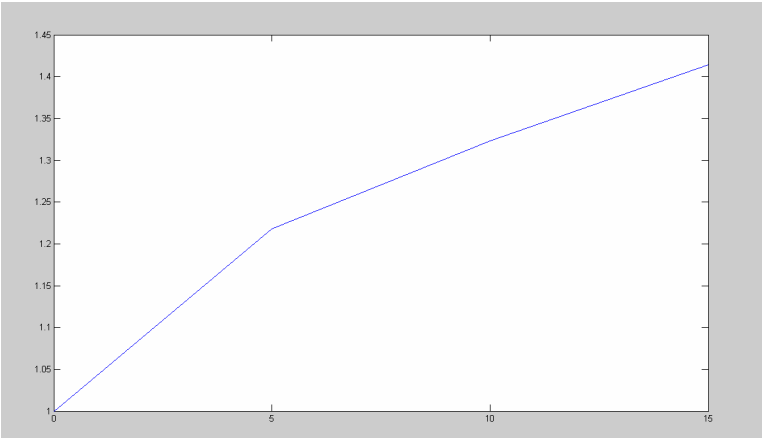
Wir geben folgende Befehle ein:
(mit % gekennzeichnete Befehle zählen als Kommentar)

```
%X-Werte:
x1=[-20]%Startstelle
x2=[-1.77]%1.Brechung
x3=[1.87]%2.Brechung
x4=[12]%Bildschirm

%Brechungsindizes
n1=[1]%1. und 3.Medium
n2=[1.5]%2.Medium

%weitere gegebene Größen
y1=[0.5]%Starthöhe (möglichst achsennah, wegen Näherung)
beta1=(pi/62.5)%Startwinkel
R1=5%1.Radius (Achtung nicht zu groß wählen!!! max.: 1/10  von d)
R2=5%2.Radius

%Rechnungen
d1=x2-x1
d2=x3-x2
d3=x4-x3

A=[n1*beta1;y1]
Q1=[1,0;d1/n1,1]%Erste Querungsmatrix durch Luft
B1=[1,(n1-n2)/R1;0,1]% Erste Brechungsmatrix
Q2=[1,0;d2/n2,1]% Zweite Querungsmatrix durch die Linse
B2=[1,-(n2-n1)/R2;0,1]% Zweite Brechungsmatrix
Q3=[1,0;d3/n1,1]% Dritte Querungsmatrix durch Luft

% Zwischenergebnisse
I=[Q1*S]
II=[B1*Q1*S]
III=[Q2*B1*Q1*S]
IV=[B2*Q2*B1*Q1*S]
V=[Q3*B2*Q2*B1*Q1*S]

% die verschiedenen Auftreffhöhen
y2=I(2)%an der linken Linsenfläche
y3=III(2)%an der rechten Linsenfläche
y4=V(2)%am Bildschirm

X=[x1,x2,x3,x4]
Y=[y1,y2,y3,y4]

%Grafische Darstellung
plot(X,Y)%Ausgabe
axis equal tight%gleichen Achsenmaßstab wählen
```



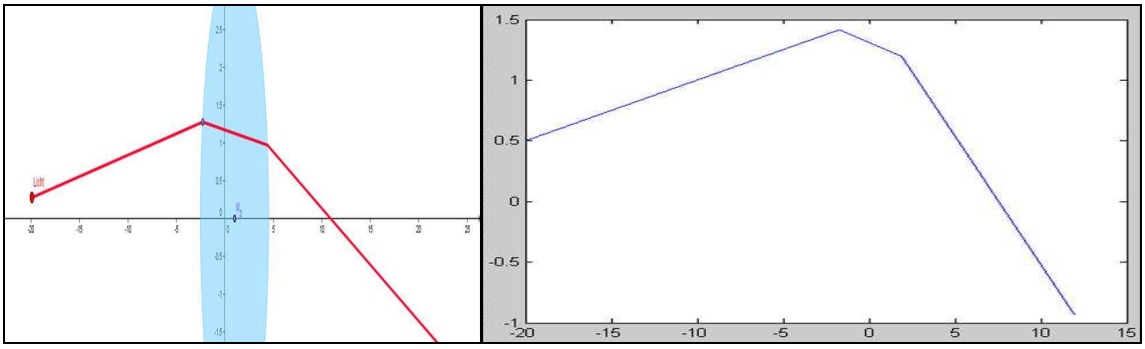
oder kürzer in Vektorschreibweise:

```
X=[0,5,10,15] %Die jeweilig eingetragen Werte bestimmen die X-Koordinaten der Position des
Strahls am Start, Eintritts-,Austrittsstelle und Auftreffpunkt am Schirm.
N=[1,1.5,1] % Brechungsindizes
R=[50,50] % Radien der Linse
D=[X(2)-X(1),X(3)-X(2),X(4)-X(3)] %Abstände zwischen den Punkten X
beta1=(2.5/180)*3.1416 % Anstiegswinkel in x1,y1
y1=1

A=[N(1)*beta1;y1]

Q1=[1,0;D(1)/N(1),1]
B1=[1,(N(1)-N(2))/R(1);0,1]
Q2=[1,0;D(2)/N(2),1]
B2=[1,-(N(2)-N(3))/R(2);0,1]
Q3=[1,0;D(3)/N(3),1]

P1=Q1*A
P2=Q2*B1*P1

E=Q3*B2*Q2*B1*Q1*A

Y=[y1,P1(2),P2(2),E(2)]


plot(X,Y)
```



3.3 Vergleich eines MatLab-Plots mit einer Strahlengangskonstruktion in GeoGebra

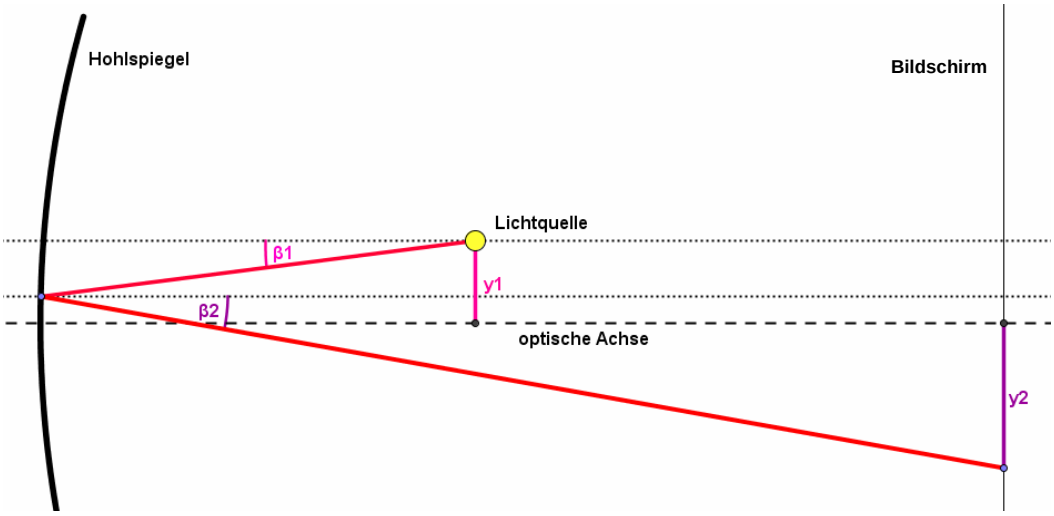
Wir setzen für die benötigten Ausgangsdaten feste Werte ein und überprüfen unsere berechneten Näherungswerte mit einer Strahlengangskonstruktion in GeoGebra:



GeoGebra MatLab

REFLEXION
eines Lichtstrahls

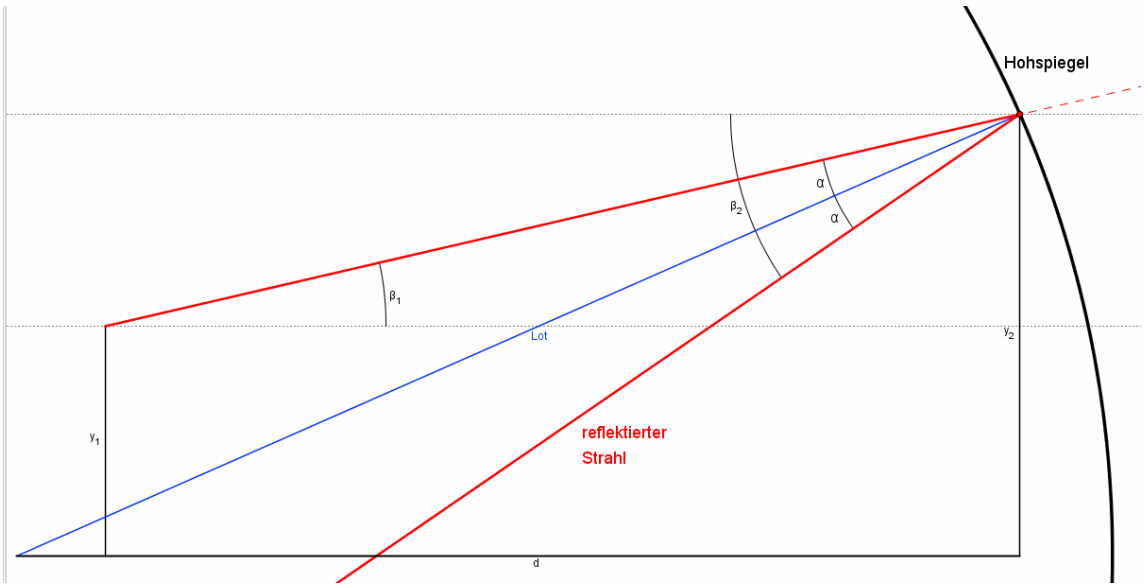
4 Berechnung der Reflexion eines Lichtstrahls am Hohlspiegel



4.1 Ausgangsdaten

Ein Lichtstrahl startet mit der Höhe y_1 und dem Winkel β_1 in Richtung Hohlspiegel und trifft dort mit der Höhe y_2 auf. Der Hohlspiegel ist mit seinem Radius *R* festgelegt. Der Abstand von Startpunkt und Auftreffpunkt ist mit der Länge *d* festgelegt. Da kein Übergang in ein anderes Medium stattfindet, gibt es nur einen Brechungsindex *n*.

4.2. Die näherungsweise Berechnung von y_2 und β_2



Als erstes berechnen wir die Höhe y_2 des Auftreffpunkts des Strahls am Hohlspiegel.

Wie in der Skizze ersichtlich, gilt für $y_2 \approx y_1 + \tan \beta_1 d$.

Aufgrund der beschriebenen Bedingungen für eine näherungsweise Berechnung dürfen wir $\beta_2 \approx \sin \beta_2 \approx \tan \beta_2$ setzen.

Die Höhe y_2 lässt sich nun wie folgt berechnen:

$$y_2 = y_1 + \beta_1 d .$$

Am Winkel selbst ändert sich nichts: $n\beta_1 = n\beta_1$

Wir bringen die beiden Gleichungen auf eine einheitliche Form, um sie anschließend in eine Matrix umformen zu können:

$$n\beta_1 = 1 * n\beta_1 + 0 * y_1$$
$$y_2 = \frac{d}{n} * n\beta_1 + 1 * y_1$$

Hieraus ergibt sich die Querungsmatrix Q:= $\begin{pmatrix} 1 & 0 \\ \frac{d}{n} & 1 \end{pmatrix}$

Als nächstes berechnen wir den Einfallswinkel α_1. Dazu verschieben wir die optische Achse in Gedanken auf die Höhe von y_1. Es entsteht ein neues Dreieck mit den Winkeln $\alpha 1$, $\beta 1$ und dem Gegenwinkel von γ.

$$\alpha_1 = \pi - \beta_1 - (\pi - \gamma) = \gamma - \beta_1$$

Aus der Skizze kann man leicht erkennen, dass gilt:

$$\beta_2 = \beta_1 + 2\alpha_1$$

Für eine einheitliche Form multiplizieren wir die Gleichung mit dem Brechungsindex:

$$n\beta_2 = n\beta_1 + 2\alpha_1 n$$

Wir ersetzen α_1: $n\beta_2 = n\beta_1 + 2(-2\beta_1 + 2\gamma)n = -\beta_1 n + 2\gamma n$.

Dann wird γ ersetzt: $n\beta_2 = -\beta_1 n + 2n\left(\frac{-y_2}{R}\right)$

$$n\ \beta_2 = -1 * n\beta_1 + y_2 * -\frac{2n}{R}$$
$$y_2 = 0 * n\beta_1 + y_2 * 1$$

Daraus ergibt sich die Reflexionsmatrix R:= $\begin{pmatrix} -1 & \frac{-2n}{R} \\ 0 & 1 \end{pmatrix}$

Wir fassen zusammen:

$$\begin{pmatrix} 1 & 0 \\ \frac{d}{n} & 1 \end{pmatrix} \begin{pmatrix} n\beta_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} n\beta_1 \\ y_2 \end{pmatrix}$$
$$\begin{pmatrix} -1 & \frac{-2n}{R} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} n\beta_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} n\beta_2 \\ y_2 \end{pmatrix}$$

Den Endvektor E erhält man, indem man den Ausgangsvektor A erst mit der Querungsmatrix Q und dann noch mit der Reflexionsmatrix R multipliziert:

$$\begin{pmatrix} -1 & \frac{-2n_1}{R} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \frac{d}{n_1} & 1 \end{pmatrix} \begin{pmatrix} n_1\beta_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} n_1\beta_2 \\ y_2 \end{pmatrix}$$

R Q A E

EXAKTE VEKTORIELLE BERECHNUNG der Reflexion und Brechung eines Lichtstrahls an einer Ebene

5 Koordinatenfreie Berechnung des Auftreffpunktes eines Lichtstrahles auf einer Grenzfläche

Zur koordinatenfreien Berechnung des Auftreffpunktes eines Lichtstrahles auf einer Grenzfläche verwenden wir Vektoren. Diese Art der Berechnung kann auch im dreidimensionalen Raum angewandt werden, benötigt jedoch viel mehr Eingangsdaten und Rechenfertigkeit.

5.1 Die Berechnung des Auftreffpunktes an einer Ebene

Im Falle der Ebene sind grundsätzlich zwei Ausgänge möglich. Es kann einen Auftreffpunkt geben, sofern der Richtungsvektor des Strahls auf die Ebene weist. Tut er dies nicht, gibt es keinen Auftreffpunkt.

Zur Berechnung des Auftreffpunkts stellen wir zunächst den Lichtstrahl in Parameterform dar:

$$S_{\vec{a},\vec{r}} = \{\vec{a} + t\vec{r} \mid t \geq 0\}$$

wobei $\vec{a}$ der Anfangspunkt und $\vec{r}_1$ ein Einheitsvektor ist, der die Strahlrichtung angibt.

Die Grenzebene geben wir in Gleichungsform an:

$$E_{\vec{n},p} = \{\vec{x} \in R^3 \mid \vec{x} \cdot \vec{n} = p\}$$

wobei $\vec{n}$ ein Einheitsvektor ist, der normal auf die Ebene steht, und *p* eine Zahl, deren Absolutbetrag den Abstand der Ebene vom Ursprung angibt.

Wir schneiden den Strahl mit der Ebene:

$$(\vec{a} + t\vec{r}) \cdot \vec{n} = p \,.$$

Falls $\vec{r} \cdot \vec{n} \neq 0$ ist, bekommen wir für *t* den Wert

$$t = \frac{p - \vec{n} \cdot \vec{a}_1}{\vec{n} \cdot \vec{r}} \,.$$

t liefert nur dann den Auftreffpunkt, wenn $t \geq 0$ ist.

5.1 Die Berechnung des Auftreffpunktes an einer Sphäre

Hier sind zwei Fälle zu unterscheiden:
- Der Strahl geht an der Sphäre vorbei.
- Der Strahl trifft die Sphäre in einem oder zwei Punkten.

Trifft der Strahl die Sphäre in zwei Punkten, so ist als Auftreffpunkt der Punkt zu wählen, der näher an der Lichtquelle liegt.

Wir stellen den Lichtstrahl in Parameterform dar:

$$S = \vec{a} + t\vec{r}_1 ,$$

$t>0$, $\vec{a}$ und $\vec{r}_1$ wie früher.

Die sphärische Grenzfläche geben wir in Gleichungsform an:

$$(\vec{x} - \vec{m}) \cdot (\vec{x} - \vec{m}) = R^2 ,$$

wobei $\vec{m}$ der Kugelmittelpunkt und R der Kugelradius ist.

Wir schneiden den Strahl mit der Sphäre:

$$R^2 = \left((\vec{a} + t\vec{r}) - \vec{m}\right) * \left((\vec{a} + t\vec{r}) - \vec{m}\right)$$

$$R^2 = t(2(\vec{r} \cdot \vec{a}) - 2(\vec{r} \cdot \vec{m})) + t^2\vec{r}^2 + \vec{a}^2 - 2\left(\vec{m} \cdot \vec{a}\right) + \vec{m}^2$$

$$R^2 = \vec{a}^2 + t\left(\vec{r} \cdot \vec{a}\right) - \vec{m} \cdot \vec{a} + t\left(\vec{r} \cdot \vec{a}\right) + t^2\vec{r}^2 - t(\vec{r} \cdot \vec{m}) - \vec{m} \cdot \vec{a} - t(\vec{r} \cdot \vec{m}) + \vec{m}^2$$

$$t_{1,2} = \vec{r} \cdot (\vec{a} - \vec{m}) \pm \sqrt{\left(\vec{r} \cdot (\vec{a} - \vec{m})\right)^2 - \left((\vec{a} - \vec{m})^2 - R^2\right)}$$

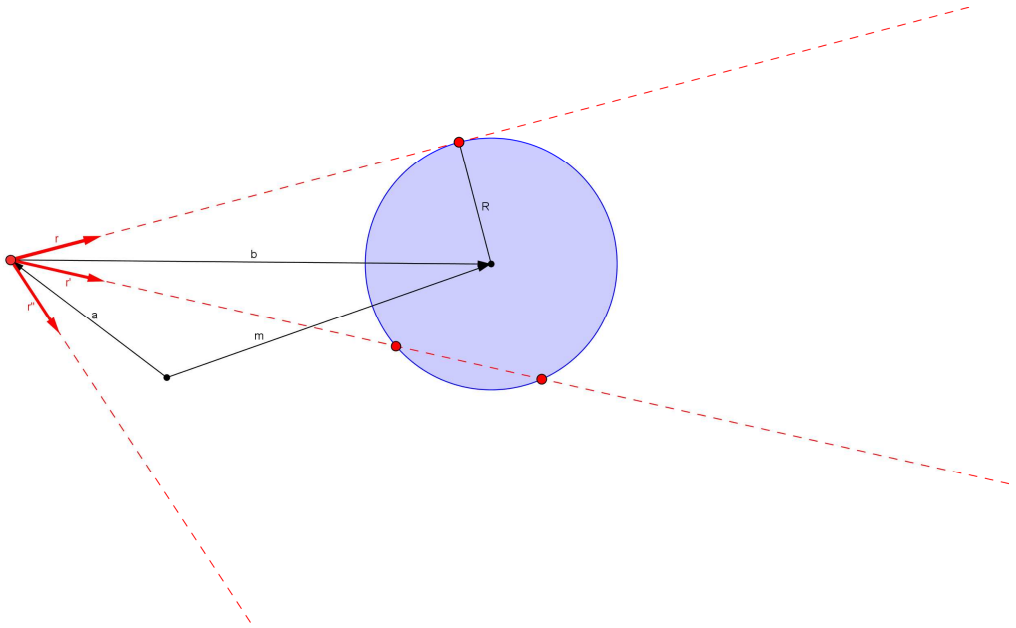
Für den Radikanden gibt es drei Möglichkeiten:

$$(\vec{r} \cdot (\vec{a} - \vec{m}))^2 + (\vec{a} - \vec{m})^2 - R^2$$

< 0 … keine Lösung
= 0 … eine Lösung
> 0 … zwei Lösungen

Im Detail ergeben sich folgende Fälle:

1. $t_1<0$; $t_2<0$ ➡ der Strahl trifft die Kugel nicht
2. $t_1>0$; $t_2<0$ ➡ der Strahl trifft die Kugel und es entsteht ein Schnittpunkt, da mindestens ein t positiv ist
3. $t_1<0$; $t_2>0$ ➡ siehe 2.Fall
4. $t_1>0$; $t_2>0$ ➡ mit dem kleinerem t lässt sich der Schnittpunkt wiederum berechnen



6 Koordinatenfreie Festlegung des Flächennormalenvektors

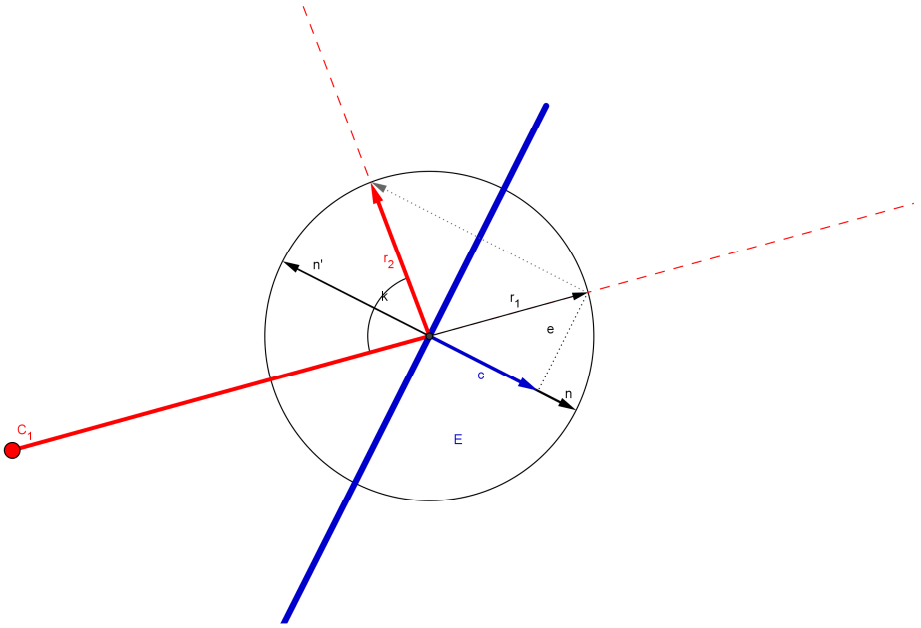
Im Auftreffpunkt gibt es zwei normierte Normalvektoren der Grenzfläche. Für die weiteren Formeln ist derjenige zu wählen, der in denselben Halbraum wie der Richtungsvektor $\vec{r}_1$ zeigt, d.h. bei dem $\vec{n} \cdot \vec{r}_1 > 0$ ist.

Das ist $\vec{e}_1 := \dfrac{\vec{r}_1 \cdot \vec{n}}{\left|\vec{r}_1 \cdot \vec{n}\right|} \vec{n}$.

7 Koordinatenfreie Berechnung des Richtungsvektors des reflektierten (gebrochenen) Strahls

Ist der einlaufende Strahl durch Anfangspunkt $\vec{a}$ und Anfangsrichtung $\vec{r}_1$ gegeben, und die Ebene durch die Gleichung $\vec{x} \cdot \vec{n} = p$
(die Sphäre durch die Gleichung $(\vec{x} - \vec{m}) \cdot (\vec{x} - \vec{m}) = R^2$),
dann kann man nach der Berechnung des Auftreffpunktes $\vec{q}$ und des Normalenvektors $\vec{e}_1$ den Richtungsvektor $\vec{r}_2$ des reflektierten (gebrochenen) Strahls durch die gegebenen Daten ausdrücken.

7.1 Die Berechnung des Richtungsvektors des reflektierten Strahls



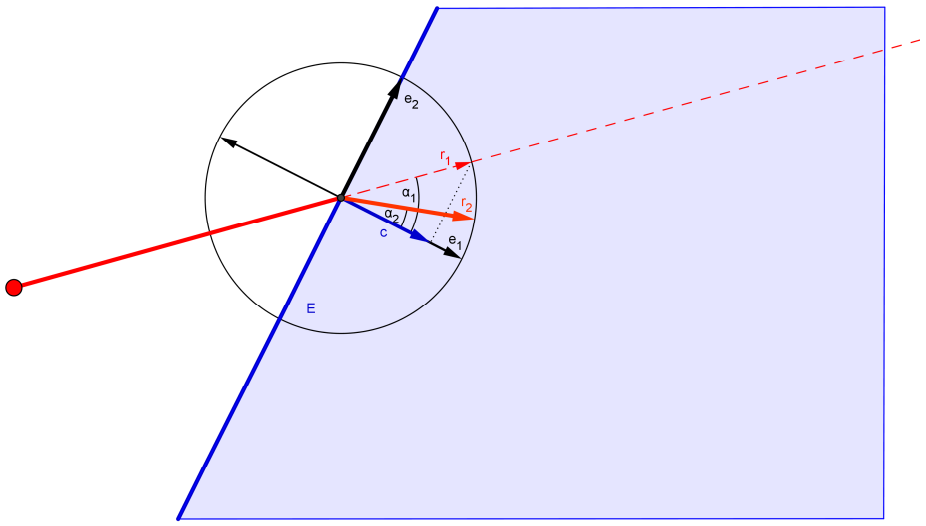
Wie in der obigen Skizze ersichtlich, können wir den reflektierten Vektor als Linearkombination darstellen:

$$r_2 = \vec{r}_1 - 2\vec{c}$$

$$\vec{c} = (\vec{r}_1 \cdot \vec{n})\vec{n}$$

$$\boxed{\vec{r}_2 = \vec{r}_1 - 2(\vec{r}_1 \cdot \vec{n})\vec{n}}$$

7.2 Die Berechnung des Richtungsvektors des gebrochenen Strahls



Um den Richtungsvektor eines gebrochenen Strahls zu berechnen sind folgende Eingangsdaten erforderlich:

$$\vec{r}_1 = (\cos\alpha_1)\vec{e}_1 + (\sin\alpha_1)\vec{e}_2$$

$$\vec{r}_2 = (\cos\alpha_2)\vec{e}_1 + (\sin\alpha_2)\vec{e}_2$$

Hierbei ist $\vec{e}_2$ der auf $\vec{e}_1$ senkrecht stehende Einheitsvektor, wie in der obigen Skizze angedeutet.

Bei der Berechnung ist der erste Schritt die Wahl des richtigen Normalvektors $\vec{e}_1$ der Ebene:

$$\vec{e}_1 = \frac{\vec{r}_1 \cdot \vec{n}}{|\vec{r}_1 \cdot \vec{n}|}\vec{n}$$

Um den Vektor $\vec{r}_1$ zu berechnen, sind neben dem Normalvektor auch Sinus und Kosinus des Einfalls- und des Austrittswinkels erforderlich:

$$\cos\alpha_1 = \vec{r}_1 \cdot \vec{e}_1$$

$$\sin\alpha_1 = \sqrt{1-\cos^2\alpha_1}$$

$$\sin\alpha_1 = \sqrt{1-(\vec{r}_1 \cdot \vec{e}_1)^2}$$

$$\frac{\sin\alpha_2}{\sin\alpha_1} = \frac{n_1}{n_2}$$ (Brechungsgesetz von Snellius)

$$\sin\alpha_2 = \frac{n_1}{n_2}\sin\alpha_1$$

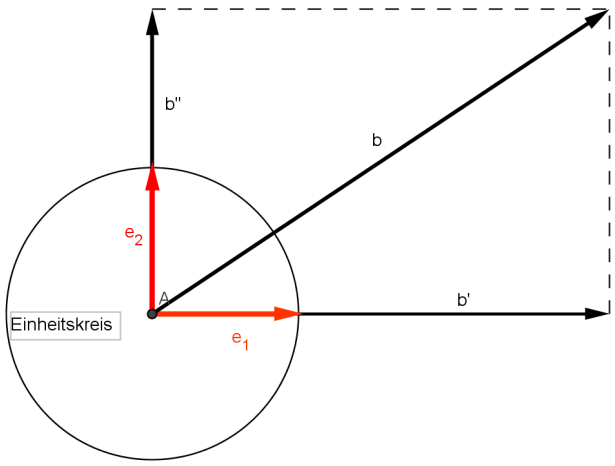
$$\sin\alpha_2 = \frac{n_1}{n_2}\sqrt{1-\cos^2\alpha_1}$$

$$\sin\alpha_2 = \frac{n_1}{n_2} \cdot \sqrt{1-(\vec{r}_1 \cdot \vec{e}_1)^2}$$

$$\cos\alpha_2 = \sqrt{1-\sin^2\alpha_2}$$

$$\cos\alpha_2 = \sqrt{1-\left(\frac{n_1}{n_2}\right)^2\left(1-(\vec{r}_1 \cdot \vec{e}_1)^2\right)}$$

Nun drücken wir noch den Ebenenvektor $\vec{e}_2$ mithilfe der Eingangsdaten($\vec{r}_1, \vec{e}_1$) aus.



$\vec{e}_1 \perp \vec{e}_2$ ($\vec{e}_1$ steht normal auf $\vec{e}_2$)

$\left(\vec{b} \cdot \vec{e}_1\right)\vec{e}_1 = \vec{b}'$ (Vektor $\vec{b}$ auf $\vec{e}_1$projizieren)

$\vec{b}'' = \vec{b} - \vec{b}'$ (Vektor $\vec{b}$ '' entsteht, indem man $\vec{b}$ ' von $\vec{b}$ abzieht)

$\vec{e}_2 = \dfrac{\vec{b}''}{\left|\vec{b}''\right|}$ (damit Vektor $\vec{e}_2$ entsteht, muss man $\vec{b}$ '' noch normieren)

$$\vec{e}_2 = \frac{\vec{b} - \left(\vec{b} \cdot \vec{e}_1\right)\vec{e}_1}{\left|\vec{b} - \left(\vec{b} \cdot \vec{e}_1\right)\vec{e}\right|} \Leftrightarrow \frac{\vec{r}_1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)\vec{e}_1}{\left|\vec{r}_1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)\vec{e}_1\right|}$$

Nachdem wir alles durch die Vektoren $\vec{r}_1$ und $\vec{e}_1$ ausgedrückt haben, können wir $\vec{r}_2$ wie folgt darstellen,

$$\vec{r}_2 = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 \left(1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)^2\right)}\vec{e}_1 + \frac{n_1}{n_2}\sqrt{1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)^2}\,\frac{\vec{r}_1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)\vec{e}_1}{\left|\vec{r}_1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)\vec{e}_1\right|}$$

oder wenn wir die Formel für $\vec{e}_2$ verwenden

$$\vec{r}_2 = \frac{n_1}{n_2}\vec{r}_1 + \left(\sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 \left(1 - \left(\vec{r}_1 \cdot \vec{e}_1\right)^2\right)} + \frac{n_1}{n_2}\left(\vec{r}_1 \cdot \vec{e}_1\right)\right)\vec{e}_1 \quad .$$

Betreuer:

Univ. Prof. Wolfgang Ring, Petra Katolnig;

Team:

Sara Basara, Stephan Bayer, Christian Burghard,
Michael Draxler, Gernot Holler, Julia Pichler;

Inhaltsverzeichnis

Bericht Projekt Spielstrategien

Problemstellung:

Fußball ist die populärste Sportart in Europa, in der realen, ebenso wie in der virtuellen Welt. Im Fernsehen angeschaut, im Freien oder auch am PC gespielt hat diese Sportart bereits jeder von uns einmal, doch niemand wäre je auf die Idee gekommen Spielsituationen selbst zu modellieren.

Wir bekamen die Aufgabe klassische Spielsituationen, die immer wieder im Laufe eines Fußballspiel auftreten, zu simulieren: Ein Stürmer steht einem Verteidiger gegenüber, zwei Stürmer versuchen an zwei Verteidigern vorbeizukommen und zwei Stürmer versuchen zwei Verteidiger zu überspielen.

Da wir nur wenig Zeit zur Verfügung hatten mussten wir uns grundlegende Gedanken über die Spielstrategie machen. Solche wie, dass der Verteidiger die Position und momentane Laufrichtung des Stürmers annimmt, um ihn von einer günstigen Schussposition fernzuhalten, bzw. um ihn unter Druck zu setzen und den Ball abzunehmen. Dem Stürmer geht es nicht anders. Auch er muss entscheiden wie sein nächster Schritt aussieht, nur dass sein Ziel natürlich dem des Verteidigers entgegengesetzt ist. Er möchte sich dem Tor nähern, um eine gute Abschussmöglichkeit zu bekommen. Antworten auf solche Fragen nach dem richtigen taktischen Verhalten fanden wir einerseits in den Lehrbüchern über Fußball, andererseits haben wir mathematische Überlegungen herangezogen, um Spielabläufe zu simulieren oder um optimale Strategien zu finden.

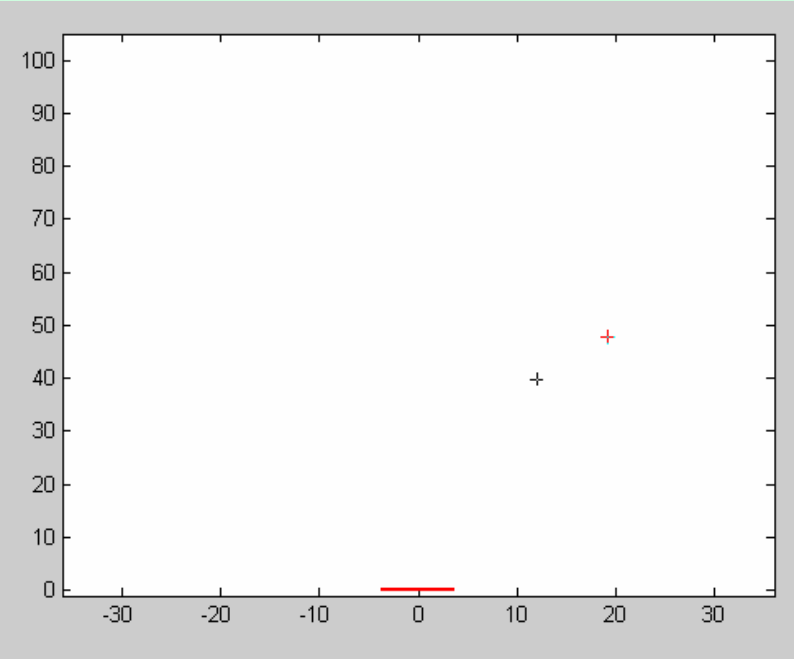
Der Vorschlag, MATLAB als unser Programmierprogramm zu verwenden wurde einheitlich angenommen. Bevor wir dieses jedoch verwenden konnten gingen wir mehrere Strategien durch und experimentierten mit verschiedensten Funktionen im MATLAB, um dem Umgang mit diesem zu erlernen.

Als wir uns das Wissen und die Fertigkeiten angeeignet hatten, das für die weiteren Schritte in der Modellierung von Nöten war, begannen wir mit einer einfachen Eins-gegen- Eins- Situation ohne Ball und erweiterten das Modell schrittweise: Mehrere Verteidiger bzw. Angreifer, Miteinbeziehen des Balles, Beschränkungen an die Beweglichkeit der Spieler etc.

Nach dem Erstellen von Entwürfen und dem Ausprobieren verschiedenster Strategien konnten wir uns ein Ziel setzten, dass eine Zwei- gegen Zwei-Situation beinhaltete.

Vorgabe:

Wie alles begann…



Das ist ein Ausschnitt aus der Simulation, die wir am Anfang der Modellierungswoche vorgesetzt bekamen. Aus dieser ersten Simulation entstanden verschiedene Ideen, eine künstliche Intelligenz zu konstruieren.

Seine vereinfachte Strategie funktionierte nach diesem Schema:

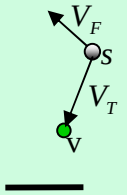
1) Strategie des Stürmers:

Die Strategie des Stürmers ist so angelegt, dass er in erster Linie direkt auf das Tor zuläuft und vor einem Verteidiger davon läuft, falls dieser in seine Nähe gelangt. Mathematisch lässt sich die Strategie des Stürmers wie folgt darstellen:

1. Komponente: direkt zum Tor
2. Komponente: Flucht vor dem Verteidiger

Skizze:

$s =$ Position des Stürmers
$v =$ Position des Verteidigers
$v_S =$ Geschwindigkeit des Stürmers
$V_T =$ Vektor zum Tor
$V_F =$ Fluchtvektor



Tor

$$Vs = \alpha V_T + \beta V_F$$

$\alpha = ?$
$\beta = ?$

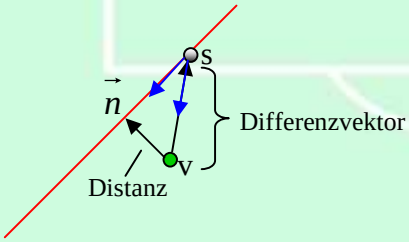
$$\left|\alpha V_T + \beta V_F\right| = vs$$

$$\alpha = \frac{|s-v|^2}{|s-v|+1}$$

$$\beta = 1 - \alpha$$

2) Strategie des Verteidigers:

$$\text{Distanz} = \left(\vec{s} - \vec{v}\right) * \vec{n}$$



Diese Skizze zeigt die 2 Anfangsstrategien des Verteidigers (1. Strategie= zum Stürmer, 2. Strategie = auf die Tangente des Stürmers zulaufen um den Torbereich abzusichern).

Entwicklung:
Von der Eins- gegen Eins-, bis zur Zwei- gegen Zwei -Situation

Die Vorlage die wir von Prof. Wolfgang Ring bekamen gab uns allen einen Anreiz weiter zu arbeiten. Nach dem wir in gewisse Lehrbücher reingeschnuppert hatten schrieben wir uns sofort Spielstrategien auf von denen wir glaubten sie verwirklichen zu können.

Taktiken:

2 Spieler (1 Stürmer, 1 Verteidiger):

Taktik des Stürmers:

1. nicht ins Out(x- Richtung, y-Richtung)
2. Richtung Torbereich (Radius 16m und 45°von den Torecken)
3. weg vom Verteidiger
4. „in die Mitte“

Taktik des Verteidigers:

1. zwischen Tor und Stürmer (5°)
2. zum Stürmer (Ball)

Abbruch wenn…

- ... wenn Abstand zwischen Stürmer und Verteidiger < 0,5m (Sieger: Verteidiger)
- … Stürmer schießen kann
  - ➢ Verteidiger im Weg (Sieger: Verteidiger)
    1. Notiz: Spiel geht weiter bis Schussfeld frei wird

- Verteidiger nicht im Weg (Sieger: Stürmer)

3 Spieler (1 Stürmer, 2 Verteidiger):

Taktik des Stürmers:
1. nicht ins Out
2. Richtung Torbereich (Radius 16m und 45°von den Torecken)
3. weg von den Verteidigern
4. „in die Mitte“

Taktik des 1. Verteidigers:
Dem Stürmer näherer Verteidiger (Abstandsmessung):
1. zwischen Tor und Stürmer (5°)
2. zum Stürmer (Ball)

Taktik des 2. Verteidigers:
Dem Stürmer weiter entfernt (Abstandsmessung):
1. zwischen Tor und Stürmer (direkte Torlinie)
2. weg vom 1. Verteidiger (gering gewichtet)
3. Winkel zum 1. Verteidiger möglichst groß
4. zum Stürmer (Ball) wird wichtiger wenn:
    i. 1. Verteidiger sehr nahe beim Stürmer
    ii. Der Stürmer dem Torbereich sehr nahe kommt

4 Spieler (2 Stürmer, 2 Verteidiger + Ball):

<u>Taktik des 1. Stürmers (= Stürmer der im Ballbesitz ist):</u>

1. nicht ins Out(x-Richtung, y-Richtung)
2. Richtung Torbereich (Radius 16m und 45°von den Torecken)
3. weg von den Verteidigern
4. „in die Mitte“ (Bezug auf Winkel)
5. Passt ab, wenn…
    i. der 2. Stürmer näher am Tor steht
    ii. Verteidiger ist nicht im Bereich der Passlinie(+/- 0.5 m)
    iii. 2. Stürmer steht im Besserem Winkel(<Mittelpunkt des Spielfeldes/Tor/Stürmer)
    iv. Ein Verteidiger zu nahe kommt
6. passt nicht ab, wenn kein Verteidiger das Schussfeld zum 2. Stürmer blockiert

<u>Taktik des 2. Stürmers:</u>

1. nicht ins Out
2. Kein Verteidiger im Bereich der Passlinie(+/- 0.5m)
3. Richtung Torbereich(Radius 16 m und 45°von den Torecken

4. weg von den Verteidigern
5. Weg vom 1. Stürmer
6. „in die Mitte“(Bezug auf Winkel)

Anfangs entwickelten wir eine Verteidigungsstrategie die uns beinahe perfekt vorkam, doch als wir diese dann umsetzen wollten sahen wir, dass dies nicht der Fall war. Dank Prof. Wolfgang Ring konnten wir eine Strategie entwickeln die mit unseren eigentlichen Vorstellung einer Verteidigung entsprechen.

Bei unserer ursprünglichen Vorlage erreichte der Verteidiger den Stürmer und der Stürmer lief vom Verteidiger weg. Doch anstatt dem Stürmer auf die Pelle zu rücken, lief der Verteidiger diesem nur hinterher. Deshalb mussten wir die bereits gezeigte Strategie völlig neu entwickeln. Nach mehreren Versuchen und langem Überlegen konnten wir schließlich eine „intelligentere“ Spielstrategie des Verteidigers entwickeln:

1) Der Verteidiger betrachtet nun das Gebiet des Stürmers:

Antizipieren der Situation des Stürmers im nächsten Schritt:

$\sim P_s^{n+1}$ = die wahrscheinliche Position des Stürmers im nächsten Schritt

P_S^{n} = Die Position des Stürmers

Δt = Schrittgröße

V_s^{n} = Geschwindigkeit des Stürmers

$$\sim P_s^{n+1} = P_s^{n} + \Delta t * V_s^{n}$$

2) Der Verteidiger sucht jenen für ihn im nächsten Schritt erreichbaren Punkt, der ihm den größtmöglichen Gebietsgewinn verschafft

Nachdem wir den Verteidiger modelliert hatten, war unsere Eins- gegen Eins-Situation fertig und wir konnten zur Zwei- gegen Zwei-Situation weitergehen.
Für diesen neuen Spielzustand mussten wir noch zwei weitere Spieler anfertigen und einen Ball hinzufügen.

Dem zweiten Verteidiger teilten wir folgende Aufgaben zu: Einerseits musste er zum zweiten Stürmer laufen und andererseits sollte er dafür sorgen, dass dieser den Ball nicht annehmen kann. Um unsere Verteidigungsstrategie zu vollenden, mussten wir bei den beiden Verteidigern bestimmen, auf welchen die Funktion des ersten Verteidigers und auf welchen jene des zweiten zutraf. Wir einigten uns darauf, dass der Verteidiger, der sich näher am ballführenden Stürmer befindet, die Funktion des ersten Verteidigers übernimmt.

Nun hatten wir eine starke Verteidigung modelliert, doch unser zweiter Stürmer zeigte keine Spielbeteiligung, denn wir hatten ihm noch keine Funktion zugeteilt.
Die Bewegung des Stürmers orientierte sich nun primär daran, so nah wie möglich zum Torbereich zu gelangen um einen Treffer zu erzielen.
Um die Stürmer zum Zusammenspielen zu bringen, bauten wir eine Passfunktion ein.

Einen endgültigen Feinschliff brachte uns die realitätsnahe Gestaltung, welche das Ganze zu einer sehenswerten Simulation macht.

Abbruchbedingungen:

Verteidiger nimmt dem Stürmer den Ball ab:

Die Abbruchbedingungen sind so angelegt das wenn der Verteidiger in den Radius (r= 0.3m) des Stürmers gelangt, der Sieg mit Hilfe einer Wahrscheinlichkeitsberechnung gelöst wird. Je näher der Verteidiger dem Stürmer ist umso höher ist die Wahrscheinlichkeit, dass er gewinnt.

Stürmer kommt in die Schussposition:

Weiters wird die Spielsituation beendet wenn der Stürmer im Strafraum in einer günstigen Schussposition steht(d.h kein Verteidiger steht im Weg, günstiger Winkel, etc.)

Programmiertechnische Umsetzung:

MATLAB:

MATLAB (die Kurzform für MATrix LABoratory) ist ein Mathematikprogramm mit integrierter Programmiersprache. Es eignet sich für einfache Rechnungen genauso gut, wie zum Formulieren komplizierter Funktionen und für die Visualisierung mathematischer Simulationen. Rechnungen können schnell und einfach in ein Kommandofenster geschrieben werden.

Programmierung:

Beim Programmieren mit MATLAB unterscheidet man Funktionen (functions), die aufgrund von Eingabeparametern ein bestimmtes Ergebnis errechnen, und Programme, die aktiv verschiedene Aktionen ausführen. Funktionen können sowohl aus dem Kommandofenster, als auch von Programmen oder anderen Funktionen aufgerufen werden.
Die Programmiersprache ist an Basic angelehnt und leicht ohne Vorkenntnisse erlernbar. Einer ihrer großen Vorteile ist, dass den Variablen automatisch ein bestimmter Datentyp zugewiesen wird, und so falsche Typen als Fehlerquelle entfallen. Programmzeilen werden mit einem Semikolon abgeschlossen, es sei denn man möchte, dass das Ergebnis der Zeile im Kommandofenster ausgegeben wird. Dies kann sehr hilfreich sein, um Fehler zu finden (die in längeren Programmen unweigerlich auftreten).

Variable:

Radien:

Abfangradius (Schuss) der Verteidiger:
Innerhalb dieses Kreises könnte ein Verteidiger einen Schuss abfangen. Er dient dazu, das Schussfeld des Ball führenden Stürmers auf das Tor oder den anderen Stürmer einzuschränken.

Abfangradius (Überspielen) der Verteidiger:
Wenn der Ball führende Stürmer diesen Kreis betritt, kann ihm der Verteidiger mit einer gewissen Wahrscheinlichkeit den Ball abnehmen. Diese wächst mit der Nähe zum Verteidiger.

Radius der Stürmer:
Gleich groß wie der Abfangradius des Verteidigers für Schüsse. Innerhalb dieses Kreises kann ein Stürmer einen Pass entgegennehmen. Er dient dazu, dem Ball führenden Stürmer ein Schussfeld für den Pass zu geben.

Koordinaten:

Koordinaten der Spieler:

Koordinaten Stürmer 1
```
pos_s = pos_s + delta_t*v_st;
```

Koordinaten Stürmer 2
```
pos_s2=pos_s2+delta_t*v_st2;
```

Koordinaten Verteidiger 1
```
pos_v = pos_v + delta_t*v_ve;
```

Koordinaten Verteidiger 2
```
pos_v2 =pos_v2 + delta_t*v_ve2;
```

Koordinaten Ball

```
pos_ball = pos_s + delta_t*v_st;
```

Geschwindigkeiten:

Geschwindigkeit der Stürmer:
Läuft aufs Tor zu (ohne Rücksicht auf Verluste!)

Geschwindigkeit der Verteidiger

- Der Verteidiger, der dem Ball führenden Stürmer am nächsten ist, lauft auf diesen zu, heftet sich an seine Fersen und versucht, ihm den Ball abzunehmen.
- Der andere Verteidiger hält sich kurz vor dem anderen Stürmer, sodass er diesen noch abfangen kann, falls dieser einen Pass empfängt.

Grafische Umsetzung:

Spieler:

Stürmer: rote Kreuze

Verteidiger: schwarze Kreuze

- Zyanblauer Kreis: Gibt den Abfangradius zum Überspielen wieder. In älteren Programmversionen ist dieser Kreis rot.
- Blauer Kreis: Gibt den Abfangradius für Schüsse wieder. In der endgültigen Version fehlt dieser Kreis.

Die Ausgangsversion

Diese frühe Version des Hauptprogramms soll einen Einblick in die grundlegende Arbeitsweise der Simulation bieten.

```
% Fussballsimulation
% Petra Katolnig, Wolfgang Ring
%
% 19.11.2008
% 11.1.2009

% loeschen aller Variablen
clear

% festlegen des Zeitvektors in Sekunden
```

```
delta_t = 0.05;
Anzahl_schritte = 200;

% maximale Geschwindigkeit des Verteidigers in m/sec
vmax_v = 6;
% maximale Geschwindigkeit des Stürmers in [m/sec]
vmax_s = 7;

% Definition der Anfangspositionen
pos_v0 = [10;40];
pos_s0 = [20;50];

tor    = [0;0];

laenge_feld = 105;
breite_feld = 70;

breite_tor = 7.32;

figure(1)
clf

% zeichne das Tor
plot([-breite_tor/2,breite_tor/2],[0 0],'linewidth',2,'color','red')
axis([-breite_feld/2-1,breite_feld/2+1,-1,laenge_feld]);

hold on

pause

% Anfangsposition
pos_v = pos_v0;
pos_s = pos_s0;

v_st0  = vmax_s * (pos_s - tor)/norm(pos_s - tor);
v_ve0  = [0;0];

v_st   = v_st0;
v_ve   = v_ve0;



% Grafik
stuermer_plot = plot(0,0);
set(stuermer_plot,'erasemode','xor','marker','+','color','red');

verteidiger_plot = plot(0,0);
set(verteidiger_plot,'erasemode','xor','marker','+','color','black');



for k = 1:Anzahl_schritte;

    % Berechnung der Geschwindigkeit des Verteidigers
```

```
    v_ve = geschw_vert(pos_v,pos_s,v_st);

    % Berechnung der Geschwindigkeit des Stuermers
    v_st = geschw_st(pos_s,pos_v,v_ve);

    % update der Position des Stuermers
    pos_s = pos_s + delta_t*v_st;

    % update fuer pos_v mit Euler-Verfahren
    pos_v = pos_v + delta_t*v_ve;


    % plot der Postion des Stuermers
    set(stuermer_plot,'xdata',pos_s(1),'ydata',pos_s(2));


    % plot der Postition des Verteitigers:
    set(verteidiger_plot,'xdata',pos_v(1),'ydata',pos_v(2));

    drawnow
    pause(0.1)

end
```

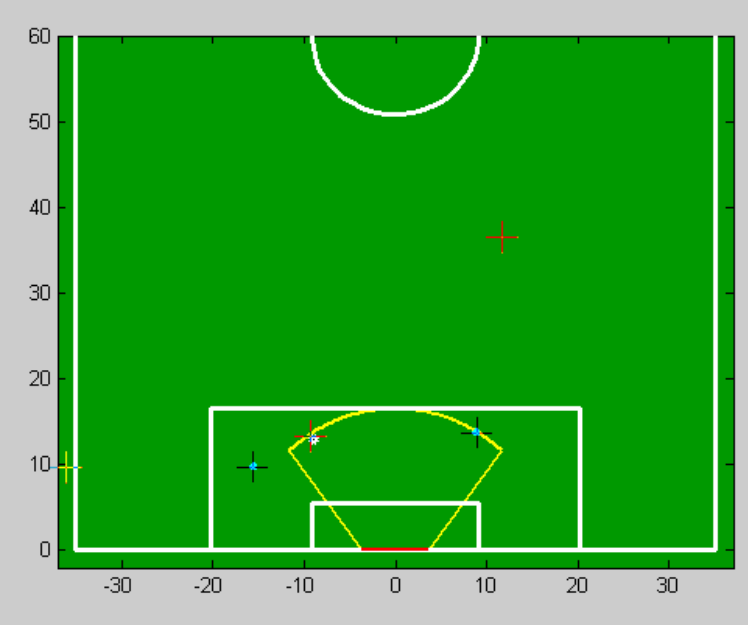
Spielfeld:

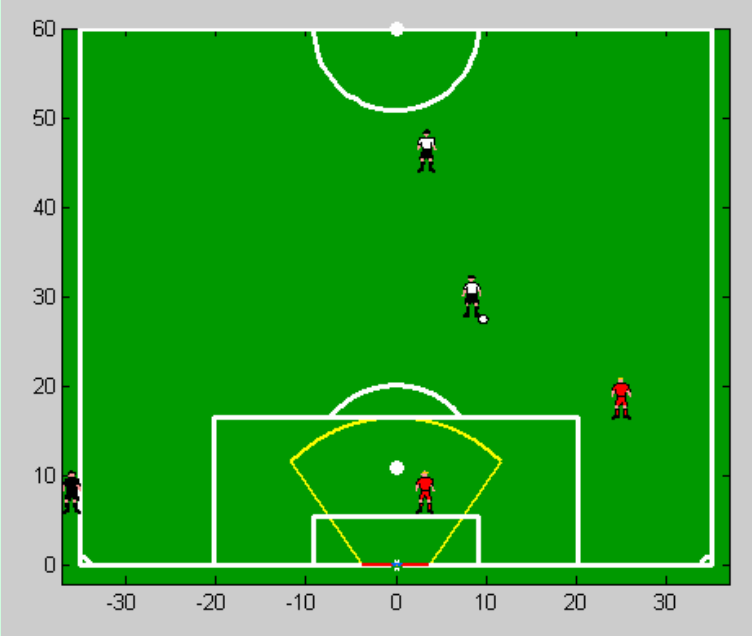
Um unser Fußballfeld realitätsnahe zu gestalten färbten wir es grün. Weiteres fügten wir weiße Linien, wie die Outlinie, den Strafraum, den 5-m Raum und den Mittelkreis hinzu.

Zusätzlich kennzeichneten wir den Raum, indem der Stürmer die Möglichkeit hat, zu schießen.

Simulationen:

Beispiele:





Zusammenfassung:

Fußball und Mathematik sind zwei Begriffe die selten zusammen verwendet werden. Üblicherweise denkt niemand an Mathematik wenn von Fußball und Spielstrategien die Rede ist. Doch in dieser Woche lernten wir, dass auch diese zwei Begriffe sehr wohl etwas miteinander zu tun haben.
Als wir unsere Problemstellung zum ersten Mal hörten, hatten wir noch keine konkrete Vorstellung, wie wir unsere Spielstrategien modellieren könnten. Deshalb bekamen wir eine kurze Einführung von Univ. Prof. Wolfgang Ring zu diesem Thema. In dieser Einleitung präsentierte er seine grundlegende Vorstellung einer einfachen Spielstrategie die wir weiter auszubauen und zu bearbeiten hatten.
Weiteres lasen wir Lehrbücher über Fußball, lernten mit MATLAB umzugehen, um unsere Spielstrategien modellieren zu können und hatten dabei mit der ein oder anderen Schwierigkeiten zu kämpfen, jedoch wurden diese alle zeitgerecht gelöst. Trotz der nicht allzu leichten Problemstellung hatten wir beim Modellieren der verschiedensten Spielsituationen eine große Freude.
Schließlich und endlich möchten wir uns bei unseren Betreuern, Prof. Wolfgang Ring und Petra Katolnig für ihre großartige Unterstützung bedanken ohne die wir wohl kaum im Stande gewesen wären diese Simulation zu modellieren. Die Spieler, welche von (Univ.) Prof. Wolfgang Ring umgesetzt wurden, und die Zeichnungen, welche von Petra Katolnig erstellt wurden, sind wahrlich Meisterwerke. Vielen Dank.

Forensische Wissenschaft

Simulation des Einsturzes eines WTC-Turms

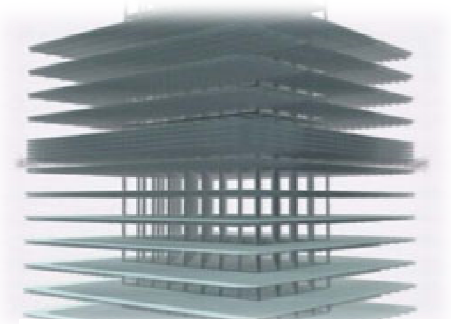
Projektleiter: Stephen Keeling

Elisabeth Gaar, Elisabeth Heschl, Eva Meisterhofer, Inge Siegl, Lisa-Maria Sommer, Jasmin Stoff

Inhaltsverzeichnis:

Problemstellung:

Die Anschläge des 11.9.2001 stellen bis heute die katastrophalsten Terroranschläge weltweit dar und lassen immer noch Zweifel offen.
An diesem Tag wurden 4 Flugzeuge der amerikanischen Airforce entführt, die vier Ziele ansteuerten: Die Twintowers, das Pentagon und das Weiße Haus, das zum Glück nicht erreicht wurde.
Um 8.46 traf das erste Flugzeug den Nordturm, um 9.03 wurde der Südturm getroffen und um 9.43 stürzte das 3.Flugzeug in das Pentagon. 62 Minuten nach der Südturmattacke (10.05) und 102 Minuten nach der Nordturmattacke (10.28) stürzten diese in die Tiefe. Das World Trade Center 7 brach um 17.20 in sich zusammen.
Nach der offiziellen Erklärung der amerikanischen Regierung ist das Feuer, das durch die Kerosinexplosion ausgelöst wurde, verantwortlich für den Einsturz der Twintowers. Die amerikanische Regierung zeigt den Hergang des Zusammenbruchs im Pancake-Modell, bei dem angenommen wird, dass sich die Eisenträger von ihren Verankerungen lösen, ein Stockwerk auf das nächste prallte und somit alle in die Tiefe gerissen werden.



Pancake-Modell

Wie es zu dem Einsturz von World Trade Center 7 kam, das nicht von einem Flugzeug getroffen wurde und was genau bei dem Anschlag auf das Pentagon passierte, wurde in der offiziellen Erklärung sehr vernachlässigt und scheinbar auch von den Medien ferngehalten.
Wegen diesen Lücken und weil es keinen vergleichbaren Einsturz von Gebäuden gibt, der von Feuer verursacht wurde, wird die offizielle Erklärung von vielen Seiten angezweifelt.
Unsere Aufgabe war es, den Einsturz eines WTC-Turmes möglichst einfach zu simulieren und zu entscheiden, ob die offizielle Erklärung plausibel ist, und wenn nicht, sollte eine Hypothese aufgestellt werden die mit den Daten besser übereinstimmt.

Formelübersicht für das Pancake-Modell:

g	Gravitationsfeldstärke	9,81m/s²
ST	Gesamtanzahl der Stockwerke	110
h	Höhe eines Stockwerks	4m
n	Anfangsanzahl der fallenden Stockwerke	16
t	Aufpralldauer	0,001s
t_0	Anfangszeit zu Beginn eines freien Falls	0
t_1	Zeit kurz vor Aufprall	
t_f	gesamte Falldauer für einen freien Fall	
t	Aufpralldauer	0,001s
x(t)	Höhe des Gebäudes am Zeitpunkt t	
m_0	Masse eines Stockwerks	2618000kg
m_1	nicht verstaubte Masse der bereits fallenden Stockwerke	
m_2	nicht verstaubte Masse des getroffenen Stockwerks	$m_0 * \mu$
µ	Staubquotient	0,25
v_0	Anfangsgeschwindigkeit zu Beginn eines freien Falls	0
v_1	Geschwindigkeit kurz vor Aufprall	
v_2	Geschwindigkeit kurz nach Aufprall	
P_0	Befestigungswiderstand	

Unser erstes Ziel war es, eine Formel für die Geschwindigkeit aufzustellen, die die herabfallenden Stockwerke erreichen. Wenn man eine Funktion vom Weg in Abhängigkeit von der Zeit betrachtet, gibt die erste Ableitung die Geschwindigkeit und die 2. Ableitung die Beschleunigung an. In unserem Modell ist jede Geschwindigkeit und damit jede Beschleunigung negativ, da wir uns von der Höhe des Gebäudes hinab bewegen. Da die fallenden Stockwerke mit einer Beschleunigung von g fallen, gilt

$$x(t)'' = -g$$

und durch integrieren nach t an den Grenzen t_1 und t_2 erhält man

$$x'(t_1) - x'(t_0) = -g(t_1 - t_0)$$

dabei lässt sich $x'(t_0) = v_0$ ersetzen, da die erste Ableitung der Geschwindigkeit entspricht.

$$x'(t_1) - v_0 = -g(t_1 - t_0)$$

Durch erneutes integrieren nach t an den Grenzen t_1 und t_0 erhält man

$$x(t_1) - x(t_0) - v_0 * (t_1 - t_0) = \frac{-g(t_1 - t_0)^2}{2}$$

und kann nun $x(t_0)$ mit h ersetzen und erhält die quadratische Gleichung in s

$$x(t_1) = \frac{g(t_1 - t_0)^2}{2} - v_0 * (t_1 - t_0) - h$$

die durch einsetzten in die große Auflösungsformel ergibt

$$t_1 - t_0 = \frac{v_0 \pm \sqrt{{v_0}^2 + 2 * g * h}}{g}$$

und schließlich ergibt sich die Formel für die Zeit kurz vor dem Aufprall

$$t_1 = t_0 + \frac{v_0 + \sqrt{{v_0}^2 + 2 * g * h}}{g}$$

Diese Formel lässt sich adaptieren für den freien Fall. Dabei geht man davon aus, dass das gesamte oberste Stockwerk von einer Höhe (h * ST) fällt, (also Höhe eines Stockwerks multipliziert mit der Anzahl der Stockwerke) also wird h damit ersetzt. Zusätzlich prallt kein Stockwerk auf ein anderes, und die gesamte Zeit lässt sich auf einmal berechnen. Wir erhalten unsere Formel für die Zeit eines freien Falls

$$t_f = \frac{v_0 + \sqrt{v_0^2 + 2 * g * h * ST}}{g}$$

Die Geschwindigkeit am Zeitpunkt t_1 ist v_1 und um v_1 zu berechnen gehen wir ein paar Rechenschritte zurück zu

$$x'(t_1) - v_0 = -g(t_1 - t_0)$$

und nun ersetzen wir das ($t_1 - t_0$) durch das Ergebnis unserer quadratischen Lösungsformel und verwenden, dass $x'(t_1) = v_1$. Damit erhalten wir

$$v_1 = -g\left(\frac{v_0 + \sqrt{{v_0}^2 + 2 * g * h}}{g}\right) + v_0$$

und können den Ausdruck vereinfachen, und erhalten somit die Formel für die Geschwindigkeit kurz vor dem Aufprall

$$v_{1=-}\sqrt{v_0^2 + 2 * h * g}$$

Mit der Formel für v_1 kann man zwar ohne Probleme die Zeit vom Beginn des Falls bis zum ersten Aufprall berechnen, doch die Geschwindigkeit wird abgebremst, sobald die fallende Masse auf ein befestigtes Stockwerk trifft. Dabei haben wir einerseits die Befestigung P_0 aber auch den Impulserhaltungssatz berücksichtigt. Laut dem Newtonschen Gesetz gilt

$$F = m * a$$

Also Kraft ist gleich Masse mal Beschleunigung. Angewandt auf unser Beispiel ergibt sich

$$x''(t) * m = -g * m$$

Wenn man diese Formel für die beiden Massen m_1 und m_2 adaptiert und zusätzlich die Kräfte F_1 (= Impulserhaltung von oben), F_2 (= Impulserhaltung von unten) und

F_0 (=Kraft die die Befestigung ausübt, allerdings NUR im Moment des Aufpralls), erhält man

$$x_1''(t_1) * m_1 = -g * m_1 + F_0 + F_1$$

$$x_2''(t_2) * m_2 = -g * m_2 * H + F_2$$

Weiters definiert man die Hilfsfunktionen

$$\boldsymbol{H(t) = \int_{-\infty}^{t} \delta(t)\, dt}$$

$$\boldsymbol{\int_{t-\varepsilon}^{t+\varepsilon} \delta\,(t)\, dt = 1}$$

um besser mit dem unendlich kleinen Moment (t steht für den Moment des Aufpralls, $\boldsymbol{\varepsilon}$ ist die unendlich kleine Zeit vor bzw. nach dem Aufprall) umgehen zu können. Man definiert die Kräfte als

$$F_1 = P_1 * \boldsymbol{\delta}$$

$$F_2 = P_2 * \boldsymbol{\delta}$$

$$\mathrm{F_0} = \mathrm{P_0} * \boldsymbol{\delta}$$

Da P_1 und P_2 Kräfte aus den entgegengesetzten Richtungen sind, ergibt sich

$$P_1 + P_2 = 0$$

Im nächsten Schritt werden die beiden Funktionen integriert nach t an den Grenzen $t_2 - \boldsymbol{\varepsilon}$ und $t_2 + \boldsymbol{\varepsilon}$ erhält man

$$(x_1'(t_2 + \boldsymbol{\varepsilon}) - x_1'(t_2 - \boldsymbol{\varepsilon})) * m_1 = P_1 + P_0$$

$$(x_2'(t_2 + \boldsymbol{\varepsilon}) - x_2'(t_2 - \boldsymbol{\varepsilon})) * m_2 = P_2$$

Man geht davon aus, dass sich die beiden Massen m_1 und m_2 nach dem Aufprall zusammen bewegen, und somit mit der gleichen Geschwindigkeit. Es gilt $x_1'(t_2 + \boldsymbol{\varepsilon}) = x_2'(t_2 + \boldsymbol{\varepsilon}) = v_2$ und des weiteren $x_2'(t_2 - \boldsymbol{\varepsilon}) = 0$, da sich die Masse des Stockwerks, auf das die oberen Stockwerke fallen, vor dem Aufprall noch nicht bewegt. Es ergibt sich logischerweise

$$m_1 * v_2 - m_1 * v_1 = P_1 + P_0$$

$$m_2 * v_2 - 0 = P_2$$

Durch das Addieren der beiden Gleichungen und Verwendung der Nebenbedingung $P_1 + P_2 = 0$ folgt des weiteren

$$(m_1 + m_2) * v_2 - m_1 * v_1 = P_0$$

In Folge dessen erhalten wir schlussendlich die Formel für die Geschwindigkeit nach dem Aufprall

$$v_2 = \frac{P_0 + m_1 * v_1}{m_1 + m_2}$$

Wobei wir P_0 als Befestigungswiderstand definierten, der sich aus dem Druck p, den ein Stockwerk pro Quadratmeter aushält, der Fläche A eines Stockwerks des WTC, der Erdbeschleunigung und der Aufpralldauer ergibt. Wir erhalten die Formel

$$P_0 = p * A * g * t$$

Und in dem wir die uns bekannten Werte einsetzen, bekommen wir einen ersten Wert für P_0

$$P_0 = 300 * 63^2 * g * t$$

$$P_0 = 11680{,}767$$

Modelle in C#

Ursprünglich wollten wir unser Pancakemodell in MatLab programmieren, allerdings hatten die meisten Teilnehmer kaum Erfahrung mit diesem Programm und auf den Rat von unserem Projektleiter, Stephen Keeling, konnten wir leider nicht zurückgreifen, da dieser an dem Tag gerade eine Vorlesung in Graz hielt.
Da ein Gruppenmitglied schon seit Jahren mit C# arbeitet, eine Programmiersprache von Windows, entschlossen wir uns, unser Modell vorerst in diesem Programm darzustellen. So hatten wir auch wirklich sehr schnell unsere ersten Ergebnisse, die noch nicht wirklich realitätsgetreu waren, da laut ihnen der Nordturm erst in 30 Jahren zum Stillstand gekommen wäre.
Nach einigen Überarbeitungen des Codes näherten wir uns immer mehr unserem eigentlichen Ziel, reale Zeiten durch unsere Berechnungen zu erlangen.
Zu allererst errechneten wir den Fall eines Gebäudes mit den richtigen Parameter des WTC (Anzahl, Masse und Höhe der Stockwerke) in Abhängigkeit des Impulserhaltungssatzes. Hierbei vernachlässigten wir den Flugzeugeinsturz und gingen davon aus, dass das höchste Stockwerk aus irgendeinem Grund als erstes in die Tiefe ging.
Daraufhin erweiterten wir unseren Code, indem wir den Flugzeugeinsturz miteinbezogen, indem wir annahmen, dass der Einsturz des Gebäudes von dem stark beschädigten Stockwerk ausging (94.Stockwerk) und die 16 Stockwerke darüber auf einmal in die Tiefe stürzten.
Als nächste Komponente nahmen wir den Staub hinzu, der während des Zusammenbruches freigesetzt wird, denn durch ihn verliert das Gebäude während des Falles immer mehr an Masse. Zunächst ließen wir bei diesem Modell den Flugzeugeinsturz wieder außer Acht, aber bei unserer nächsten und letzten Erweiterung brachten wir es wieder ein und kamen schlussendlich auf einen sehr realistischen Wert unseres Pancakemodells, und zwar 11,56 Sekunden.
Zum Vergleich codierten wir auch den freien Fall, der einer kontrollierten Sprengung entspricht, und kamen hier auf 8,9 Sekunden.

Um besser zu veranschaulichen wie unser Modell funktioniert, folgen der Quelltext mit Kommentaren und die graphische Darstellung von C#.

Pancakesimulation

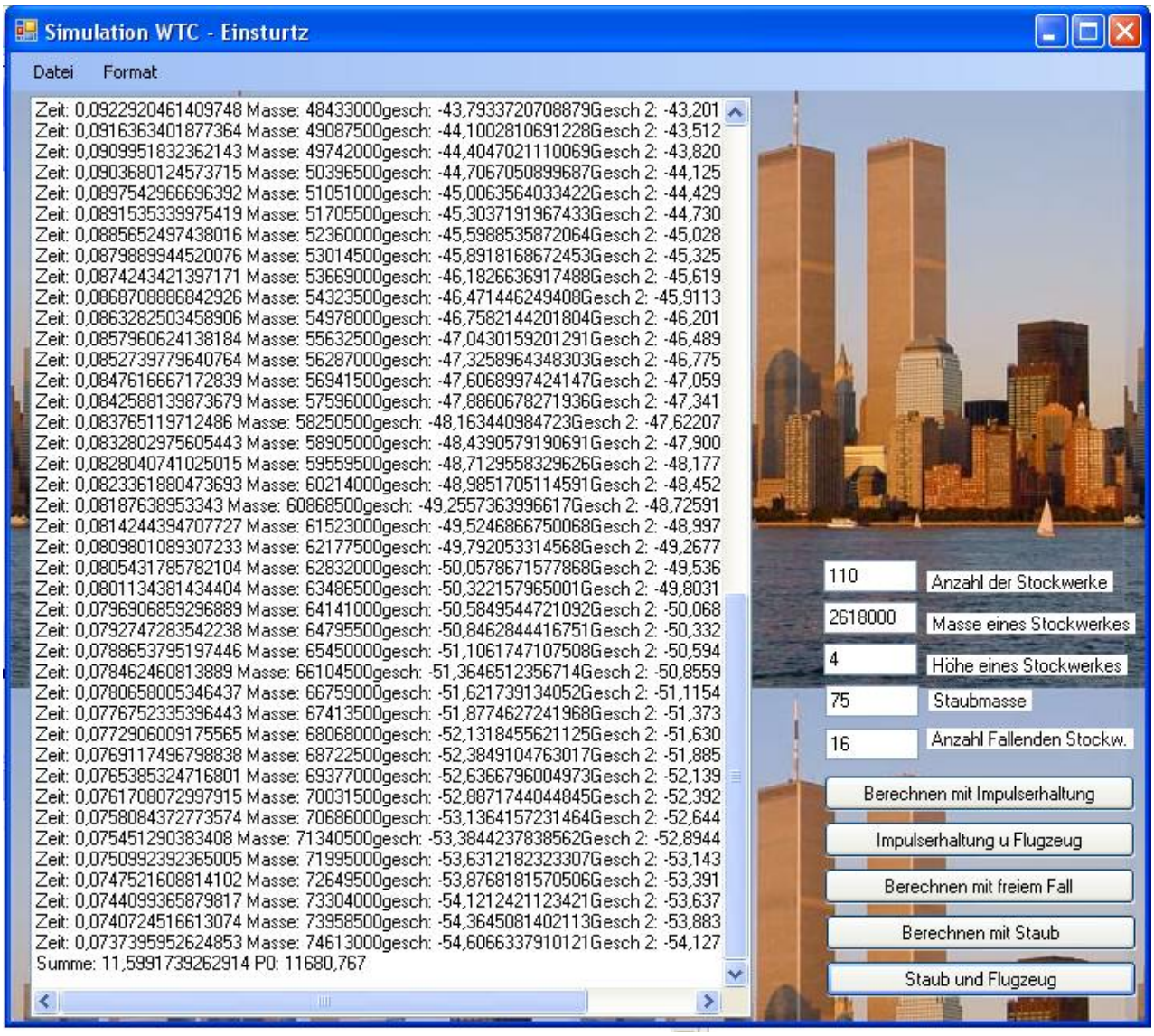
```
double h = int.Parse(m_tbHoehe.Text); //höhe eines Stockwerkes
            double g = (double)981/(double)100;
            double quot = double.Parse(m_tbQuotient.Text) / 100;
                                        //Prozentsatz des Zerfalles
            int n = int.Parse(m_tbAnzahl.Text); //Anzahl der
                                                          Stockwerke
             double M = double.Parse(m_tbMasse.Text); //Masse eines Stockwerkes
            double v = 0; //anfangsgesch.
            double v1 = 0; //gesch. kurz vor dem Aufprall
            double v2 = 0; //gesch. kurz nach dem Aufprall
            double t_1 = 0;
            double x = h * n; //gesamthöhe
            double summe = 0;
            double stockwerke = double.Parse(m_tbFallende.Text);
                          //wie viele Stockwerke auf einmal fallen
                double o=63*63;
                double y=(double)1/(double)1000;
                double P_0 = 300 * o * y * g;
            double m1 = stockwerke*M; //Masse der Fallenden
                                              Stockwerke
            int stockwerke_for = int.Parse(m_tbFallende.Text); //wie
                                        viele Stockwerke auf einmal fallen
            double[] zeit = new double[n];
            double[] Masse = new double[n];
            double[] gesch2 = new double[n];




            // berechnung der Geschwindigkeiten und der Zeit der auf
             einmal fallenden Stockwerke
            v1 = -Math.Sqrt(v * v + 2* h * g);
            t_1 = (v + Math.Sqrt(v * v + 2 * h * g)) / g;
            v2 = (P_0 + m1 * v1) / (m1 + M - quot * M);
```

```
            v = v2;


            //Erneuerung der Masse
            m1 += M - stockwerke*quot * M;
            summe += t_1;
            double[] gesch = new double[n];
            for (int i = stockwerke_for; i < n; i++)
            {
                v1 = -Math.Sqrt(v * v + 2 * h * g); //Gesschw. kurz
                                                                vor Aufprall
                t_1 = (v + Math.Sqrt(v * v + 2 * h * g)) / g; //Zeit
                                                            kurz vor Aufprall
                v2 = (P_0 + m1 * v1) / (m1 + M - quot * M);
                //v2 = (i + 1) * v1 / (i + 2); //Impulserhaltung
                v = v2;
                gesch2[i] = v2;
                m1 += M - quot * M;
                zeit[i] = t_1;
                summe += zeit[i];
```

Freierfall-Simulation:

```
class Freierfall
    {
        public double summeberechnen(double h, double M, int n, double g)
        {
            double v = 0;
            double v1 = 0;
            double v2 = 0;
            double t_1 = 0;
            double x = h * (n - 16);
            double summe = 0;
            double[] zeit = new double[n];
            for (int i = 0; i < n; i++)
            {
                v1 = -Math.Sqrt(v * v + 2 * h * g); //Gesschw. kurz
                                                                vor Aufprall
                t_1 = (v1 + Math.Sqrt(v1 * v1 + 2 * h * g)) / g; //Zeit kurz vor
Aufprall
                v2 = v1;
                v = v2;
                x = x - h; //Höhe
                zeit[i] = t_1;
                summe += zeit[i];
            }
            return summe;
        }
    }
```

Graphische Darstellung von C#



MATLAB-Modell

Ziel unseres MATLAB-Modells war es, die für einen Pfannkuchenkollaps benötigte Zeit der für einen freien Fall des obersten Stockwerkes benötigten Zeit gegenüberzustellen.

Die Zeit (t_f) für den freien Fall berechnen wir mit der Formel

$$t_f = \frac{\sqrt{2 * h * ST}}{g},$$

h = Höhe eines Stockwerks

ST = Gesamtanzahl der Stockwerke

Die Zeit für den Pfannkuchenkollaps des untersten der anfangs fallenden Stockwerke wird genau wie beim C#-Modell aus den Geschwindigkeiten vor und nach dem Aufprall unter Berücksichtigung des Impulserhaltungssatzes, der Befestigung der Stockwerke und der Verstaubung berechnet.

Im C#-Modell sind der Impuls P_0, den ein Stockwerk aushält, und der Verstaubungsgrad μ, der angibt, wie viel Prozent der Masse nicht verstaubt werden, konstant.

Im MATLAB-Modell aber verhalten sich diese beiden Variablen indirekt proportional zum Gewicht der bereits fallenden Masse: Wir nehmen an, dass die Befestigungen der unteren Stockwerke durch Druckwellen zerstört werden und P_0 somit immer kleiner wird. Durch die mit der Zeit größer werdende fallende Masse wird außerdem ein immer größerer Prozentteil der Stockwerke verstaubt – somit wird auch μ immer kleiner.

P_0 und μ werden nach jedem Aufprall neu berechnet:

$$pr = \frac{m_2}{m_1}$$

ist das Verhältnis der neu dazukommenden Masse m_2 zur alten Masse m_1. Die neuen Werte für μ und P_0 erhält man durch folgende Formeln:

$$\mu = \mu_0 * pr^k$$

μ_0 = der Wert von μ beim ersten Aufprall

k = Konstante, hier 0,25

und

$t = t - (t * pr)$

$P_0 = 300 * 63^2 * g * t$

t = Dauer des Aufpralls.

Zum besseren Vergleich des Pfannkuchenmodells mit dem freien Fall, der beim obersten Stockwerk beginnt, brauchen wir zusätzlich zur Falldauer des untersten Stockwerks von N (N = die anfangs fallenden Stockwerke) auch die des obersten Stockwerks. Ist das unterste Stockwerk von N am Boden angekommen, berechnen wir die Zeit für den Zusammenbruch der restlichen Stockwerke im freien Fall:

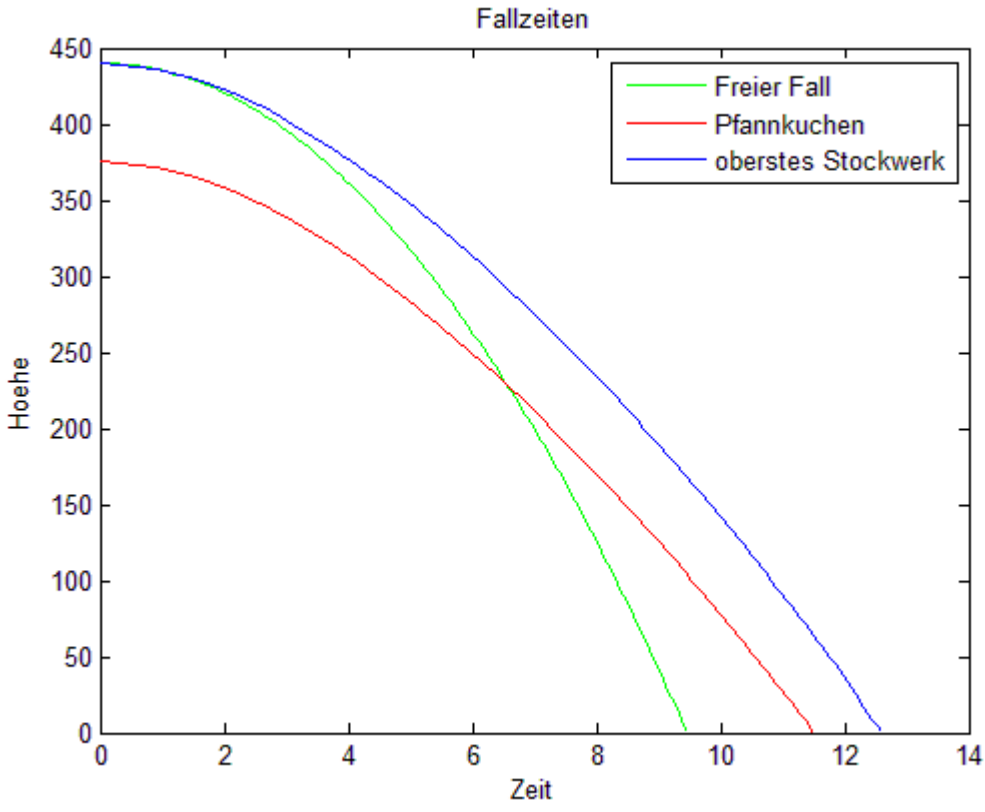
$$t = Z(end) + \frac{v_{o+\sqrt{v_0{}^2 + 2*g*h*N}}}{g}$$

Z(end) = Zeit, zu der das unterste Stockwerk von N am Boden angekommen ist.

v0 = Fallgeschwindigkeit zum Zeitpunkt Z(end)

h * N = Gesamthöhe der restlichen Stockwerke

Berechnen wir die Fallzeiten mit den Werten des World Trade Centers (Gesamtzahl der Stockwerke ST = 110, Anzahl der anfangs fallenden Stockwerke N = 16, Stockwerkhöhe h = 4, Masse m0 eines Stockwerks = 2618000, Aufpralldauer t am Anfang = 2, Teil der anfangs nicht verstaubten Masse mu = 0,75), bekommen wir folgende Grafik:



For n 27

Die Ergebnisse dieser Berechnung sind:

Dauer eines freien Falls = 9.4712 Sekunden

Dauer eines Pfannkuchenfalls = 11.5038 Sekunden

Dauer eines Pfannkuchenfalls für das oberste Stockwerk = 12.5846 Sekunden

Zur besseren Veranschaulichung noch der Quelltext mit Kommentaren:

wtc.m

```
ST = 110;   %Gesamtzahl der Stockwerke
N = 16;     %Anzahl der anfangs fallenden Stockwerke
S = ST-N;   %Anzahl der noch nicht betroffenen Stockwerke

h = 4;      %Raumhöhe (m)
g = 9.81;   %Erdbeschleunigung

%Variablen fuer das Pancake-Modell
v0 = 0;     %Anfangsgeschwindigkeit (m/s)
v1 = 0;
v2 = 0;
t0 = 0;     %Anfangszeit (s)
t1 = 0;

t = 2;                          %Aufpralldauer am Anfang (s)
ts = zeros(1,(S+1));            %alle t
mts = 0;                        %Mittel aller t
ts(1) = t;
P = (300 * (63^2) * g * t);     %Impuls, den m2 aushält

m0 = 2618000;   %Masse eines Stockwerks
m1 = m0*N;      %Masse der anfangs fallenden Stockwerke
m2 = 0;
m3 = m1;        %Anfangsmasse zur Verstaubungsberechnung
pr = 0;
k = 0.25;       %Exponent von pr

mu0 = 0.75;             %Teil der anfangs nicht verstaubten Masse
mu = mu0;
mus = zeros(1,(S+1));   %alle mu
mmus = 0;               %Mittel aller mu
mus(1) = mu;

%Variablen fuer den freien Fall
tf = 0;
v1f = 0;
t1f = zeros(1,ST);
Xf = zeros(1,ST);
tof = 0;
t1of = zeros(1,N);
Xof = zeros(1,N);

%fuer die Grafik
Z = t0;
X = S*h;

for H = 1:S

    %Geschwindigkeit
    v1 = - sqrt(v0^2 + 2 * g * h);  %Geschwindigkeit kurz vor dem Aufprall

    m2 = m0*mu;
    v2 = (P + m1 * v1)/(m1 + m2);   %Geschwindigkeit nach dem Aufprall
```

```
    %Verstaubung der Anfangsmasse
    m1 = m1 - (m3 * (1-mu)) / S;
    m3 = m3 - (m3 * (1-mu)) / S;

    if (v2 > 0)
        disp(strcat('Das Gebäude stürzt nicht weiter ein.'));
        break;
    end

    %Zeit
    t1 = t0 + ((v0 + sqrt(v0^2 + 2 * h * g))/g);    %Zeit kurz vor dem
                                                    Aufprall
    %Neue Anfangswerte
    v0 = v2;
    t0 = t1;
    pr = m2 / m1;   %Massenaenderung
    m1 = (m1 + m2);
    mu = mu0 * pr^k;
    t = t - (t * pr);
    P = (300 * (63^2) * g * t);

   %fuer die Grafik
    Z = [Z,t0];
    X = [X,X(end)-h];

    mus(H+1) = mu;
    ts(H+1) = t;

end
%Mittelwert aller mu und t
for q = 1:S
    mmus = mmus + mus(1,q);
    mts = mts + ts(1,q);
end
mmus = mmus / S;
mts = mts / S;
disp(strcat('Mittel von mu=',num2str(mmus),', Mittel von t=',num2str(mts)));

%freier Fall
tf = sqrt((2 * h * ST) / g);    %Zeit
v1f = -g * tf;                  %Geschwindigkeit
t1f = (0:ST) * tf / ST;         %Zeit
Xf = h * ST - g * t1f.^2 / 2;   %Hoehe

if v2 < 0
    %freier Fall der obersten Stockwerke am Ende
    tof = Z(end) + (v0 + sqrt(v0^2 + 2 * g * h * N)) / g;           %Zeit
    t1of = Z(end) + (0:N) * (tof - Z(end)) / N;                     %Zeit
    Xof = h * N + v0 * (t1of - Z(end)) - g / 2 * (t1of - Z(end)).^2; %Hoehe
    Xof(end) = abs(Xof(end));
end

disp(strcat('Dauer eines freien Falls=',num2str(t1f(end)),' Sekunden'));

%Pfannkuchenfall
disp(strcat('Dauer eines Pfannkuchenfalls=',num2str(Z(end)),' Sekunden'));
disp(strcat('Dauer eines Pfannkuchenfalls für das oberste
Stockwerk=',num2str(t1of(end)),' Sekunden'));

%Grafik: Zeit-Hoehe
plot(t1f,Xf,'g',Z,X,'r',Z,X+h*N,'b',t1of,Xof,'b');
title('Fallzeiten');
xlabel('Zeit');
ylabel('Hoehe');
legend('Freier Fall','Pfannkuchen','oberstes Stockwerk');
```

Auswirkung von Verstaubungsgrad und Befestigung

Logischerweise ist eine Abhängigkeit der Fallzeit des World Trade Centers von der Stärke der Befestigung der Stockwerke und dem Grad der Verstaubung der fallenden Masse gegeben. Deshalb interessierte uns dieser Zusammenhang sehr.

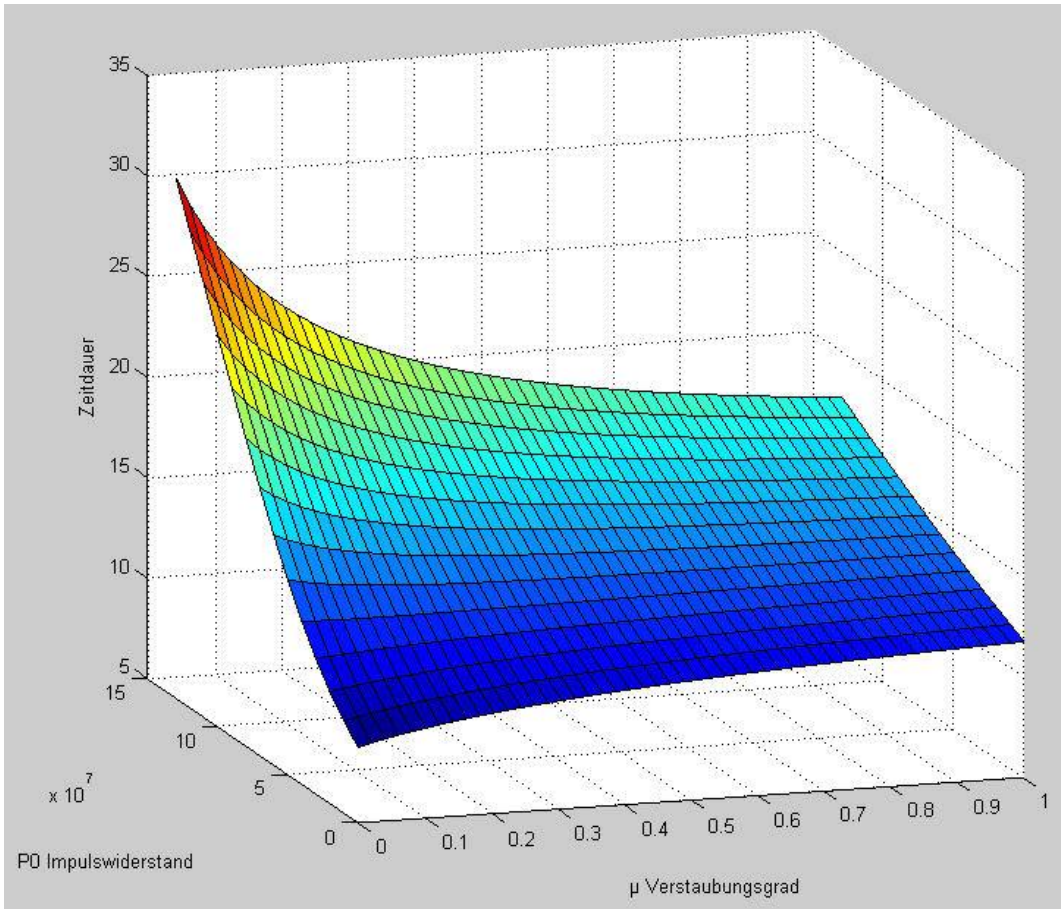
Zur Veranschaulichung wollten wir eine Grafik erstellen in der die drei Werte dargestellt werden. So sollte einfach zu erkennen sein, bei welcher Verstaubung und Befestigungsstärke die kürzeste, beziehungsweise längste Fallzeit zu erwarten ist.

Dazu wurde ein Code programmiert, der für jede beliebige μ-P Kombination die Fallzeit berechnet.

Fallgrafik.m

```
n = 50;    % Anzahl der verschiedenen mu
m = 14;    % Anzahl der verschiedenen P
% Anlegen der Variablen
x = zeros(n,m);
y = zeros(n,m);
T = zeros(n,m);

% Die for-Schleife in der jede mu-P Kombination durchlaufen wird
   for i = 1:n
      mu = (i-1)/(n-1);   % Bildet n Werte für mu zwischen 0 und 1
      for j = 1:m
         P = (j-1)* 1.0e7;    % Bildet m Befestigungswiderstände
         T(i,j)= gesch_dauer(mu,P);   % Berechnet für jede mu-P Kombination
               die Fallzeit (indem die Funktion gesch_dauer ausgeführt wird)
         x(i,j)= mu; % Weist der momentanen x-Position den Wert von mu zu
         y(i,j)= P;  % Weist der momentanen y-Position den Wert von P zu
```

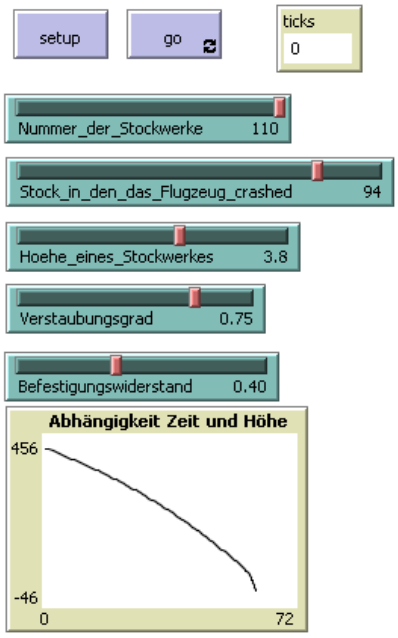
```
    end     % for j
end     % for i

figure(1)
surf (x,y,T); %  weist den Werten die Achsen in der Grafik zu

% Beschriftung
    xlabel('µ Verstaubungsgrad')
    ylabel('P0 Impulswiderstand')
    zlabel('Zeitdauer')
```



Das Ergebnis war eine Grafik in der leicht ersichtlich ist, dass die längste Falldauer [rot] bei der stärksten Befestigung und der Verstaubung 0 (es bleibt also keine Masse mehr übrig) besteht. Dies lässt sich damit erklären, dass Masse notwendig ist um die Befestigung zu durchbrechen. Wenn aber die vollständige Masse der Stockwerke, auf die das Oberste fällt, verstaubt wird, vergrößert sich die fallende Masse nicht, sondern bleibt immer nur bei der Masse des ersten Stockwerkes. So wird es schwerer, die einzelnen Befestigungen zu durchbrechen und der Fall verlangsamt sich. Der Tiefpunkt [blau] ist wiederrum bei dem kleinsten Widerstand erreicht wobei aber auch hier die Verstaubung 0 ist.

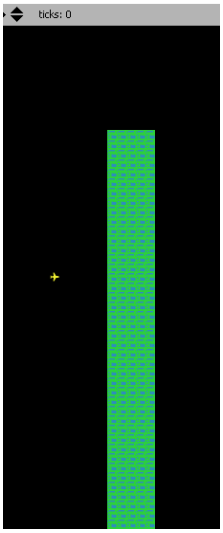
Nebenbei ist zu erkennen, dass μ einen wesentlich kleineren Einfluss als P auf die Fallzeit hat.

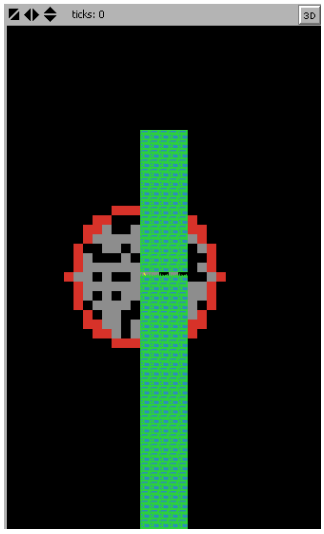
Modell mit NetLogo



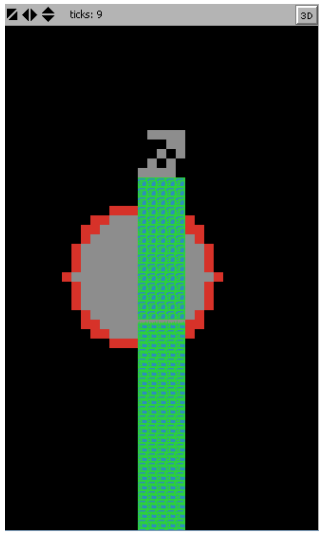
Um unsere Annahmen und Berechnungen anschaulicher zu machen haben wir uns den Computer zu nutze gemacht. In C# programmierten wir ein Programm in dem ein User die Daten nach Belieben ändern kann und in Matlab stellte wir unsere eigene Hypothese in den Mittelpunkt, veranschaulicht in einem Diagramm. Doch wollten wir auch den Einsturz des Turmes möglichst realitätsgetreu darstellen.

Zu diesem Zweck machten wir uns mit dem Programm NetLogo vertraut, was sich als sehr schwierig und anstrengend herausstellte. Alleine 1 ½ Stunden verbrachten wir nur mit dem Tutorial, bei dem uns zum Glück Steve zur Seite gestanden hat.





Für die grafische Simulation haben wir ein Flugzeug und einen Turm erstellt. Unser Turm setzt sich aus 5 einzelnen Spalten zusammen um die Größenverhältnisse genauer zu machen. Vor der Simulation kann man sich aussuchen, wie viele Stockwerke das Gebäude hat, wie hoch ein Stockwerk sein soll und in welches Stockwerk das Flugzeug hineinfliegt. Des Weiteren können auch der Verstaubungsgrad und der Befestigungswiderstand vom Benutzer bestimmt werden.



Erstmals fliegt das Flugzeug auf den Turm zu und verursacht Beschädigungen in diesem Stockwerk und eine riesige Staubwolke. Nach kurzer Zeit beginnt dann schon der wirkliche Einsturz des Towers. Da wir vom richtigen Modell nicht weit abweichen wollten, stürzen auch bei uns die obersten Stockwerke gemeinsam nach unten. Zusätzlich wird über random zufällige Koordinaten für die „Staubwolken“ gesetzt. Am Schluss liegen am Boden einige kaputte Stockwerksteile.

Quellcode:

```
breed[planes plane]         ;integriert die benötigten patches

breed[floores floore]

breed[kaputte kaputt]

breed[brokens broken]

floores-own[ycor0]          ;eine eigenschaft für die einzellnen Turtles
wird erstellt in der die Anfangs y-koordiante gespeichert wird


globals

[

  zeit                   ; globale Variablen, die noch öfter gebraucht
werden

  hoehe

  grenze

  num-stock

  crash-floor

  floor-hoehe

  k

]


to setup

  clear-all            ; löscht alles, das bisher im Interface zu sehen war

  set-parameters1      ; setzt die 1. Parameter

  berechnen            ; das Unterprogramm berechnen wird aufgerufen

  mochs-gray           ; der STaub wird im Hintergrund schon erschaffen

  setup-plane

  setup-kaputte

  setup-floors

  move-planes

  remove-unnoetige

  set-parameters2

end
```

```
to mochs-gray

        ; wir greifen auf alle patches zu, die iwo hinter dem Turm liegen

        ask patches with [pxcor > 13 and pxcor < 19 and random-pycor < num-
stock - 50 ]

        [

           set pcolor gray ;setzen ihre farbe auf grau (späterer Rauch)

        ]

        ask patches

        [

          if(pycor >= 110)

          [

             set pcolor black ; sollten patches überhalb des turmes grau
werden, setzen wir sie hier wieder schwarz

          ]

        ]

end



to setup-plane

  set-default-shape planes "plane"

  create-planes 1          ; ein Flugzeug wird erstellt

  [

    set xcor 0 set ycor crash-floor ; wir setzen es an die stelle, die sich
der Benutzer aussucht

    set heading 90                  ; wir setzen seine Richtung nach vorne

  ]

end



to setup-kaputte

  set-default-shape kaputte "kaputt"

  create-kaputte (5 * num-stock) ;erst werden die Kaputten steine
aufgestellt

  [

    let gz (int ((who - 1) / num-stock))

    set xcor (gz + 14)
```

```
    set ycor ((who - (gz * num-stock)) - 1)
  ]
end


to setup-floors
  set-default-shape floores "floore"   ;die ganzen Stockwerke kommen über
die Kaputten
  create-floores (5 * num-stock)
  [
    let gz (int ((who - 1) / num-stock))
    set xcor (gz + 9)
    set ycor ((who - (gz * num-stock)) - 1)
  ]
end


to move-planes
  ask planes
  [
    let imax 14    ; 14 weil der turm bei der xcor 14 beginnt
    let it 0
     while [it < imax]
    [
      fd 1                ; damit nicht alles auf einmal geht benutzen wir
eine while-schleife
      set it (it + 1)
      wait 0.5            ; damit das flugzeug nicht zu schnell fliegt
lassen wir es bremsen
    ]
    flugzeug-crash
    ask patches in-radius 8 [set pcolor red] ;;;; Staubwolke
    ask patches in-radius 7 [ set pcolor gray]
    ask patches in-radius 7 [ set pcolor black]
    ask patches in-radius 7 [ set pcolor gray]
    die
```

```
  ]
end


to flugzeug-crash
  ask floores
    [
      ; genau das betroffene STockwerk wird gelöscht (wo das Flugzeug
reinflog)
      if( ycor = crash-floor and xcor > 13 and xcor < 19)
      [
        die
      ]
    ]
end


to remove-unnoetige
  ask kaputte ; die kaputten werden auch zerstört, wo sie nicht mehr
gebraucht werden
  [
    if(xcor > 13 and xcor < 19 and ycor > crash-floor)
    [
      die
    ]
  ]
end


to set-parameters1
  set grenze crash-floor
  set num-stock Nummer_der_Stockwerke
  set crash-floor Stock_in_den_das_Flugzeug_crashed
  set floor-hoehe Hoehe_eines_Stockwerkes
end


to set-parameters2
```

```
  ask floores
  [
   set ycor0 ycor
  ]
end


to go
  fall-down
  delete-hiniges
  tick
  if (ticks > num-stock)
   [
    stop
   ]
end


to fall-down
  ; wir greifen auf das array zeit zu und überprüfen wie viele werte
kleiner sind als der aktuelle tick-wert
  let time ((ticks / num-stock ) * (last zeit ))
  ; der Faktor wie viele STockwerke hinunterfliegen wird errechnet
  set k (length (filter [? <= time] zeit ) - 1)
  ; die untere grenze der ganz bleibenden Stockwerke wird immer wieder
aktuallisiert
  set grenze (crash-floor - k)
  set grenze (max (list 0 grenze)) ; Grenze darf nicht unter null sein
  show time
  ;show k
  ;show grenze
  ask floores with [ycor >= grenze] ; alle stockwerke über der Grenze
werden angesprochen
    [
        set ycor (max (list 0 (ycor0 - k ))) ; und ihre Y-Koordinate wird
um k heruntergesetzt
```

```
    ]

  make-broken

end

to delete-hiniges
  ask kaputte
  [
    if ((ycor >= (crash-floor - k + 1)) and (xcor > 13) and (xcor < 19))
;die kaputten die nicht meh gebraucht werden, werden gelöscht
    [
      die
    ]
  ]
end


to make-broken
  if ( k >= (num-stock - 6) )
  [
    ; die Steine die am Schluss am Boden noch überbleiben werden hier
erstellt
    set-default-shape brokens "broken"
    create-brokens 100
      [
        ; ihre koordinaten werden durch Zufall zugewiesen
        setxy (int(random-normal 16 4))  (int(random-normal 0 2))
      ]
    ask brokens
    [
      if (ycor > 10)
      [
       ; wenn die zufallskoordinaten zu weit oben liegen, so werden die
Steine wieder gelöscht
       die
      ]
    ]
  ]
End

to berechnen
  ; der Matlab-Code für die Berechnung der Zeit wird eingebaut
  set zeit (list 0) ;wir erstelln ein Array für die Zeiten
  set hoehe (list (num-stock * floor-hoehe)) ; und eins für die Höhen
  let imax crash-floor
  ; die Variablen die nur hier gebraucht werden, werden erstellt
  let i 1
  let g 9.81
  let m0 2618000
  let m1 m0 * (num-stock - crash-floor)
  let m2 m0 * (1 - Verstaubungsgrad)
  let P0 Befestigungswiderstand * 1000
  let v0 0
  let v1 0
  let v2 0
  let t1 0
  let t2 0
  let x1 0
  let x2 0
```

```
  while [ i <= imax ]
  [
    set x1 last hoehe
    set t1 last zeit
    set t2 ( sqrt ( v0 * v0 + 2 * floor-hoehe * g ))
    set t2 (t2 + v0)
    set t2 t1 + t2 / g
    set v1 (0 - sqrt ( v0 * v0 + 2 * g * floor-hoehe))
    set v0 ((P0 + (m1 * v1)) / (m1 + m2))
    set zeit lput t2 zeit
    set x2 (x1 - floor-hoehe)
    set hoehe lput x2 hoehe
    set i (i + 1)
    set m1 (m1 + m2)
  ]
  set imax num-stock - crash-floor
  set i 1

  while [ i <= imax ]
  [
    set x1 last hoehe
    set t1 last zeit
    set t2 ( t1 + ((v0 + (sqrt ((v0 * v0) + (2 * floor-hoehe * g))) ) / g))
    set v0 (0 - sqrt ( v0 * v0 + 2 * g * floor-hoehe))
    set zeit lput t2 zeit
    set x2 (x1 - floor-hoehe)
    set hoehe lput x2 hoehe
    set i (i + 1)
    set m1 (m1 + m2)
  ]
  show last zeit
  set imax num-stock
  set i 1
  while [ i <= imax ] ;; Die Kurve wird ausgegeben
  [
    set-current-plot "Abhängigkeit Zeit und Höhe"
    plotxy (item i zeit) (item i hoehe)
    set i (i + 1)
  ]
end
```

Konklusion:

Wir sind der Ansicht, dass die Erklärung der amerikanischen Regierung nicht plausibel ist, da unsere ermessene Einsturzzeiten, basierend auf realitätsgetreuen Daten, nur im freien Fall mit der tatsächlichen Einsturzzeit übereinstimmt.