Physik der sozio-ökonomischen Systeme *mit dem Computer*

JOHANN WOLFGANG GOETHE UNIVERSITÄT
14.11.2025

MATTHIAS HANAUSKE

FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

5. Vorlesung

Plan für die heutige Vorlesung

- Kurze Wiederholung der Inhalte der 4. Vorlesung
- Numerisches Lösen von Differentialgleichungen
- Einführung in die evolutionäre Spieltheorie
  - Die Differentialgleichung eines evolutionären, symmetrischen (2x3)-Spiels
    - Die 19 Zeeman – Klassen
- Anwendungsfelder Spieltheorie
  - Anwendungsfelder in den Wirtschafts- Sozialwissenschaften und Biologie
    - Experimentelle Ökonomie
    - Die Finanzkrise als Falke-Taube Spiel
    - Die Entstehung einer dritten Strategie im Elfmeter-Spiel (Nesken Effekt)
    - Evolutionäre Entwicklung einer Eidechsen Population als symmetrisches (2x3)-Spiel
    - Das Räuber-Beute Spiel und die Lotka-Volterra-Gleichung
    - Die Klimakrise als Populationsdilemma

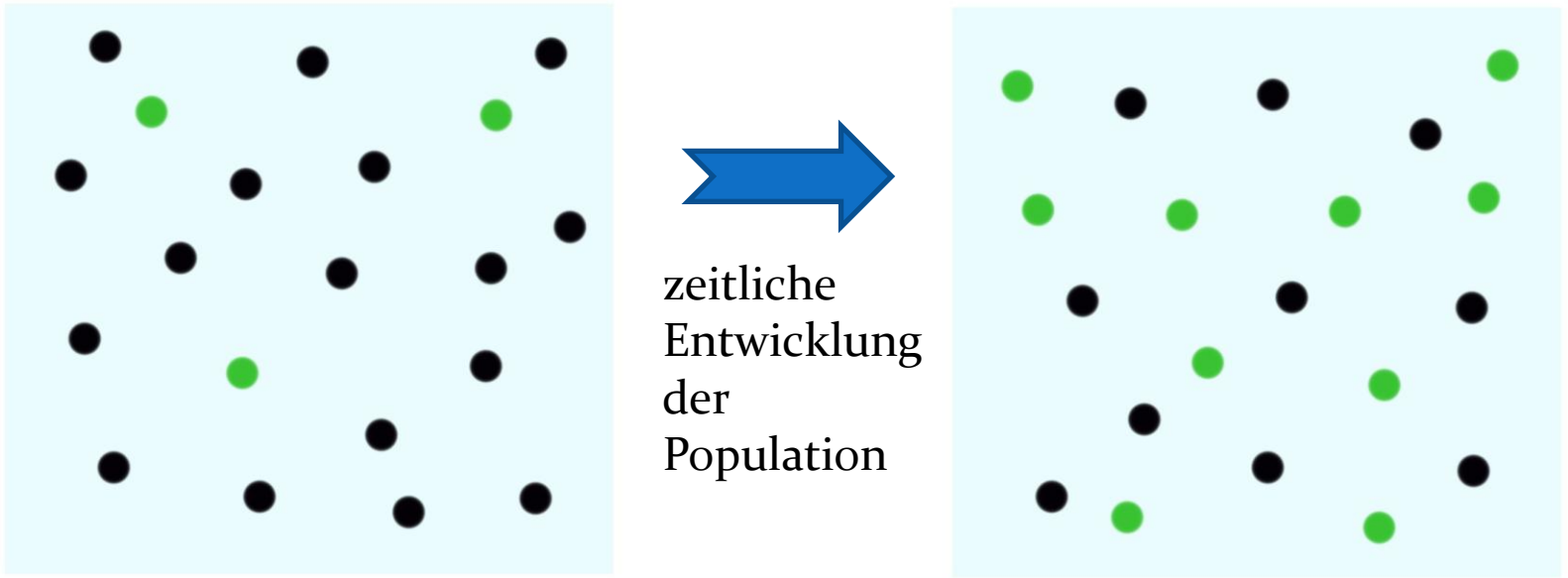
Inhalte Vorlesung 4

- Einführung in die Evolutionäre Spieltheorie
  - Die Differentialgleichung eines evolutionären, symmetrischen (2x2)-Spiels
    - Dominante Spiele
    - Koordinationsspiele
    - Anti-Koordinationsspiele
  - Das System von Differentialgleichung eines evolutionären, unsymmetrischen (2x2)-Spiels (Bi-Matrix Spiele)
    - Eckenspiele (Corner Class Games)
    - Sattelpunktspiele (Saddle Class Games)
    - Zentrumspiele (Center Class Games)

Evolutionäre Spieltheorie

Symmetrische (2x2)-Spiele einer Population

Die evolutionäre Spieltheorie betrachtet die zeitliche Entwicklung des strategischen Verhaltens einer gesamten Spielerpopulation.



x(0)=0.15 x(10)=0.5

Mögliche Strategien: (grün , schwarz), Parameter t stellt die „Zeit“ dar.
x(t) : Anteil der Spieler, die im Zeitpunkt t die Strategie „grün“ spielen.

Nimmt man zusätzlich ein symmetrisches Spiel an ($\hat{\$} := \hat{\$}^A = \left(\hat{\$}^B\right)^T$), in welchem die Auszahlungswerte (Fitness-Werte) der Populationsgruppen gleich sind, so kann man die beiden Gruppen von ihrer mathematischen Struktur her als ununterscheidbare Spielergruppen mit identischen Populationsvektoren $x(t) = y(t)$ annehmen. Die Differentialgleichung schreibt sich dann wie folgt:

$$\frac{dx(t)}{dt} = \left[(\$_{11} - \$_{21})(x - x^2) + (\$_{12} - \$_{22})(1 - 2x + x^2)\right] x(t) =: g(x) \qquad (3)$$

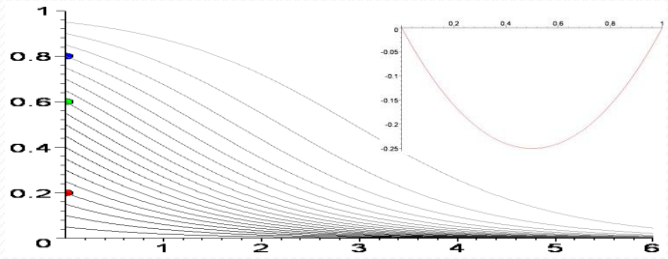
Verallgemeinert man diese Differentialgleichung wieder auf mehr als zwei Strategien, so kann man abkürzend die folgende Formulierung schreiben:

$$\frac{d\vec{x}}{dt} = \hat{\mathbf{x}}\left(\hat{\$}\,\vec{x}\right) - \left(\left(\hat{\$}\,\vec{x}\right)^T \vec{x}\right)\vec{x}$$

Klassifizierung von evolutionären, symmetrischen (2x2)-Spielen

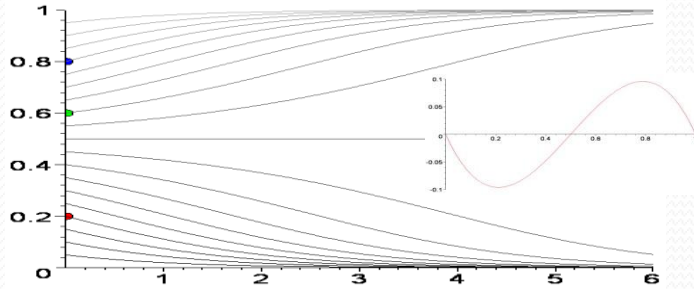
- **Dominante Spiele**
  (2. Strategie dominiert 1.Strategie)

Es existiert ein Nash - Gleichgewicht, welches die anderen Strategien dominiert. ESS bei x=0.



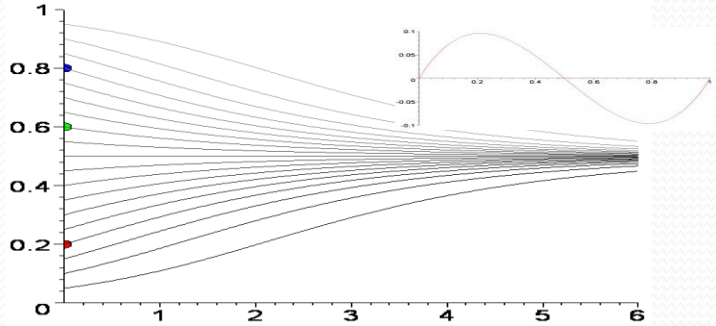
- **Koordinationsspiele**

Es existieren drei Nash – Gleichgewichte und zwei reine ESS, die abhängig von der Anfangsbedingung realisiert werden.



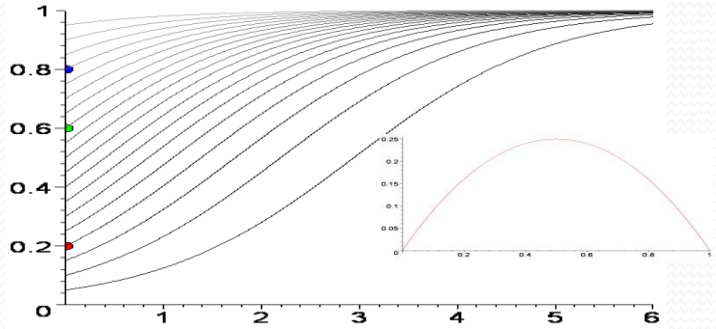
- **Anti – Koordinationsspiele**

Es existieren drei Nash – Gleichgewichte aber nur eine gemischte ESS, die unabhängig von der Anfangsbedingung realisiert wird.

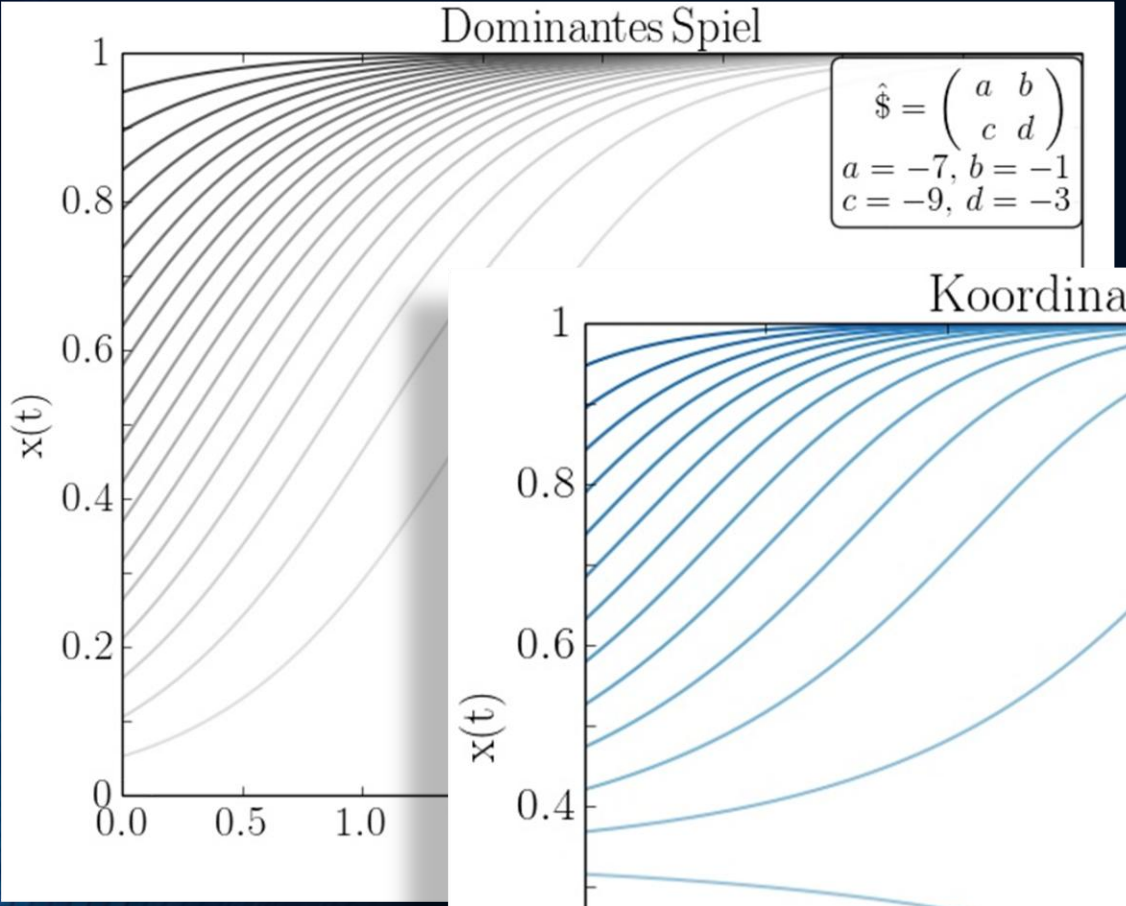


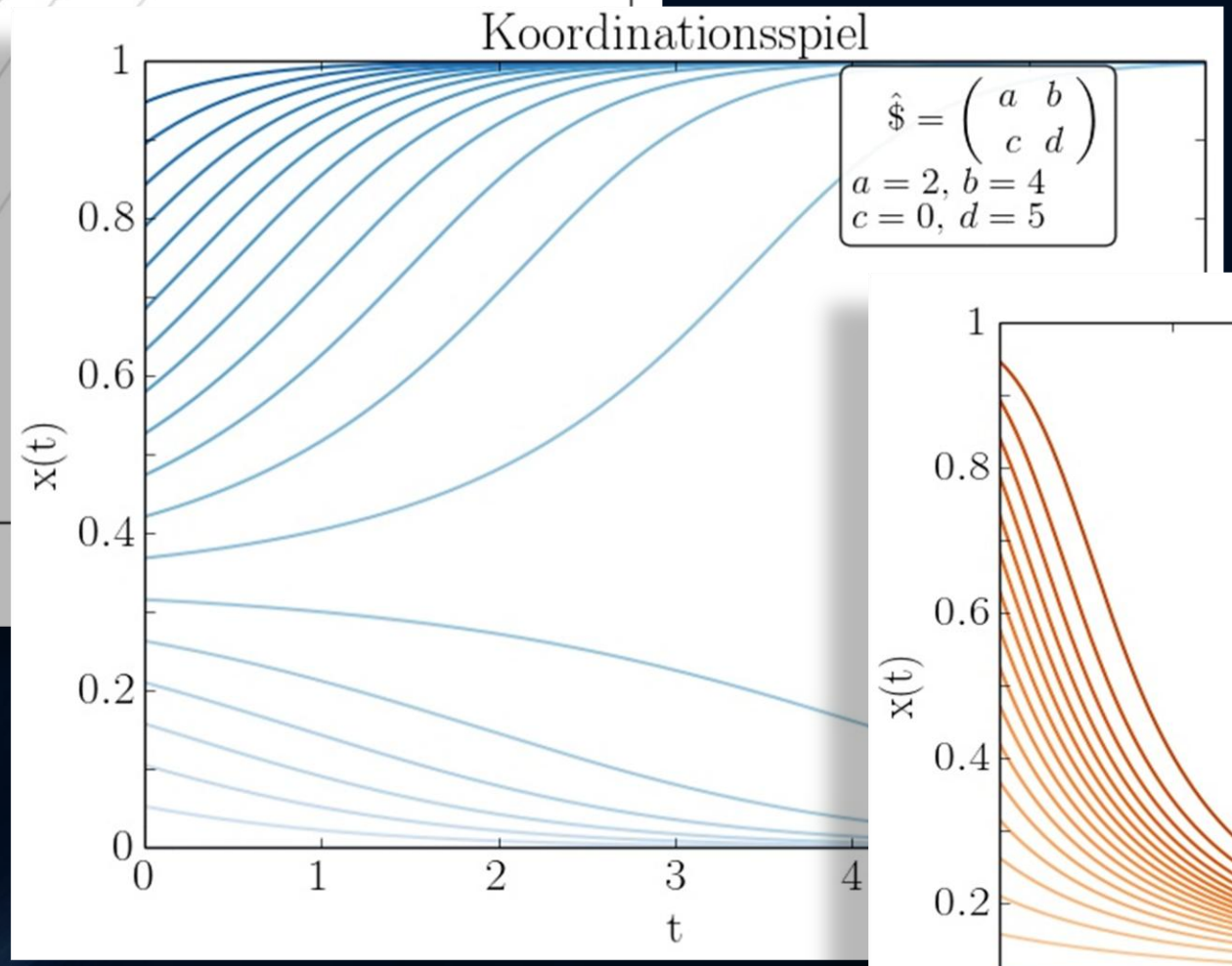
- **Dominante Spiele**
  (1. Strategie dominiert 2.Strategie)

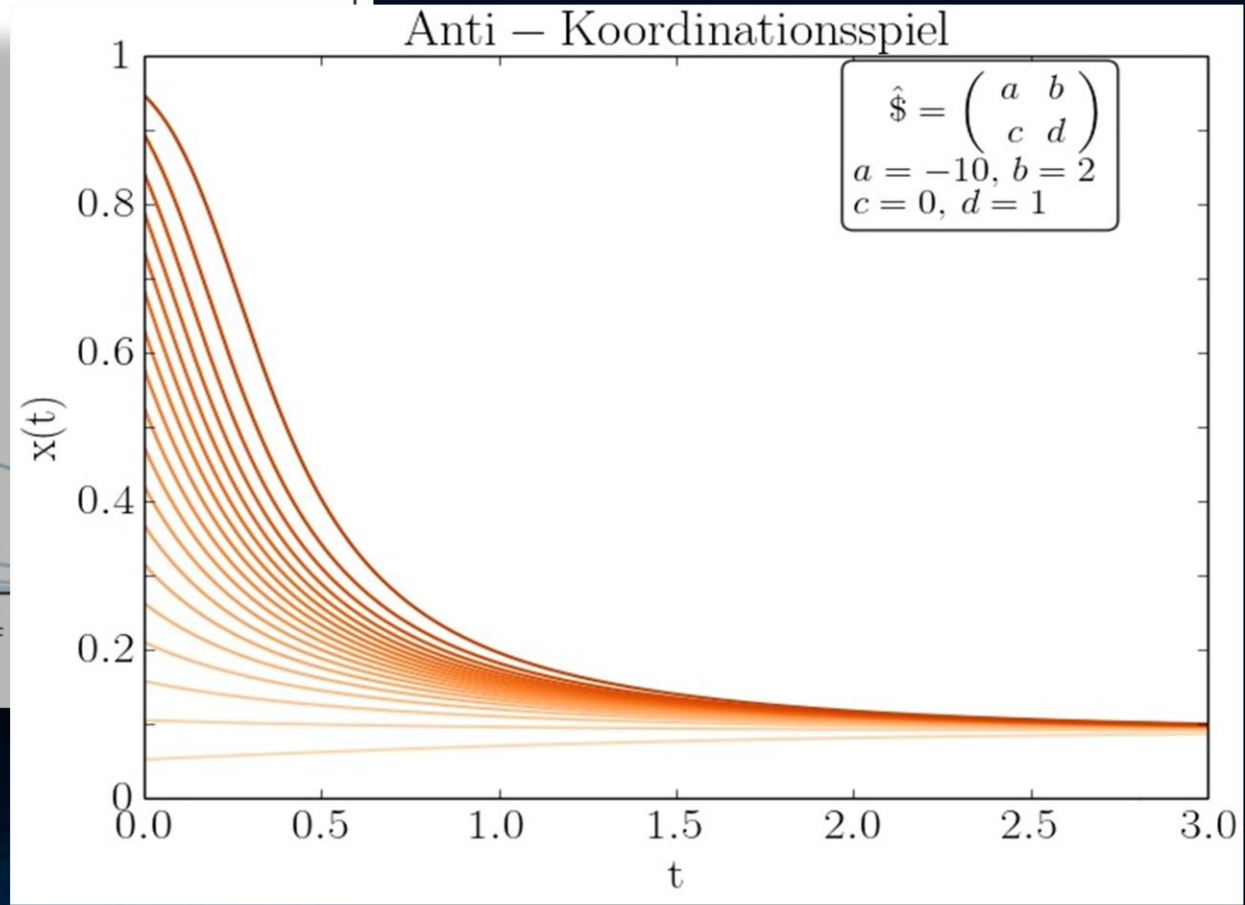
Es existiert ein Nash - Gleichgewicht, welches die anderen Strategien dominiert. ESS bei x=1.



Lösen des evolutionären Spiels mit Python Version evol1.py



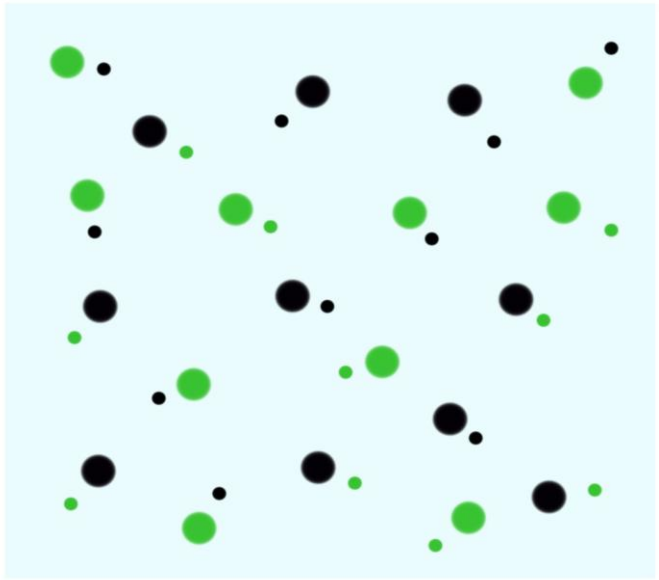




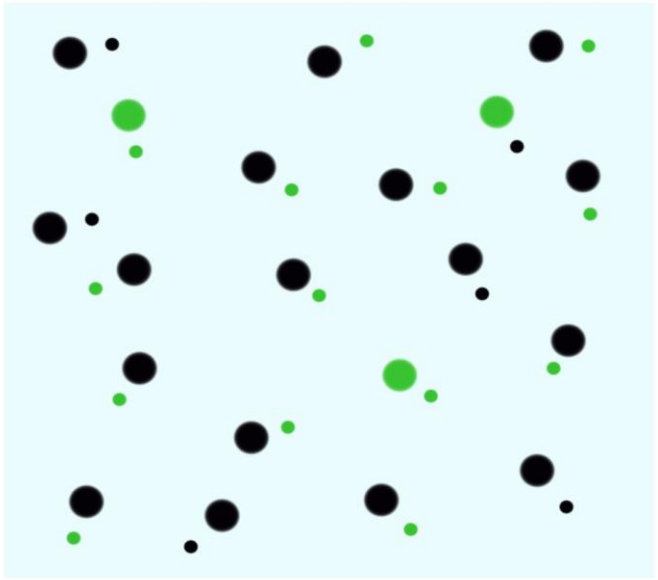
Evolutionäre Spieltheorie

Unsymmetrische (2x2)-Spiele (Bimatrixspiele) zweier Populationen

Bei unsymmetrischen (2x2)-Spielen besteht die zugrundeliegende Population aus zwei Gruppen (hier große und kleine Kreise). Aufgrund der unterschiedlichen Auszahlungsmatrizen können die Populationsgruppen sich in ihren Strategieentscheidungen (grün , schwarz) unterschiedlich entwickeln.



zeitliche Entwicklung der Population



x(0)=0.5 , y(0)=0.5

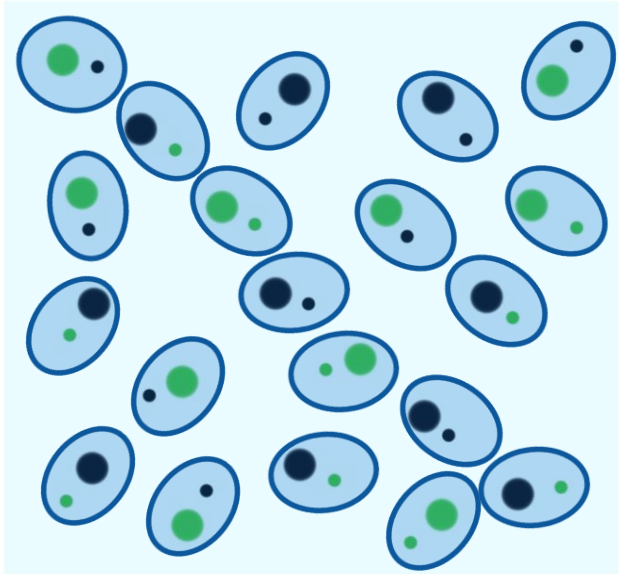
x(10)=0.15 , y(10)=0.7

Mögliche Strategien: (grün , schwarz), Parameter t stellt die „Zeit“ dar.
x(t) : Anteil der großen Spieler, die im Zeitpunkt t die Strategie „grün“ spielen.
y(t) : Anteil der kleinen Spieler, die im Zeitpunkt t die Strategie „grün“ spielen.

Evolutionäre Spieltheorie

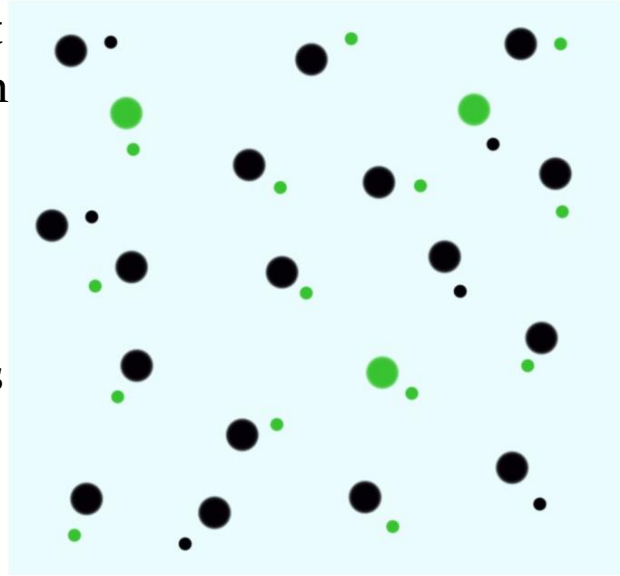
Unsymmetrische (2x2)-Spiele (Bimatrixspiele)

Die einzelnen Akteure innerhalb der betrachteten gesamten Population spielen ein andauernd sich wiederholendes Spiel miteinander, wobei sich jeweils zwei Spieler mit unterschiedlichen Gruppenzugehörigkeiten zufällig treffen, das Spiel spielen und danach zu dem nächsten Spielpartner der anderen Gruppe wechseln .



x(10)=0.5 , y(10)=0.5

Die Anfangspopulation von Spielern spielt zum Zeitpunkt t=0 das erste Mal das Spiel. Es bilden sich stets Zweier-Gruppen aus großen und kleinen Kreisen.



x(10)=0.15 , y(10)=0.7

Das evolutionäre Spiel schreitet voran und die grüne Strategie wird für die kleinen Spieler zunehmend attraktiver (y(10)=0.7), wohingegen sie für die großen Spieler zunehmend weniger attraktiv wird (x(10)=0.15).

Wir beschränken uns im folgenden auf den 2-Strategien Fall ($m_A = m_B = 2$), lassen jedoch weiter eine Unsymmetrie der Auszahlungsmatrix zu. Die beiden Komponenten der zweidimensionalen gruppenspezifischen Populationsvektoren lassen sich dann, aufgrund ihrer Normalisierungsbedingung, auf eine Komponente reduzieren ($x_2^A = 1 - x_1^A$ und $x_2^B = 1 - x_1^B$). Das zeitliche Verhalten der Komponenten der Populationsvektoren (Gruppe A: $x(t) := x_1^A(t)$ und Gruppe B: $y(t) := x_1^B(t)$) wird in der Reproduktionsdynamik mittels des folgenden Systems von Differentialgleichungen beschrieben (siehe z.B. [4], S:116 oder [3], S:69):

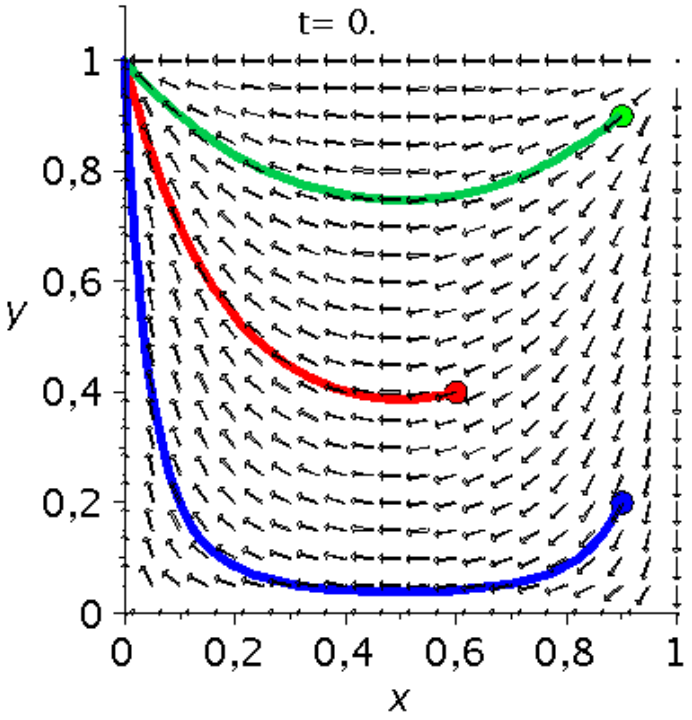
$$\frac{dx(t)}{dt} = \left[\left(\$_{11}^A + \$_{22}^A - \$_{12}^A - \$_{21}^A\right) y(t) + \left(\$_{12}^A - \$_{22}^A\right)\right] \left(x(t) - (x(t))^2\right) =: g_A(x,y)$$

$$\frac{dy(t)}{dt} = \left[\left(\$_{11}^B + \$_{22}^B - \$_{12}^B - \$_{21}^B\right) x(t) + \left(\$_{21}^B - \$_{22}^B\right)\right] \left(y(t) - (y(t))^2\right) =: g_B(x,y)$$

Klassifizierung von Bi-Matrix Spielen

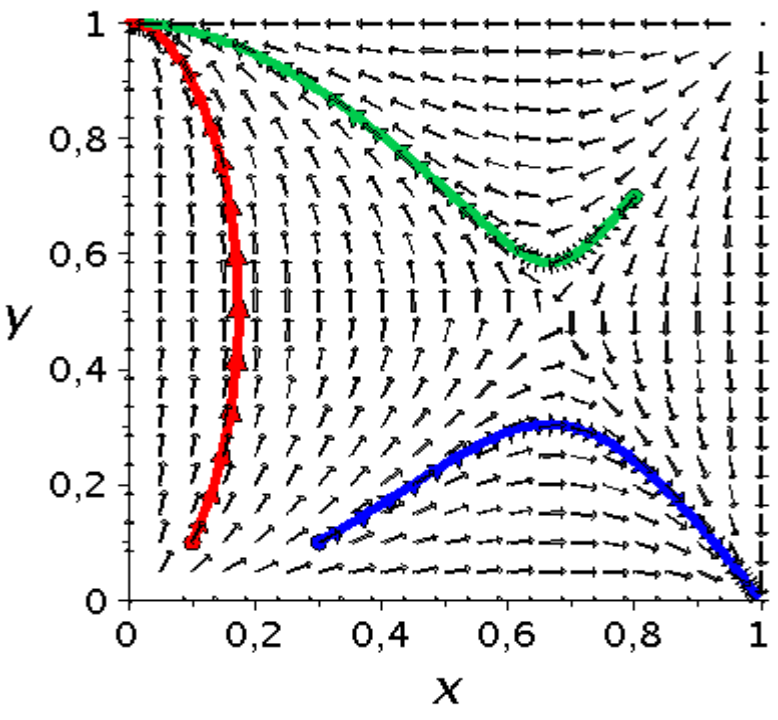
Eckspiele

Die Spielklasse der Gruppe A oder der Gruppe B ist ein *Dominantes Spiel*



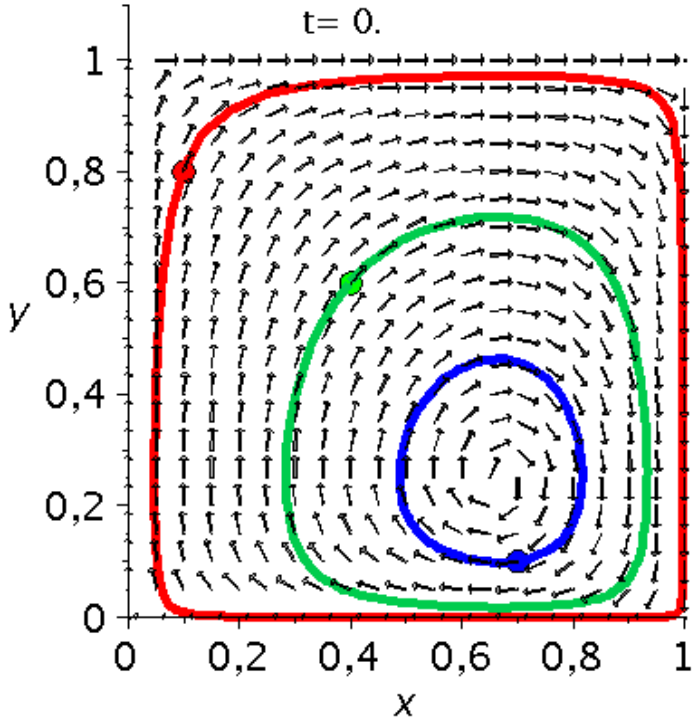
Sattelspiele

Spiel A: *Koordinationspiel*
Spiel B: *Koordinationspiel*
oder
Spiel A: *Anti-Koordinationspiel*
Spiel B: *Anti-Koordinationspiel*



Zentrumsspiele

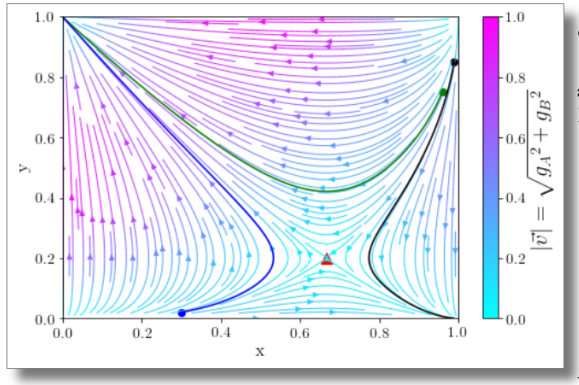
Spiel A: *Koordinationspiel*
Spiel B: *Anti-Koordinationspiel*
oder
Spiel A: *Anti-Koordinationspiel*
Spiel B: *Koordinationspiel*



Jupyter Notebook

Auf der Internetseite der Vorlesung

Evolutionäre unsymmetrische (2 × 2) Spiele (Bi-Matrix Spiele)



Die im oberen Bereich des rechten Panels dieser Vorlesung dargestellten Gleichungen der Reproduktionsdynamik lassen sich nach Spezifikation der Auszahlungsmatrizen $\hat{\$}^A$ und $\hat{\$}^B$ auf unterschiedlichste Populationsspiele anwenden. Generell lassen sich evolutionäre unsymmetrische (2×2) Spiele in die folgenden drei Spielklassen gliedern: Eckenspiele, Sattelpunktspiele und Zentrumsspiele. Ein Eckenspiel liegt vor, falls zumindest eine der Auszahlungsmatrizen der Spielergruppen eine dominante Struktur hat. Aufgrund der Dominanz der Strategie endet die zeitliche Entwicklung der Populationen, unabhängig von der Anfangsbedingungen, in einer Ecke des x-y Populationsvektor Diagramms - die ESS des Eckenspiels. Ein Sattelpunktspiel liegt vor, falls beide Spielergruppen gleichzeitig ein Koordinationsspiel oder Anti-Koordinationsspiel spielen. Bei einem Sattelpunktsspiel das aus zwei Anti-Koordinationsspielen besteht, existieren zwei ESSs ((x=0,y=1) oder (x=1,y=0)) zu denen sich die Populationsgruppen im Laufe der Zeit entwickeln. Welche dieser evolutionär stabilen Strategien erreicht wird, hängt von der Anfangsstrategiewahl der Population ab. Die nebenstehende Abbildung zeigt die zeitliche Entwicklung von drei Anfangszuständen in einem solchen Eckenspiel. Auch bei sehr ähnlichen Werten der Anfangsstrategiewahl kann es geschehen, dass sich die Population im Laufe der Zeit zu unterschiedlichen Ecken entwickelt (siehe grüne und schwarze Trajektorien). Eine besondere Bedeutung hat der Sattelpunkt des Spiels (siehe rotes Dreieck). Die Position des Sattelpunktes im x-y Diagramm entspricht dem Wert des gemischten Nash-Gleichgewichtes bzw. lässt sich durch die Nullstellen der Funktionen $g_A(x,y)$ und $g_B(x,y)$ bestimmen. Bei einem Zentrumsspielen existiert keine ESS, da die Strategiewahl der Population sich im Laufe der Zeit ständig verändert und um ein Zentrum kreist. Die Position dieses Zentrums im x-y Diagramm entspricht dem Wert des gemischten Nash-Gleichgewichtes bzw. lässt sich durch die Nullstellen der Funktionen $g_A(x,y)$ und $g_B(x,y)$ bestimmen (siehe Evolutionäre Spieltheorie unsymmetrischer (2x2)-Spiele).

Weiterführende Links

- Folien der 4.Vorlesung
- Vorlesungsaufzeichnung der 4.Vorlesung: WS 2022/23 bzw. WS 2021/22
- View Jupyter Notebook: Evolutionäre Spieltheorie symmetrischer (2x2)-Spiele
- Download Jupyter Notebook: Evolutionäre Spieltheorie symmetrischer (2x2)-Spiele
- Download Python Programm: Evolutionäre Spieltheorie symmetrischer (2x2)-Spiele (Version 1 , Version 2 , Version 3)
- View Jupyter Notebook: Evolutionäre Spieltheorie unsymmetrischer (2x2)-Spiele
- Download Jupyter Notebook: Evolutionäre Spieltheorie unsymmetrischer (2x2)-Spiele
- Download Python Programm: Evolutionäre Spieltheorie unsymmetrischer (2x2)-Spiele (Version 1 , Version 2 , Version 3)

Physik der sozio-ökonomischen Systeme mit dem Computer

(Physics of Socio-Economic Systems with the Computer)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main

(Wintersemester 2025/26)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 27.10.2025

Erster Vorlesungsteil:

Klassifizierung evolutionärer Bi-Matrix Spiele (unsymmetrische (2×2)-Spiele)

Einführung

In diesem Unterkapitel werden die unterschiedlichen Spieltypen evolutionärer Bi-Matrix Spiele (unsymmetrische (2×2)-Spiele) klassifiziert. Ausgangspunkt sind die folgenden allgemeinen Auszahlungsmatrizen der Spielergruppen A und B. Da es sich um unsymmetrische Auszahlungsmatrizen nehmen wir das Folgende an: $\hat{\$}^B \neq \left(\hat{\$}^A\right)^T$.

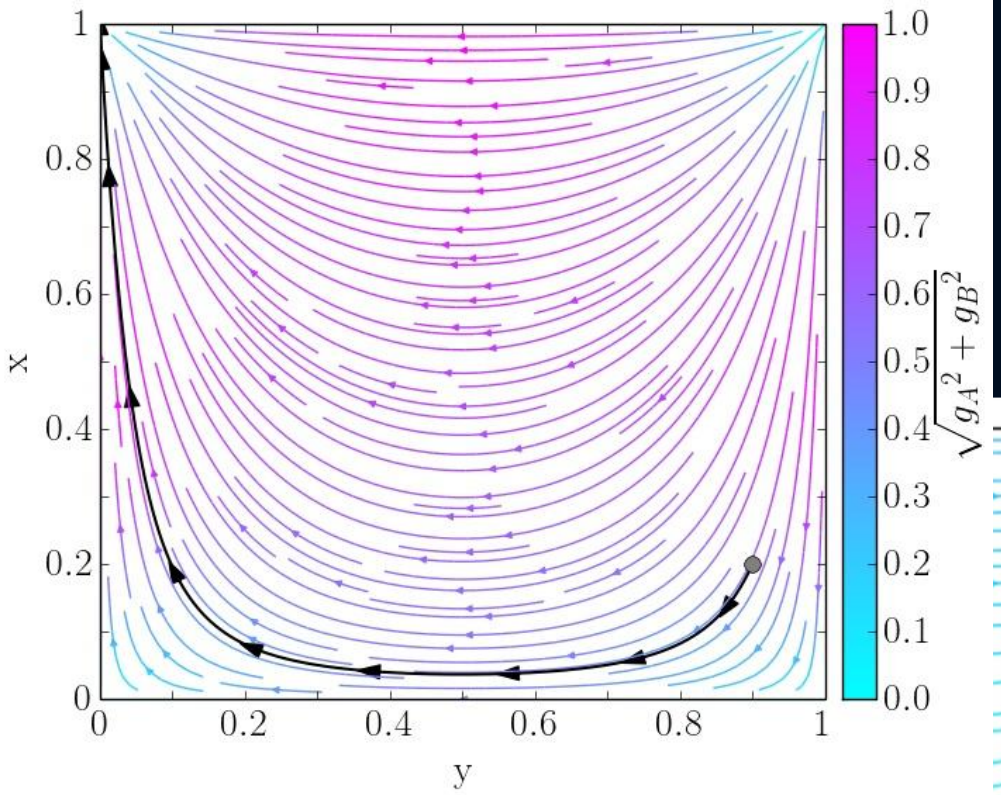
$$\hat{\$}^A = \begin{pmatrix} \$_{11}^A & \$_{12}^A \\ \$_{21}^A & \$_{22}^A \end{pmatrix} \quad , \quad \hat{\$}^B = \begin{pmatrix} \$_{11}^B & \$_{12}^B \\ \$_{21}^B & \$_{22}^B \end{pmatrix} \tag{1}$$

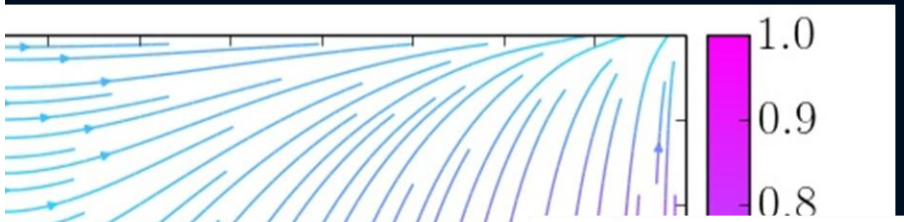
Unsymmetrische (2×2) Spiele lassen sich in die folgenden Spielklassen gliedern:

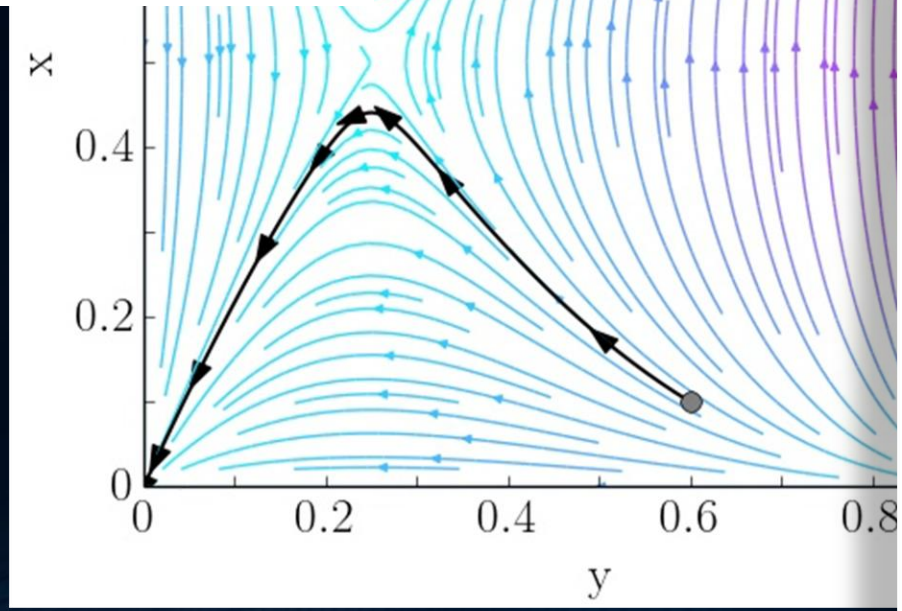
Die Klasse der Eckenspiele (engl.: corner class games)

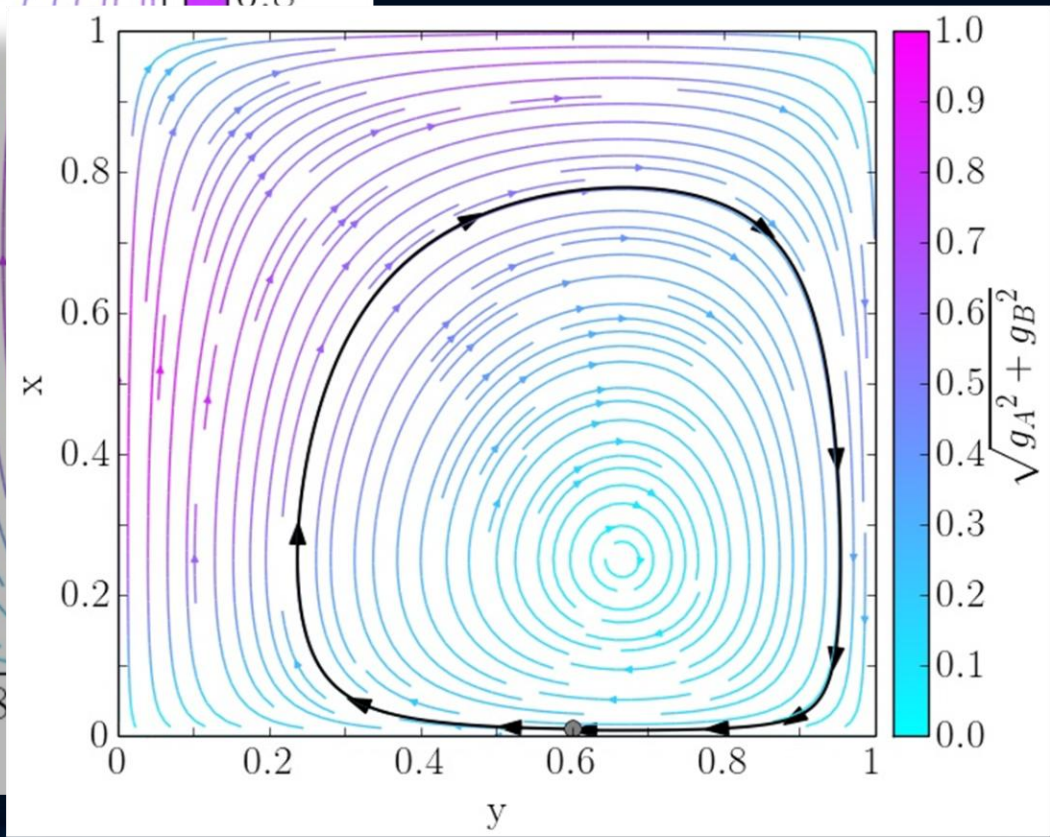
Ein Eckenspiel liegt vor, falls eine der Auszahlungsmatrizen der Spielergruppen eine dominante Struktur hat; falls $\$^A$ oder $\$^B$ ein dominantes Spiel ist.

Bi-Matrix Spiele mit den Python Programmen bimatrix1.py und bimatrix1a.py



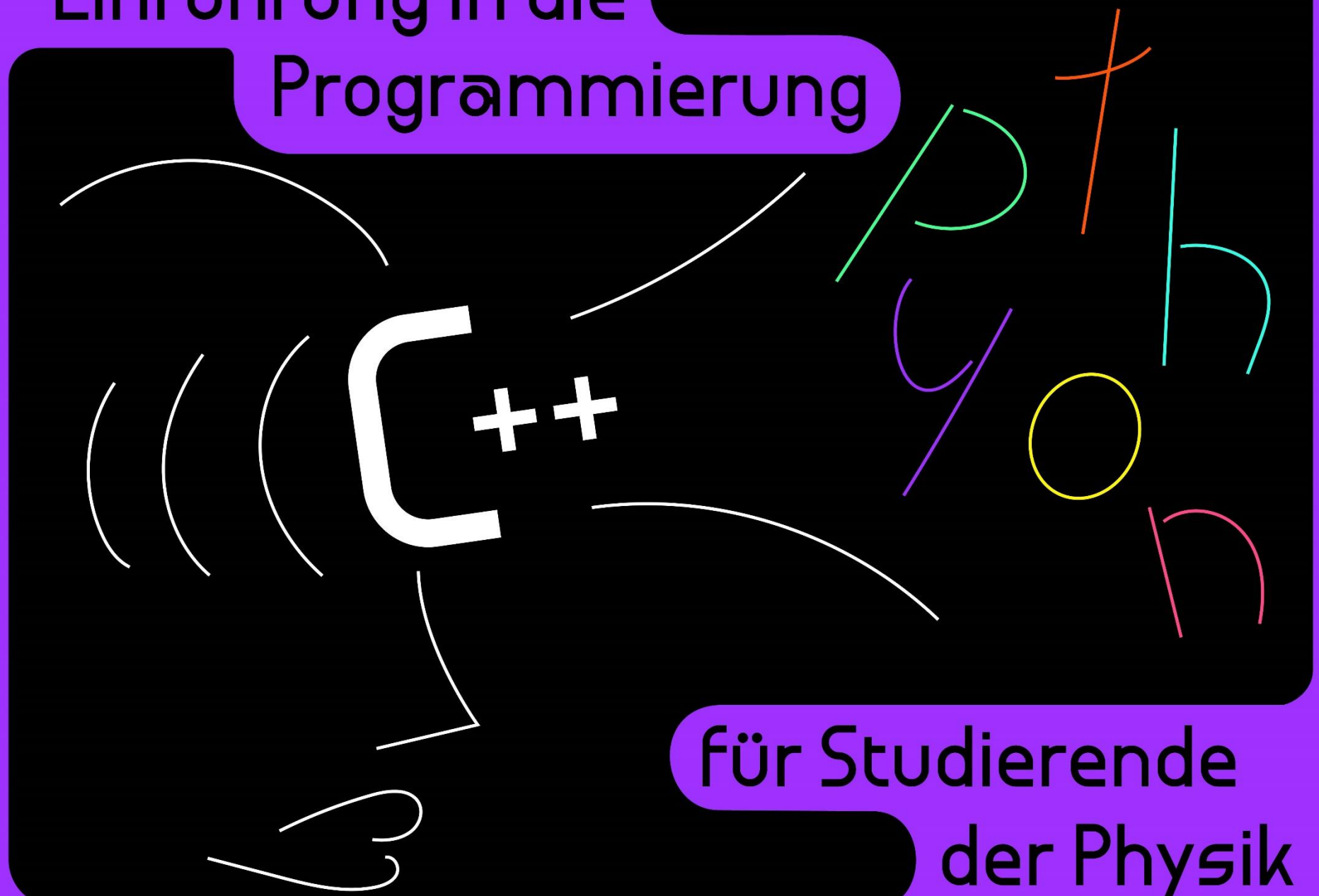






Einführung in die Programmierung

http://itp.uni-frankfurt.de/~hanauske/VPROG/index.html



für Studierende der Physik

Differentialgleichungen: Numerische Lösung von Anfangswertproblemen

Im vorigen Unterpunkt hatten wir die Bewegung einzelner Teilchen in einer Kiste simuliert. In der Physik ist die zeitliche Entwicklung eines Systems of in Form von Differentialgleichungen (DGLs) gegeben. In diesem Unterpunkt betrachten wir das numerische Lösen einer Differentialgleichung erster Ordnung der Form

$$\dot{y}(t) = \frac{dy(t)}{dt} = f(t, y(t)) \quad \text{, mit: } a \leq t \leq b\,,\ y(a) = \alpha \quad .$$

Die Funktion $f(t, y(t))$ bestimmt die DGL und somit das Verhalten der gesuchten Funktion $y(t)$. Es wird hierbei vorausgesetzt, dass $f(t, y(t))$ auf einer Teilmenge $\mathcal{D} = \{(t, y) | a \leq t \leq b\,,\ -\infty \leq y \leq \infty\}$ kontinuierlich definiert ist. Weiter wird angenommen, dass das so definierte Anfangswertproblem "well-posed" ist und eine eindeutige Lösung $y(t)$ existiert ("well-posed" bedeutet hier, dass die Differentialgleichung eine Struktur hat, bei der kleine Störungen im Anfangszustand nicht exponentiell anwachsen). Wir hatten bereits gesehen, wie man Differentialgleichungen mittels Jupyter Notebooks und SymPy DGLs analytisch löst (siehe Jupyter Notebooks und das Rechnen mit symbolischen Ausdrücken). Nicht jede DGL lässt sich analytisch lösen und falls der Befehl "dsolve()" keine sinnvollen Resultate liefert, muss man die zeitliche Entwicklung der Funktion $y(t)$ numerisch berechnen. Die numerische Lösung der DGL kann man sich auch direkt in Python mittels der Methode "integrate.odeint()" berechnen (Python-Modul "scipy"). Möchte man die Lösung jedoch in einem C++ Programm berechnen, so ist man auf die Anwendung eines numerischen Verfahrens angewiesen.

Das einfache Euler Verfahren zum Lösen einer DGL

Das wohl einfachste Verfahren zum Lösen einer DGL erster Ordnung ist die Euler Methode. Hierzu schreibt man die DGL als eine Differenzengleichung um

$$\frac{dy(t)}{dt} = f(t, y(t)) \;\rightarrow\; \Delta y = f(t, y) \cdot \Delta t \;\rightarrow\; \Delta y = h \cdot f(t, y)$$

und unterteilt das Zeitintervall $[a, b]$ in $N + 1$ äquidistante Zeit-Gitterpunkte $(t_0, t_1, t_2, \ldots, t_N)$, wobei $t_i = a + i\,h \quad \forall i = 0, 1, 2, \ldots, N$. Im Algorithmus der Euler Methode startet man bei $t = t_0$ und $y = y_0 = \alpha$ (Anfangsbedingungen des Systems) und erhöht dann iterativ die Zeit t um den Wert von h. Den neuen y-Wert erhält man mittels $y_1 = y_0 + h \cdot f(t_0, y_0)$ und man führt das Verfahren so lange aus, bis man an den letzten zeitlichen Gitterpunkt gelangt.

Betrachten wir z.B. die einfache Differentialgleichung

$$\frac{dy(t)}{dt} = f(t, y(t)) = -y(t) \quad ,$$

die den exponentiellen Abfall einer Funktion $y(t)$ beschreibt. Obwohl sich die allgemeine Lösung der DGL einfach bestimmen lässt ($y(t) = \alpha \cdot e^{-t}$, mit $\alpha = y(0)$), möchten wir die DGL auf numerischem Wege lösen. Das folgende C++ Programm benutzt die Eulermethode und entwickelt die obere Differentialgleichung im Zeitintervall $[a, b] = [0, 2]$ mittels 101 Gitterpunkten. Die simulierten Daten werden dann, zusammen mit der analytischen Lösung im Terminal ausgegeben.

Numerische Verfahren zum Lösen von Differentialgleichung erster Ordnung

Das Euler Verfahren:

$$y_{i+1} = y_i + h \cdot f(t_i, y_i)$$

Mittelpunktmethode:

$$y_{i+1} = y_i + h \cdot \left[f\left(t_i + \frac{h}{2}, y_i + \frac{h}{2} f(t_i, y_i)\right)\right]$$

Modifizierte Euler Methode:

$$y_{i+1} = y_i + \frac{h}{2} \cdot [f(t_i, y_i) + f(t_{i+1}, y_i + h\, f(t_i, y_i))]$$

Runge-Kutta Ordnung vier:

$$y_{i+1} = y_i + \frac{1}{6} \cdot (k_1 + 2\,k_2 + 2\,k_3 + k_4) \quad \text{, wobei:}$$

$$k_1 = h\, f(t_i, y_i)$$

$$k_2 = h\, f\left(t_i + \frac{h}{2}, y_i + \frac{1}{2} k_1\right)$$

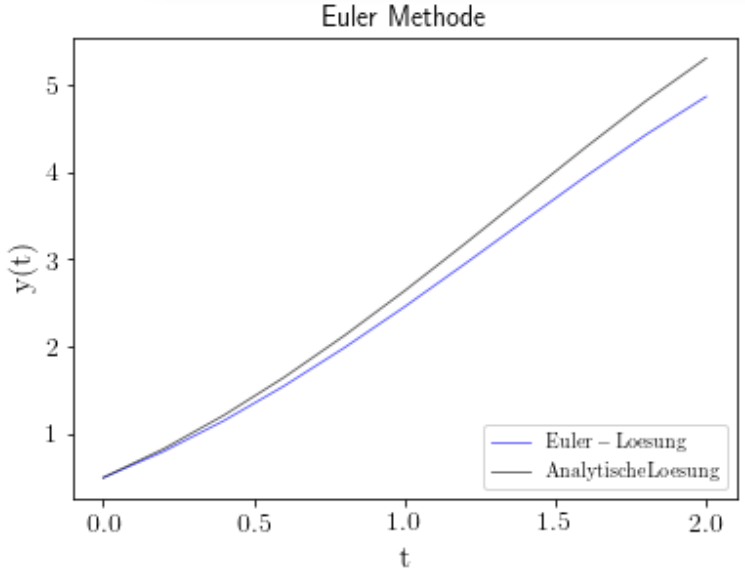
$$k_3 = h\, f\left(t_i + \frac{h}{2}, y_i + \frac{1}{2} k_2\right)$$

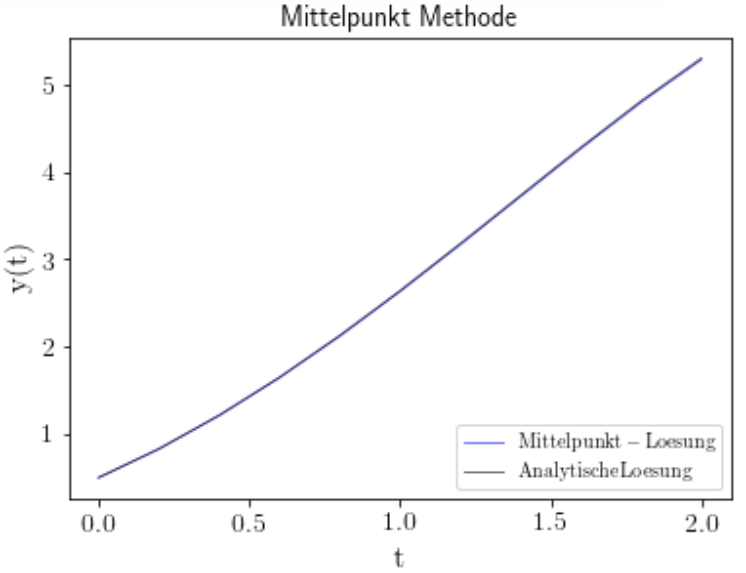
$$k_4 = h\, f\left(t_{i+1}, y_i + k_3\right)$$

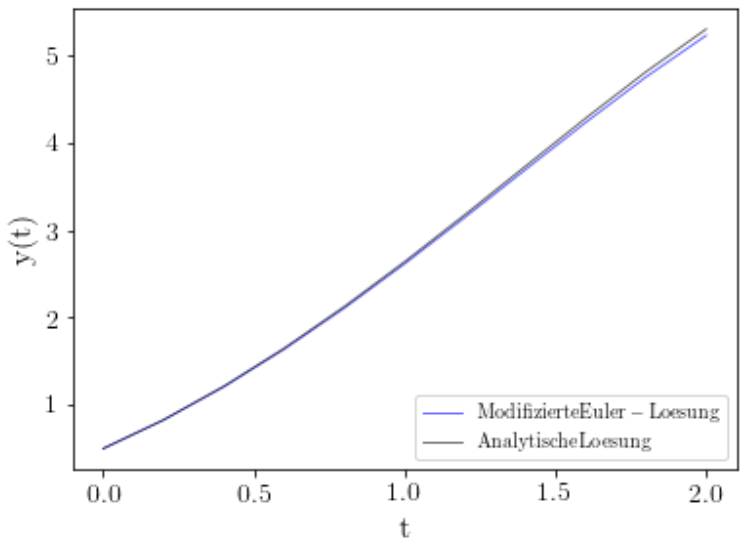
Anwendung auf die DGL: $\frac{dy}{dt} = y - t^2 + 1$

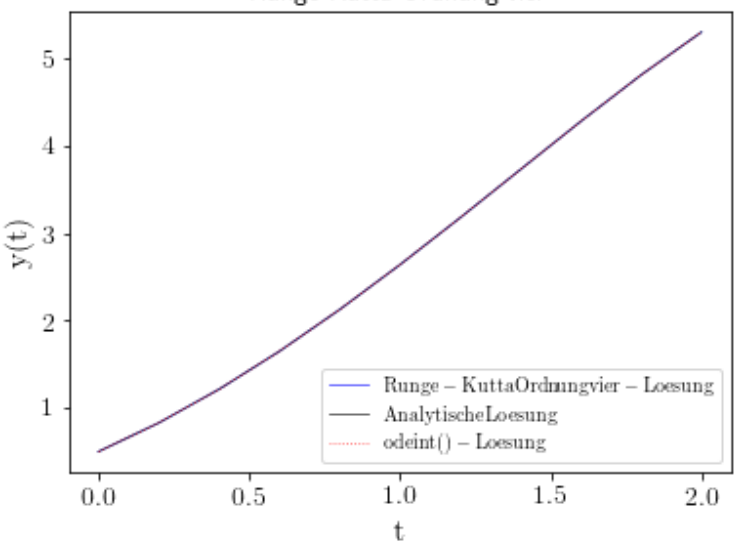
```
 * Zeitentwicklung der fuer
 *  Ausgabe zum Plotten mittels Python Jupyter Notebook DGL_1.ipynb:  ./a.out > DGL_1.dat
 */
#include <stdio.h>                                          // Standard Input- und Output Bibliothek in C, z.B. printf(...)
#include <cmath>                                            // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

double f(double t, double y){                               // Definition der Funktion f(t,x)
    double wert;
    wert = y - pow(t,2) +1;                                 // Eigentliche Definition der Funktion
    return wert;                                            // Rueckgabewert der Funktion f(t,x)
}                                                           // Ende der Funktion f(t,x)

double y_analytisch(double t, double alpha){                // Analytische Loesung der DGL
    double wert;                                            // bei gegebenem Anfangswert y(a)=alpha
    wert = (alpha + (pow(t,2) + 2*t + 1)*exp(-t) -1)*exp(t); // Eigentliche Definition der analytische Loesung
    return wert;                                            // Rueckgabewert
}                                                           // Ende der Definitiom

int main(){                                                 // Hauptfunktion
    double a = 0;                                           // Untergrenze des Zeit-Intervalls [a,b] in dem die Loesung berechnet werden soll
    double b = 2;                                           // Obergrenze des Intervalls [a,b]
    int N = 10;                                             // Anzahl der Punkte in die das t-Intervall aufgeteilt wird
    double h = (b - a)/N;                                   // Abstand dt zwischen den aequidistanten Punkten des t-Intervalls (h=dt)
    double alpha = 0.5;                                     // Anfangswert bei t=a: y(a)=alpha
    double t;                                               // Aktueller Zeitwert
    double y_Euler = alpha;                                 // Deklaration und Initialisierung der numerischen Loesung der Euler Methode
    double y_Midpoint = alpha;                              // Deklaration und Initialisierung der numerischen Loesung der Mittelpunkt Methode
    double y_Euler_M = alpha;                               // Deklaration und Initialisierung der numerischen Loesung der modifizierte Euler Methode
    double y_RungeK_4 = alpha;                              // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double k1, k2, k3, k4;                                  // Deklaration der vier Runge-Kutta Parameter

    printf("# 0: Index i \n# 1: t-Wert \n# 2: Euler Methode \n");                 // Beschreibung der ausgegebenen Groessen
    printf("# 3: Mittelpunkt Methode \n# 4: Modifizierte Euler Methode \n");     // Beschreibung der ausgegebenen Groessen
    printf("# 5: Runge-Kutta Ordnung vier  \n# 6: Analytische Loesung  \n");     // Beschreibung der ausgegebenen Groessen

    for(int i = 0; i <= N; ++i){                                                  // for-Schleife ueber die einzelnen Punkte des t-Intervalls
        t = a + i*h;                                                              // Zeit-Parameter wird um h erhoeht
        printf("%3d %19.15f %19.15f %19.15f %19.15f %19.15f %19.15f\n",i, t, y_Euler, y_Midpoint, y_Euler_M, y_RungeK_4, y_analytisch(t,0.5)); // Ausgaben der Loesung

        y_Euler = y_Euler + h*f(t,y_Euler);                                       // Euler Methode
        y_Midpoint = y_Midpoint + h*f(t+h/2,y_Midpoint+h/2*f(t,y_Midpoint));      // Mittelpunkt Methode
        y_Euler_M = y_Euler_M + h/2*( f(t,y_Euler_M) + f(t+h,y_Euler_M+h*f(t,y_Euler_M)) ); // Modifizierte Euler Methode

        k1 = h*f(t,y_RungeK_4);                                                   // Runge-Kutta Parameter 1
        k2 = h*f(t+h/2,y_RungeK_4+k1/2);                                          // Runge-Kutta Parameter 2
        k3 = h*f(t+h/2,y_RungeK_4+k2/2);                                          // Runge-Kutta Parameter 3
        k4 = h*f(t+h,y_RungeK_4+k3);                                              // Runge-Kutta Parameter 4
        y_RungeK_4 = y_RungeK_4 + (k1 + 2*k2 + 2*k3 + k4)/6;                      // Runge-Kutta Ordnung vier Methode
    }                                                                             // Ende for-Schleife ueber die einzelnen Punkte des t-Intervalls
}                                                                                 // Ende der Hauptfunktion
```

Anwendung auf die DGL: $\frac{dy}{dt} = y - t^2 + 1$









Einführung in die Programmierung für Studierende der Physik

(Introduction to Programming for Physicists)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main

(Sommersemester 2025)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Numerisches Lösen einer DGL erster Ordnung mit Python

Erster Vorlesungsteil:

Numerisches Lösen von Differentialgleichungen (das Anfangswertproblem)

Dieses Jupyter Notebook basiert auf den Materialien der Vorlesung "Einführung in die Programmierung für Studierende der Physik (Introduction to Programming for Physicists)" (siehe Vorlesung 8).

Allgemeine Betrachtungen

Zunächst wird das Python Modul "sympy" eingebunden, das ein Computer-Algebra-System für Python bereitstellt und eine Vielzahl an symbolischen Berechnungen im Bereich der Mathematik und Physik relativ einfach möglich macht (weiteres siehe Vorlesung 6 Jupyter Notebooks und das Rechnen mit symbolischen Ausdrücken). Falls Sie das "sympy" Modul das erste Mal verwenden, müssen Sie es zunächst in Ihrer Python 3 Umgebung installieren (z.B. in einem Linux Terminal mit "pip3 install sympy").

```
from sympy import *
init_printing()
```

Wir betrachten in diesem Jupyter Notebook das numerische Lösen einer Differentialgleichung (DGL) erster Ordnung der Form

$$\dot{y}(t) = \frac{dy(t)}{dt} = f(t, y(t)) \quad \text{, mit: } a \leq t \leq b\,,\ y(a) = \alpha \quad . \tag{1}$$

Die Funktion $f(t, y(t))$ bestimmt die DGL und somit das Verhalten der gesuchten Funktion $y(t)$. Es wird hierbei vorausgesetzt, dass $f(t, y(t))$ auf einer Teilmenge

Physik der sozio-ökonomischen Systeme mit dem Computer

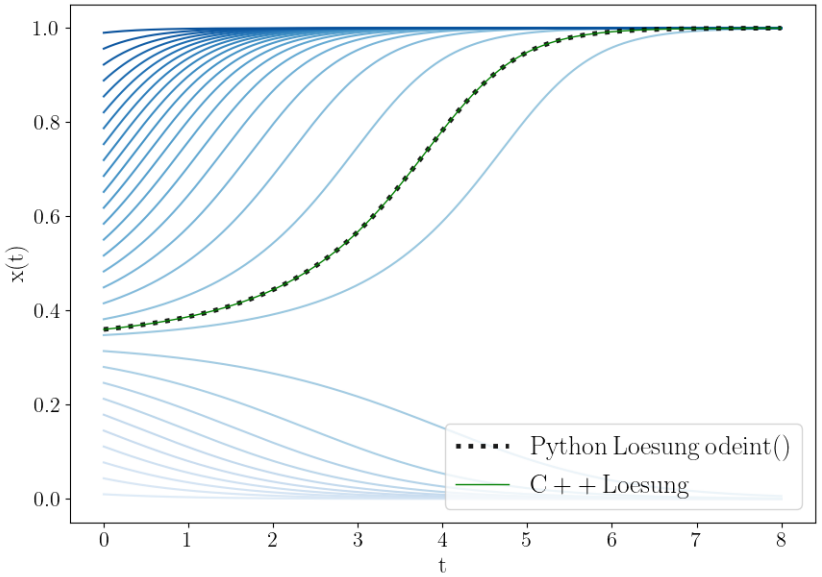
(Physics of Socio-Economic Systems with the Computer)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main

(Wintersemester 2025/26)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 22.08.2025



Anwendung: Evolutionäre Spieltheorie

Download Jupyter Notebook VPSOC_DGL_1.ipynb
View Jupyter Notebook VPSOC_DGL_1.html
Download C++ Programm evol1.cpp

Erster Vorlesungsteil:

Numerisches Lösen von Differentialgleichungen (das Anfangswertproblem)

Dieses Jupyter Notebook basiert auf den Materialien der Vorlesung "Einführung in die Programmierung für Studierende der Physik (Introduction to Programming for Physicists)" (siehe Vorlesung 8). Das Anfangswertproblem wird zunächst allgemein behandelt und am Ende auf die Differentialgleichung der Replikatordynamik der evolutionären Spieltheorie angewendet.

Allgemeine Betrachtungen

Zunächst wird das Python Modul "sympy" eingebunden, das ein Computer-Algebra-System für Python bereitstellt und eine Vielzahl an symbolischen Berechnungen im Bereich der Mathematik und Physik relativ einfach möglich macht (weiteres siehe Vorlesung 6 Jupyter Notebooks und das Rechnen mit symbolischen Ausdrücken). Falls Sie das "sympy" Modul das erste Mal verwenden, müssen Sie es zunächst in Ihrer Python 3 Umgebung installieren (z.B. in einem Linux Terminal mit "pip3 install sympy").

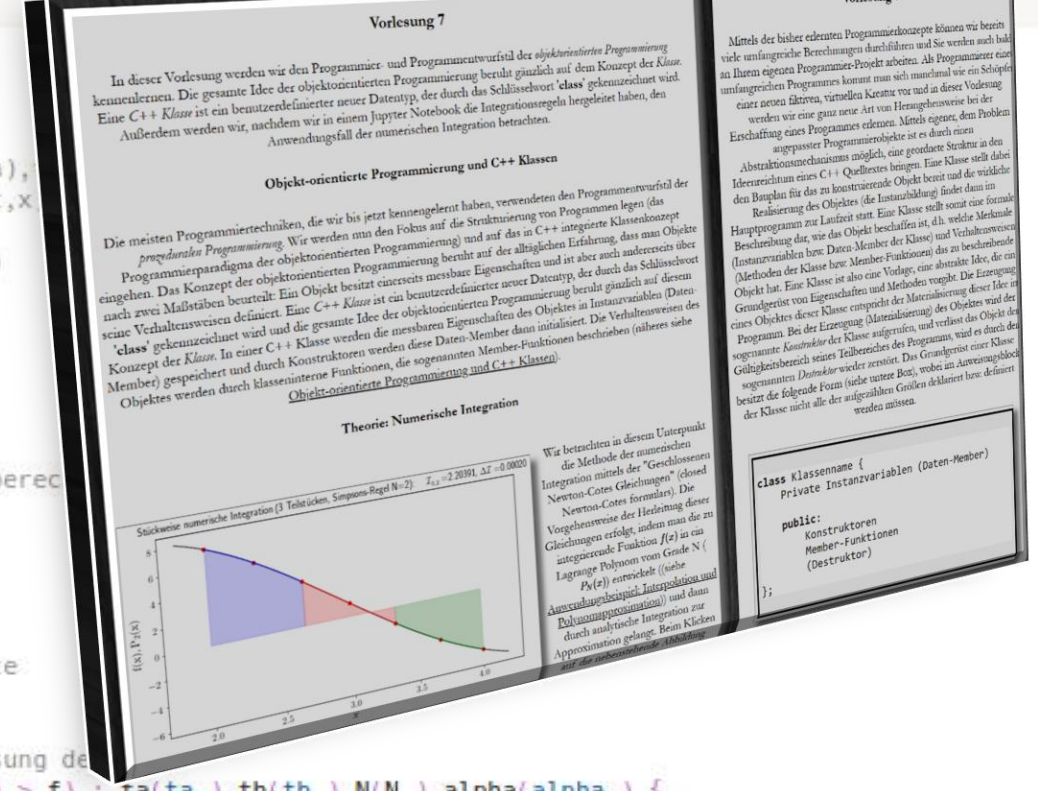
```
from sympy import *
init_printing()
```

Wir betrachten in diesem Jupyter Notebook das numerische Lösen einer Differentialgleichung (DGL) erster Ordnung der Form

$$\dot{y}(t) = \frac{dy(t)}{dt} = f(t, y(t)) \quad \text{, mit: } a \leq t \leq b \,,\; y(a) = \alpha \quad . \tag{1}$$

C++ Programm
evol1.cpp

```cpp
// Klasse dsolve
/* Berechnung der Loesung einer Differentialgleichung der Fo
 * mittels Runge-Kutta Ordnung vier Verfahren
 * Verfahren zur Loesung der DGL ist in eine Klasse a
 * Zeitentwicklung der fuer unterschiedliche t-Werte
 * Konstruktor: dsolve(Anfangszeit ta, Endzeit tb, An                angswert alpha=x(ta),
 * Anwendungsfall: Evolutionäre Spieltheorie eines sy                chen (2x2)-Spiels, g(x)=g(t,x)=f(t,x
 */
#include <stdio.h>              // Standard Input- und Output Bibliothek in C, z.B. printf(...)
#include <cmath>                // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
#include <vector>               // Vector-Container der Standardbibliothek
#include <functional>           // Funktionen in der Argumentenliste von Funktionen
using namespace std;            // Benutze den Namensraum std

class dsolve{                   //Definition der Klasse 'dsolve'
    double ta = 0;              // Untergrenze des Zeit-Intervalls [ta,tb] in dem die Loesung berec
    double tb = 2;              // Obergrenze des Intervalls [ta,tb]
    int N = 10;                 // Anzahl der Punkte in die das t-Intervall aufgeteilt wird
    double alpha = 0.5;         // Anfangswert bei t=ta: x(ta)=alpha

    vector<double> x;           // Deklaration eines double Vektors zum speichern der Loesung
    vector<double> Zeit;        // Deklaration eines double Vektors zum speichern der Zeit-Werte

    public:
        // Konstruktor mit fünf Argumenten (Initialisierung der Parameter, Berechnung der Loesung de
        dsolve(double ta_, double tb_, int N_, double alpha_, function< double(double, double) > f) : ta(ta_),tb(tb_),N(N_),alpha(alpha_) {
            double h = (tb - ta)/N;                             // Abstand dt zwischen den aequidistanten Punkten des t-Intervalls (h=dt)
            double k1, k2, k3, k4;                              // Deklaration der vier Runge-Kutta Parameter
            Zeit.push_back(ta_);                                // Zum Zeit-Vektor die Anfangszeit eintragen
            x.push_back(alpha_);                                // Zum x-Vektor den Anfangswert alpha_=x(ta) eintragen

            for(int i=0; i < N; ++i){                           // for-Schleife ueber die einzelnen Punkte des t-Intervalls
                k1 = h*f(Zeit[i],x[i]);
                k2 = h*f(Zeit[i]+h/2,x[i] + k1/2);
                k3 = h*f(Zeit[i]+h/2,x[i] + k2/2);
                k4 = h*f(Zeit[i]+h,x[i] + k3);

                x.push_back(x[i] + (k1 + 2*k2 + 2*k3 + k4)/6);
                Zeit.push_back(ta + (i+1)*h);
            }
        };

        const vector<double>& get_x() const { return x; }
        const vector<double>& get_zeit() const { return Zeit; }
};
```




```cpp
// Definition der Funktion g(t,x), Gleichung der DGL dx/dt=g(t,x)=f(t,x)
double g(double t, double x){
    double a = 2;               // Auszahlungsparameter a des symmetrischen (2x2)-Spiels
    double b = 4;               // Auszahlungsparameter b des symmetrischen (2x2)-Spiels
    double c = 0;               // Auszahlungsparameter c des symmetrischen (2x2)-Spiels
    double d = 5;               // Auszahlungsparameter d des symmetrischen (2x2)-Spiels
    double wert =  ( (a-c)*(x-x*x) + (b-d)*(1.0 - 2.0*x + x*x) )*x;
    return wert;                // Rueckgabewert der Funktion
}                               // Ende der Funktion g(t,x)

int main(){                                                 // Hauptfunktion
    dsolve Loes {0,8,1000,0.36,g};                          // Benutzt Konstruktor mit ta=0, tb=8, N=1000, alpha=0.36, g

    printf("# 0: Index i \n# 1: t-Wert \n# 2: x-Wert \n"); // Beschreibung der ausgegebenen Groessen

    for(size_t i=0; i  < Loes.get_zeit().size(); ++i){      // for-Schleife zur Terminalausgabe der Loesung
        printf("%3ld %19.15f %19.15f \n",i, Loes.get_zeit()[i], Loes.get_x()[i]);
    }                                                       // Ende for-Schleife der Terminalausgabe
}                                                           // Ende der Hauptfunktion
```

Vorlesung 9

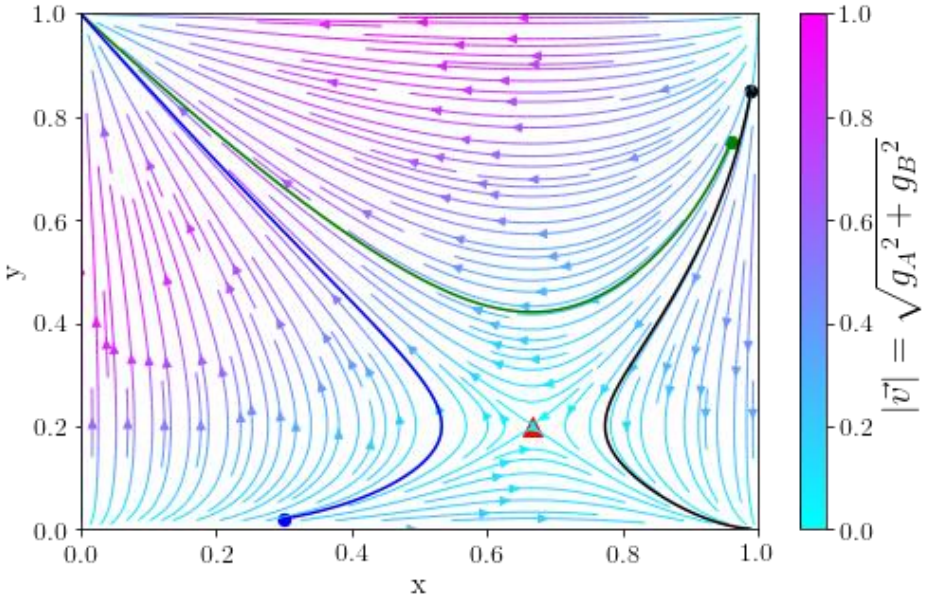
In dieser Vorlesung befassen wir uns zunächst mit dem numerischen Lösen von Systemen gekoppelter Differentialgleichungen und Differentialgleichungen zweiter Ordnung und stellen im darauf folgenden Teil mögliche Programmierprojekte vor, von denen wir einige im Laufe der Vorlesung noch bearbeitet werden. Beim Klicken auf die Überschriften der Projekte gelangen Sie zu einer detaillierteren Beschreibung der einzelnen Projektthemen.

Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung

In der vorigen Vorlesung hatten wir die unterschiedlichen Verfahren zum Lösen von Differentialgleichungen erster Ordnung kennengelernt. Die Bewegungsgleichungen vieler physikalischer Systeme sind jedoch von zweiter Ordnung in der Zeit und in diesem Teilkapitel beschreiben wir die Vorgehensweise wie man solche Differentialgleichungen höherer Ordnung numerisch mittels eines C++ Programmes löst. Um eine Differentialgleichung zweiter Ordnung mittels des Computers lösen zu können, schreibt man die DGL zunächst in ein System von zwei gekoppelten Differentialgleichungen ersten Ordnung um und diese löst man dann mit den Verfahren, die in der vorigen Vorlesung behandelt wurden. In diesem Unterpunkt werden wir uns zunächst mit Systemen von gekoppelten Differentialgleichungen erster Ordnung befassen und dann das numerische Lösen von Differentialgleichungen zweiter Ordnung vorstellen (näheres siehe Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung).

Vorlesung 9

Das numerische Lösen von Differentialgleichungen ist ein mathematisch anspruchsvolles Thema und kann in dieser Vorlesung nicht im Detail erläutert werden. Im ersten Teil dieser Vorlesung sollen die im vorigen Unterpunkt (Differentialgleichungen: Numerische Lösung von Anfangswertproblemen) besprochenen Verfahren auf Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung angewendet werden. Die Bewegungsgleichungen vieler physikalischer Systeme sind von zweiter Ordnung in der Zeit und es soll die Vorgehensweise besprochen werden, wie man solche Differentialgleichungen höherer Ordnung numerisch mittels eines C++ Programmes löst.



Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung

Im vorigen Unterpunkt hatten wir die unterschiedlichen Verfahren zum Lösen von Differentialgleichungen erster Ordnung kennengelernt. Die Bewegungsgleichungen vieler physikalischer Systeme sind jedoch von zweiter Ordnung in der Zeit und dieses Unterkapitel der Vorlesung 9 befasst sich damit, wie man solche Differentialgleichungen höherer Ordnung numerisch mittels eines C++ Programmes löst. Um ein Differentialgleichung zweiter Ordnung mittels des Computers lösen zu können, schreibt man zunächst die DGL in ein System von zwei gekoppelten Differentialgleichungen ersten Ordnung um und diese löst man dann mit den Verfahren, die in der vorigen Vorlesung behandelt wurden. In diesem Unterpunkt werden wir uns zunächst mit Systemen von gekoppelten Differentialgleichungen erster Ordnung befassen und dann das numerische Lösen von Differentialgleichungen zweiter Ordnung vorstellen.

Systeme von gekoppelten Differentialgleichungen

Wir betrachten zunächst das numerische Lösen eines Systems von m-gekoppelten Differentialgleichungen (DGLs) erster Ordnung der Form

$$\begin{aligned}
\dot{y_1}(t) &= \frac{dy_1}{dt} = f_1(t, y_1, y_2, \ldots, y_m) \\
\dot{y_2}(t) &= \frac{dy_2}{dt} = f_2(t, y_1, y_2, \ldots, y_m) \\
\dot{y_3}(t) &= \ldots = \\
\ldots &= \ldots \\
\dot{y}_m(t) &= \frac{dy_m}{dt} = f_m(t, y_1, y_2, \ldots, y_m) \quad ,
\end{aligned}$$

wobei die zeitliche Entwicklung der Vektorfunktion $\vec{y}(t) = (y_1(t), y_2(t), \ldots, y_m(t))$ in den Grenzen $a \leq t \leq b$ gesucht wird. Die m-Funktionen $f_i(t, y_1, y_2, \ldots, y_m)\,,\ i \in [1, 2, \ldots, m]$ bestimmen das System der DGLs und somit das Verhalten der gesuchten Funktion $\vec{y}(t)$. Es wird hierbei vorausgesetzt, dass die Funktionen $f_i(t, y_1, y_2, \ldots, y_m)$ auf einer Teilmenge $\mathcal{D}$ ($\mathbb{R}^{m+1} \supseteq \mathcal{D}$) kontinuierlich definiert sind und das so definierte Anfangswertproblem "well-posed" ist und eine eindeutige Lösung $\vec{y}(t)$ existiert. Bei gegebener Anfangskonfiguration

$$y_1(a) = \alpha_1\,,\ y_2(a) = \alpha_2\,,\ \ldots\ , y_m(a) = \alpha_m$$

ist es dann numerisch möglich das System von gekoppelten DGLs zu lösen.

Systeme von gekoppelten Differentialgleichungen

Wir betrachten zunächst das numerische Lösen eines Systems von m-gekoppelten Differentialgleichungen (DGLs) erster Ordnung der Form

$$\begin{aligned}
\dot{y_1}(t) &= \frac{dy_1}{dt} = f_1(t, y_1, y_2, \ldots, y_m) \\
\dot{y_2}(t) &= \frac{dy_2}{dt} = f_2(t, y_1, y_2, \ldots, y_m) \\
\dot{y_3}(t) &= \ldots = \\
&\ldots = \ldots \\
\dot{y}_m(t) &= \frac{dy_m}{dt} = f_m(t, y_1, y_2, \ldots, y_m) \quad ,
\end{aligned}$$

wobei die zeitliche Entwicklung der Vektorfunktion $\vec{y}(t) = (y_1(t), y_2(t), \ldots, y_m(t))$ in den Grenzen $a \leq t \leq b$ gesucht wird. Die m-Funktionen $f_i(t, y_1, y_2, \ldots, y_m)\,,\ i \in [1, 2, \ldots, m]$ bestimmen das System der DGLs und somit das Verhalten der gesuchten Funktion $\vec{y}(t)$. Es wird hierbei vorausgesetzt, dass die Funktionen $f_i(t, y_1, y_2, \ldots, y_m)$ auf einer Teilmenge $\mathcal{D}$ ($\mathbb{R}^{m+1} \supseteq \mathcal{D}$) kontinuierlich definiert sind und das so definierte Anfangswertproblem "well-posed" ist und eine eindeutige Lösung $\vec{y}(t)$ existiert. Bei gegebener Anfangskonfiguration

$$y_1(a) = \alpha_1\,,\ y_2(a) = \alpha_2\,,\ \ldots\ , y_m(a) = \alpha_m$$

ist es dann numerisch möglich das System von gekoppelten DGLs zu lösen.

Beispiel: Numerische Lösung eines Systems von zwei gekoppelten Differentialgleichungen erster Ordnung

Wir betrachten speziell das folgende System bestehend aus zwei gekoppelten DGLs ($m = 2$):

$$\dot{y_1}(t) = \frac{dy_1}{dt} = 3y_1 + 2y_2 - \left(2t^2 + 1\right) \cdot e^{2t} =: f_1(t, y_1, y_2)$$
$$\dot{y_2}(t) = \frac{dy_2}{dt} = 4y_1 + y_2 + \left(t^2 + 2t - 4\right) \cdot e^{2t} =: f_2(t, y_1, y_2) \quad ,$$

und sind an der numerischen Lösung $\vec{y}(t) = (y_1(t), y_2(t))$ im Zeitintervall $t \in [0, 1]$ interessiert. Die Anfangsbedingungen lauten

$$y_1(0) = \alpha_1 = 1\,,\ y_2(0) = \alpha_2 = 1 \quad .$$

Das Lösen dieses Systems von DGLs ist auf gleichem Wege möglich, wie man einzelne Differentialgleichungen numerisch approximiert.

C++ Programm: Implementierung der DGL bestimmenden Funktionen

und analytische Lösungen zum Vergleich

DGL_2.cpp

```cpp
/* Berechnung der Lösung eines Systems von Differentialgleichung
 * der Form y1'=f1(t,y1,y2) , y2'=f2(t,y1,y2)
 * mittels der Euler Methode und Runge-Kutta Ordnung vier Methode
 * Zeitentwicklung von y1(t), y2(t) der fuer unterschiedliche t-Werte in [a,b]
 *  Ausgabe zum Plotten mittels Python Jupyter Notebook DGL_2.ipynb: "./a.out > DGL_2.dat
 */
#include <stdio.h>              // Standard Input- und Output Bibliothek in C, z.B. printf(...)
#include <cmath>                // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

double f_1(double t, double y_1, double y_2){                  // Deklaration und Definition der Funktion f_1(t,y1,y2)
    double wert;
    wert = 3*y_1 + 2*y_2 - (2*pow(t,2) + 1)*exp(2*t);          // Eigentliche Definition der Funktion
    return wert;                                               // Rueckgabewert der Funktion f_1
}                                                              // Ende der Funktion f_1

double f_2(double t, double y_1, double y_2){                  // Deklaration und Definition der Funktion f_2(t,y1,y2)
    double wert;
    wert = 4*y_1 + y_2 + (pow(t,2) +2*t - 4)*exp(2*t);         // Eigentliche Definition der Funktion
    return wert;                                               // Rueckgabewert der Funktion f_2
}                                                              // Ende der Funktion f_2

double y_1_analytisch(double t){                               // Analytische Loesung y_1(t)
    double wert;                                               // bei gegebenem Anfangswert y_1(a)=1, y_2(a)=1
    wert = exp(5*t)/3 - exp(-t)/3 + exp(2*t);                  // Eigentliche Definition der analytische Loesung
    return wert;                                               // Rueckgabewert
}                                                              // Ende der Definition

double y_2_analytisch(double t){                               // Analytische Loesung y_2(t)
    double wert;                                               // bei gegebenem Anfangswert y_1(a)=1, y_2(a)=1
    wert = exp(5*t)/3 + 2*exp(-t)/3 + pow(t,2)*exp(2*t);       // Eigentliche Definition der analytische Loesung
    return wert;                                               // Rueckgabewert
}                                                              // Ende der Definitiom
```

```c
int main(){                                                    // Hauptfunktion
    double a = 0;                                              // Untergrenze des Zeit-Intervalls [a,b] in dem die Loesung berechnet werden soll
    double b = 1;                                              // Obergrenze des Intervalls [a,b]
    int N = 100;                                               // Anzahl der Punkte in die das t-Intervall aufgeteilt wird
    double h = (b - a)/N;                                      // Abstand dt zwischen den aequidistanten Punkten des t-Intervalls (h=dt)
    double alpha_1 = 1;                                        // 1.Anfangswert bei t=a: y_1(a)=alpha_1
    double alpha_2 = 1;                                        // 2.Anfangswert bei t=a: y_2(a)=alpha_2
    double t;                                                  // Aktueller Zeitwert
    double y_Euler_1 = alpha_1;                                // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_1
    double y_RungeK_4_1 = alpha_1;                             // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double k1_1,k2_1,k3_1,k4_1;                                // Deklaration der vier Runge-Kutta Parameter fuer y_1
    double y_Euler_2 = alpha_2;                                // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_2
    double y_RungeK_4_2 = alpha_2;                             // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double k1_2,k2_2,k3_2,k4_2;                                // Deklaration der vier Runge-Kutta Parameter fuer y_2
    double tmp;                                                // Variable zum Zwischenspeichern von Ergebnissen

    printf("# 0: Index i \n# 1: t-Wert \n# 2: Euler Methode y1 \n# 3: Euler Methode y2 \n");   // Beschreibung der ausgegebenen Groessen
    printf("# 4: Runge-Kutta Ordnung vier y1 \n# 5: Runge-Kutta Ordnung vier y2 \n");          // Beschreibung der ausgegebenen Groessen
    printf("# 6: Analytische Loesung y1 \n# 7: Analytische Loesung y2 \n");                   // Beschreibung der ausgegebenen Groessen
    printf("%3d %19.15f %19.15f %19.15f %19.15f ",0, t, y_Euler_1, y_Euler_2, y_RungeK_4_1);   // Ausgaben der t=a Werte
    printf(" %19.15f %19.15f %19.15f \n", y_RungeK_4_2, y_1_analytisch(t), y_2_analytisch(t)); // Ausgaben der t=a Werte

    for(int i=0; i <= N; ++i){                                 // for-Schleife ueber die einzelnen Punkte des t-Intervalls
        t = a + i*h;                                           // Zeit-Parameter wird um h erhoeht
        printf("%3d %19.15f %19.15f %19.15f %19.15f ",i, t, y_Euler_1, y_Euler_2, y_RungeK_4_1);   // Ausgaben der Loesungen
        printf(" %19.15f %19.15f %19.15f \n", y_RungeK_4_2, y_1_analytisch(t), y_2_analytisch(t)); // Ausgaben der Loesungen

        tmp = y_Euler_1 + h*f_1(t,y_Euler_1,y_Euler_2);        // Euler Methode
        y_Euler_2 = y_Euler_2 + h*f_2(t,y_Euler_1,y_Euler_2);  // y_2 Euler Methode
        y_Euler_1 = tmp;                                       // y_1 Euler Methode

        k1_1 = h*f_1(t,y_RungeK_4_1,y_RungeK_4_2);             // Runge-Kutta Parameter k1 fuer y_1
        k1_2 = h*f_2(t,y_RungeK_4_1,y_RungeK_4_2);             // Runge-Kutta Parameter k1 fuer y_2
        k2_1 = h*f_1(t+h/2,y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2);   // Runge-Kutta Parameter k2 fuer y_1
        k2_2 = h*f_2(t+h/2,y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2);   // Runge-Kutta Parameter k2 fuer y_2
        k3_1 = h*f_1(t+h/2,y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2);   // Runge-Kutta Parameter k3 fuer y_1
        k3_2 = h*f_2(t+h/2,y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2);   // Runge-Kutta Parameter k3 fuer y_2
        k4_1 = h*f_1(t+h,y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2);         // Runge-Kutta Parameter k4 fuer y_1
        k4_2 = h*f_2(t+h,y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2);         // Runge-Kutta Parameter k4 fuer y_2
        y_RungeK_4_1 = y_RungeK_4_1 + (k1_1 + 2*k2_1 + 2*k3_1 + k4_1)/6;   // Runge-Kutta Ordnung vier Methode fuer y_1
        y_RungeK_4_2 = y_RungeK_4_2 + (k1_2 + 2*k2_2 + 2*k3_2 + k4_2)/6;   // Runge-Kutta Ordnung vier Methode fuer y_2
    }                                                          // Ende for-Schleife ueber die einzelnen Punkte des t-Intervalls
}                                                              // Ende der Hauptfunktion
```

main()-Programm Zur Lösung des Systems bestehend aus zwei DGLs erster Ordnung

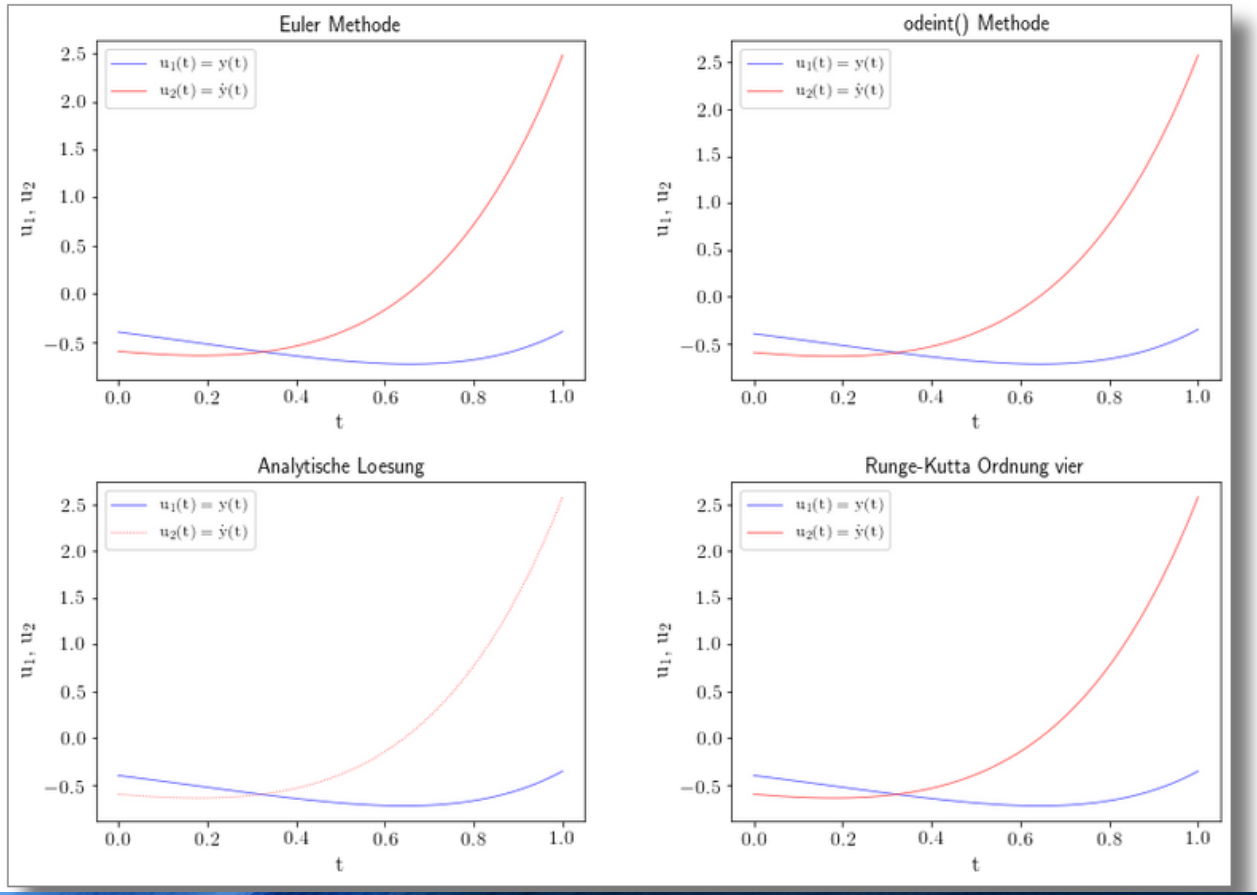
```cpp
int main(){
    double a = 0;
    double b = 1;
    int N = 100;
    double h = (b - a)/N;
    double alpha_1 = 1;
    double alpha_2 = 1;
    double t;
    double y_Euler_1 = alpha_
    double y_RungeK_4_1 = alp
    double k1_1,k2_1,k3_1,k4_
    double y_Euler_2 = alpha_
    double y_RungeK_4_2 = alp
    double k1_2,k2_2,k3_2,k4_
    double tmp;

    printf("# 0: Index i \n#
    printf("# 4: Runge-Kutta
    printf("# 6: Analytische
    printf("%3d %19.15f %19.1
    printf(" %19.15f %19.15f

    for(int i=0; i <= N; ++i){                                  // for-Schleife ueber die einzelnen Punkte des t-Intervalls
        t = a + i*h;                                            // Zeit-Parameter wird um h erhoeht
        printf("%3d %19.15f %19.15f %19.15f %19.15f ",i, t, y_Euler_1, y_Euler_2, y_RungeK_4_1);   // Ausgaben der Loesungen
        printf(" %19.15f %19.15f %19.15f \n", y_RungeK_4_2, y_1_analytisch(t), y_2_analytisch(t)); // Ausgaben der Loesungen

        tmp = y_Euler_1 + h*f
        y_Euler_2 = y_Euler_2
        y_Euler_1 = tmp;

        k1_1 = h*f_1(t,y_Rung
        k1_2 = h*f_2(t,y_Rung
        k2_1 = h*f_1(t+h/2,y_
        k2_2 = h*f_2(t+h/2,y_
        k3_1 = h*f_1(t+h/2,y_
        k3_2 = h*f_2(t+h/2,y_
        k4_1 = h*f_1(t+h,y_Ru
        k4_2 = h*f_2(t+h,y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2);  // Runge-Kutta Parameter k4 fuer y_2
        y_RungeK_4_1 = y_RungeK_4_1 + (k1_1 + 2*k2_1 + 2*k3_1 + k4_1)/6;     // Runge-Kutta Ordnung vier Methode fuer y_1
        y_RungeK_4_2 = y_RungeK_4_2 + (k1_2 + 2*k2_2 + 2*k3_2 + k4_2)/6;     // Runge-Kutta Ordnung vier Methode fuer y_2
    }                                                           // Ende for-Schleife ueber die einzelnen Punkte des t-Intervalls
}                                                               // Ende der Hauptfunktion
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V9$ g++ DGL_2.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V9$ ./a.out
# 0: Index i
# 1: t-Wert
# 2: Euler Methode y1
# 3: Euler Methode y2
# 4: Runge-Kutta Ordnung vier y1
# 5: Runge-Kutta Ordnung vier y2
# 6: Analytische Loesung y1
# 7: Analytische Loesung y2
  0   0.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000
  0   0.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000   1.000000000000000
  1   0.010000000000000   1.040000000000000   1.010000000000000   1.040608426315013   1.010558940181763   1.040608427569041   1.010558941425456
  2   0.020000000000000   1.081195946197052   1.021097006868275   1.082468186487541   1.022272409932680   1.082468189115352   1.022272412539396
  3   0.030000000000000   1.123637330492212   1.033343871369918   1.125632778808522   1.035197418735856   1.125632782938618   1.035197422833657
  4   0.040000000000000   1.167375849311085   1.046795999898338   1.170158168240702   1.049393799070299   1.170158174010907   1.049393804796552
  5   0.050000000000000   1.212465508925469   1.061511473409989   1.216102907913251   1.064924341689591   1.216102915471323   1.064924349191579
  6   0.060000000000000   1.258962735934773   1.077551171969109   1.263528266739016   1.081854937478282   1.263528276243294   1.081854946913853
  7   0.070000000000000   1.306926493163091   1.094978905171568   1.312498363466237   1.100254726211197   1.312498375086330   1.100254737749452
  8   0.080000000000000   1.356418401240555   1.113861548730063   1.363080307492500   1.120196252557172   1.363080321410022   1.120196266379195
  9   0.090000000000000   1.407502866150967   1.134269187516644   1.415344346785449   1.141755629685859   1.415344363194790   1.141755645985495
 10   0.100000000000000   1.460247213041785   1.156275265373008   1.469364023272402   1.165012710854272   1.469364042381559   1.165012729838951
 11   0.110000000000000   1.514721826607265   1.179956742014139   1.525216336079555   1.190051269368648   1.525216358111003   1.190051291260258
```

```
 89   0.890000000000000  30.930880042824768  30.063568069606163  34.335278279597091  33.513116889352368  34.335285835073890  33.513124438401135
 90   0.900000000000000  32.306838455932400  31.516766532214152  35.919826655719859  35.176963303871311  35.919834678006687  35.176971319508823
 91   0.910000000000000  33.747878176687038  33.040117636018252  37.581811296515667  36.923393059891680  37.581819813513697  36.923401570012345
 92   0.920000000000000  35.257179970562206  34.636996585061937  39.325128086735063  38.756498858038242  39.325137127976433  38.756507892167264
 93   0.930000000000000  36.838082119087382  36.310942423138201  41.153871130268442  40.680578010174486  41.153880727010005  40.680587599561534
 94   0.940000000000000  38.494088264736298  38.065666064560588  43.072342882456958  42.700142759460462  43.072353067779481  42.700152937178956
 95   0.950000000000000  40.228875647423045  39.905058721679517  45.085064801277419  44.819931124470799  45.085075610190323  44.819941925522599
 96   0.960000000000000  42.046303752173394  41.833200748846942  47.196788543991907  47.044918294100583  47.196800013543999  47.044929755526724
 97   0.970000000000000  43.950423388516661  43.854370923514082  49.412507737215932  49.380328601353476  49.412519906612403  49.380340762351139
 98   0.980000000000000  45.945486223170619  45.973056186177487  51.737470349791899  51.831648105542975  51.737483260518005  51.831661007589204
 99   0.990000000000000  48.035954788670189  48.193961861970656  54.177191699361046  54.404637813947794  54.177205395312626  54.404651500929958
100   1.000000000000000  50.226512991722970  50.522022387834483  56.737468125110773  57.105347575549835  56.737482652732375  57.105362093903814
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V9$
```

Visualisierung und numerische Lösung mittels Python

Beim Klicken auf das untere rechte Bild gelangen Sie zu dem Jupyter Notebook DGL_2.ipynb in dem eine Visualisierung der Ergebnisse des oberen C++ Programms programmiert ist. Die untere Abbildung auf der linken Seite zeigt z.B. die Visualisierung der Daten von DGL_2.cpp. Zusätzlich wird in dem Notebook auch die numerische Lösung direkt in Python generiert (Methode "integrate.odeint()" im Python-Modul "scipy") und mit den simulierten Daten der unterschiedlichen Verfahren verglichen.



Einführung in die Programmierung für Studierende der Physik

(Introduction to Programming for Physicists)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main

(Sommersemester 2025)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 27.01.2025

Numerisches Lösen von Differentialgleichungen

Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung

Im Jupyter Notebook DGL_1_2025.ipynb haben wir einige in Python implementierte Lösungsmethoden für Differentialgleichungen erster Ordnung kennengelernt. In diesem Notebook werden wir uns zunächst mit Systemen von gekoppelten Differentialgleichungen erster Ordnung befassen und dann das numerische Lösen von Differentialgleichungen zweiter Ordnung vorstellen.

Systeme von gekoppelten Differentialgleichungen

Wir betrachten zunächst das numerische Lösen eines Systems von m-gekoppelten Differentialgleichungen (DGLs) erster Ordnung der Form

$$\dot{y_1}(t) = \frac{dy_1}{dt} = f_1(t, y_1, y_2, \ldots, y_m)$$
$$\dot{y_2}(t) = \frac{dy_2}{dt} = f_2(t, y_1, y_2, \ldots, y_m)$$
$$\dot{y_3}(t) = \ldots =$$

Replikatordynamik (2xM)-Spiele

Wir beschränken uns zunächst auf symmetrische (2xM)-Spiele , d.h. zwei Personen - M Strategien Spiele. Da es sich um symmetrische Spiele handelt, sind alle Spieler gleichberechtigt und man kann von einer homogenen Population ausgehen.
Die Differentialgleichung der Replikatordynamik beschreibt wie sich die einzelnen Populationsanteil der zur Zeit t gewählten Strategien $x_j(t)$, j=1,2,...M im Laufe der Zeit entwickeln.

$$\dot{x}_j(t) := \frac{dx_j(t)}{dt} = x_j(t) \cdot \left[\sum_{k=1}^{M} \$_{jk} \cdot x_k(t) - \sum_{l=1}^{M} \sum_{k=1}^{M} \$_{kl} \cdot x_k(t) \cdot x_l(t) \right]$$

Wobei die Parameter $\$_{kl}$ die einzelnen Einträge in der Auszahlungsmatrix des 1.Spielers darstellen

$$\hat{\$} = \hat{\$}^1 = \begin{pmatrix} \$_{11} & \$_{12} & \$_{13} & \dots & \$_{1M} \\ \$_{21} & \$_{22} & \$_{23} & \dots & \$_{2M} \\ \$_{31} & \$_{32} & \$_{33} & \dots & \$_{3M} \\ \dots & \dots & \dots & \dots & \dots \\ \$_{M1} & \$_{M2} & \$_{M3} & \dots & \$_{MM} \end{pmatrix}$$

Fitness der Strategie j

Durchschnitt-licher Erfolg der j-ten Strategie

Durschnittliche Fitness (Auszahlung) der gesamten Population

Replikatordynamik

(für symmetrische (2x3)-Spiele)

Wir beschränken uns nun auf symmetrische (2x3)-Spiele , d.h. zwei Personen - 3 Strategien Spiele (M=3). Die Differentialgleichung der Replikatordynamik vereinfacht sich unter dieser Annahme wie folgt:

$$\frac{dx_j(t)}{dt} = x_j(t) \cdot \left[\sum_{k=1}^{3} \$_{jk} \cdot x_k(t) - \sum_{l=1}^{3} \sum_{k=1}^{3} \$_{kl} \cdot x_k(t) \cdot x_l(t) \right]$$

$$\frac{dx_j}{dt} = x_j \cdot \left[\begin{array}{l} \$_{j1} \cdot x_1 + \$_{j2} \cdot x_2 + \$_{j3} \cdot x_3 - \\ -\left(\begin{array}{l} \$_{11} \cdot x_1 \cdot x_1 + \$_{12} \cdot x_1 \cdot x_2 + \$_{13} \cdot x_1 \cdot x_3 + \\ +\$_{21} \cdot x_2 \cdot x_1 + \$_{22} \cdot x_2 \cdot x_2 + \$_{23} \cdot x_2 \cdot x_3 + \\ +\$_{31} \cdot x_3 \cdot x_1 + \$_{32} \cdot x_3 \cdot x_2 + \$_{33} \cdot x_3 \cdot x_3 \end{array} \right) \end{array} \right]$$

$j = 1,2,3$

$\overline{\$}$

Replikatordynamik

(für symmetrische (2x3)-Spiele)

Man erhält ein System von drei gekoppelten Differentialgleichungen:

$$\frac{dx_1}{dt} = x_1 \cdot \left[\$_{11} \cdot x_1 + \$_{12} \cdot x_2 + \$_{13} \cdot x_3 - \bar{\$}\right]$$

$$\frac{dx_2}{dt} = x_2 \cdot \left[\$_{21} \cdot x_1 + \$_{22} \cdot x_2 + \$_{23} \cdot x_3 - \bar{\$}\right]$$

$$\frac{dx_3}{dt} = x_3 \cdot \left[\$_{31} \cdot x_1 + \$_{32} \cdot x_2 + \$_{33} \cdot x_3 - \bar{\$}\right]$$

Das System von Differentialgleichungen lässt sich bei gegebener Auszahlungsmatrix $\hat{\$}$ und Anfangsbedingung $(x_1(0), x_2(0), x_3(0))$ meist nur nummerisch (auf dem Computer) lösen. Die Lösungen bestehen dann aus den drei (zeitlich abhängigen) Populationsanteilen $(x_1(t), x_2(t), x_3(t))$.

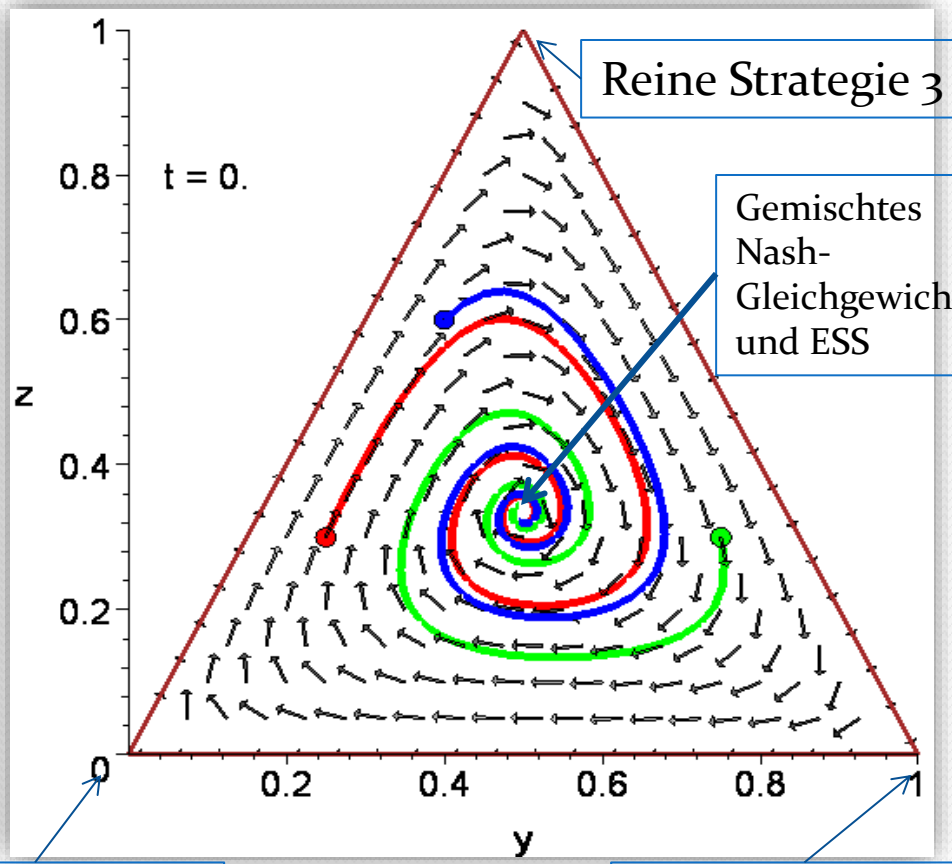
Replikatordynamik

(für symmetrische (2x3)-Spiele, **Beispiel 1**)

Wir betrachten im Folgenden ein Beispiel eines (2x3)-Spiels mit der rechts angegebenen Auszahlungsstruktur:

	Strategie 1	Strategie 2	Strategie 3
Strategie 1	(0, 0)	(2, -1)	(-1 , 2)
Strategie 2	(-1 , 2)	(0 , 0)	(2 , -1)
Strategie 3	(2 , -1)	(-1 , 2)	(0 , 0)

Die rechte Abbildung zeigt die zeitliche Entwicklung der relativen Populationsanteile der gewählten Strategien für drei mögliche Anfangsbedingungen. Die einzige evolutionär stabile Strategie dieses Beispiels befindet sich beim gemischten Nash-Gleichgewicht $\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$
Die einzelnen Pfeile im Dreieck veranschaulichen den durch die Spielmatrix bestimmten Strategien-„Richtungswind“, dem die Population zeitlich folgen wird.



Zur Visualisierung der evolutionären Entwicklung benutzt man of die sogen. barycentric coordinates:

$$y := x_2 + \frac{x_3}{2}$$

$$z := x_3$$

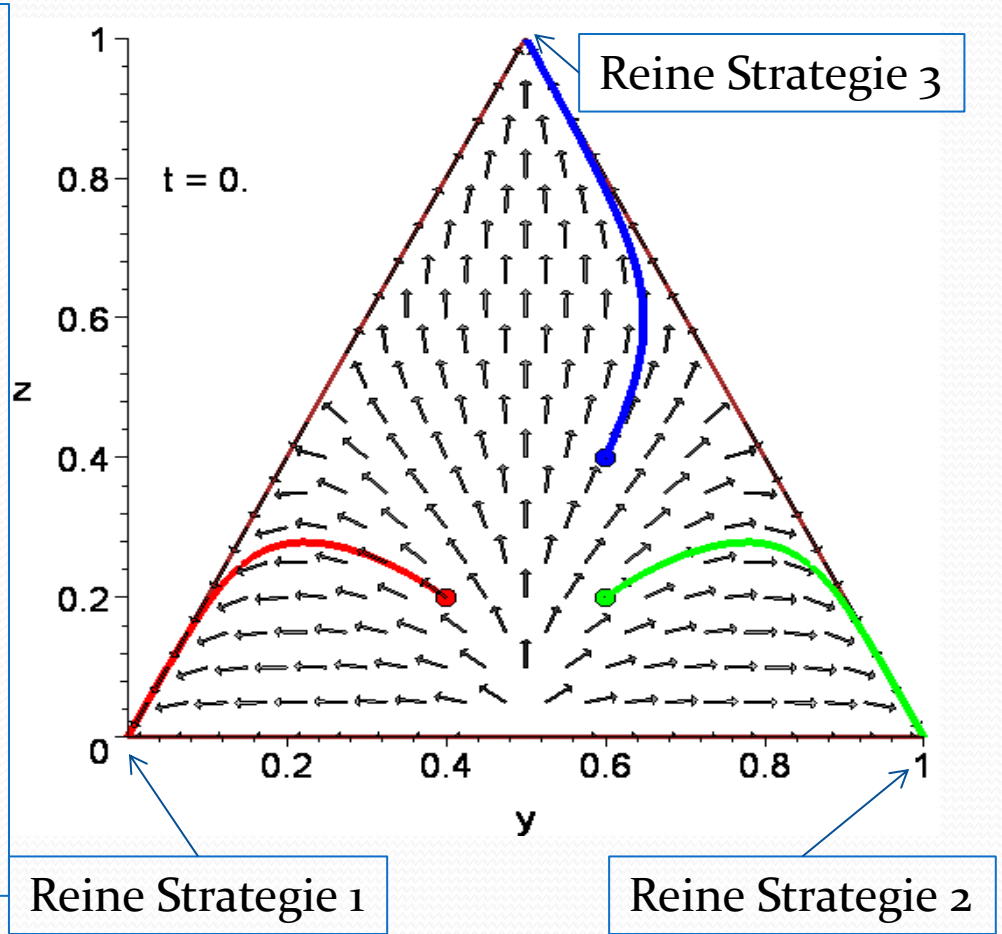
Replikatordynamik

(für symmetrische (2x3)-Spiele, **Beispiel 2**)

Wir betrachten im Folgenden ein Beispiel eines (2x3)-Spiels mit der rechts angegebenen Auszahlungsstruktur:

	Strategie 1	Strategie 2	Strategie 3
Strategie 1	(0, 0)	(-3, -3)	(-1 , -1)
Strategie 2	(-3 , -3)	(0 , 0)	(-1 , -1)
Strategie 3	(-1 , -1)	(-1 , -1)	(0 , 0)

Die rechte Abbildung zeigt die zeitliche Entwicklung der relativen Populationsanteile der gewählten Strategien für drei mögliche Anfangsbedingungen. Das Spiel besitzt drei Nash-Gleichgewichte in reinen Strategien, die ebenfalls evolutionär stabile Strategien darstellen. Welche der drei ESS die Population realisiert hängt von dem Anfangswert der Populationsanteile ab. Die zeitliche Entwicklung folgt wieder dem Strategien-„Richtungswind“ der zugrundeliegenden Auszahlungsmatrix.

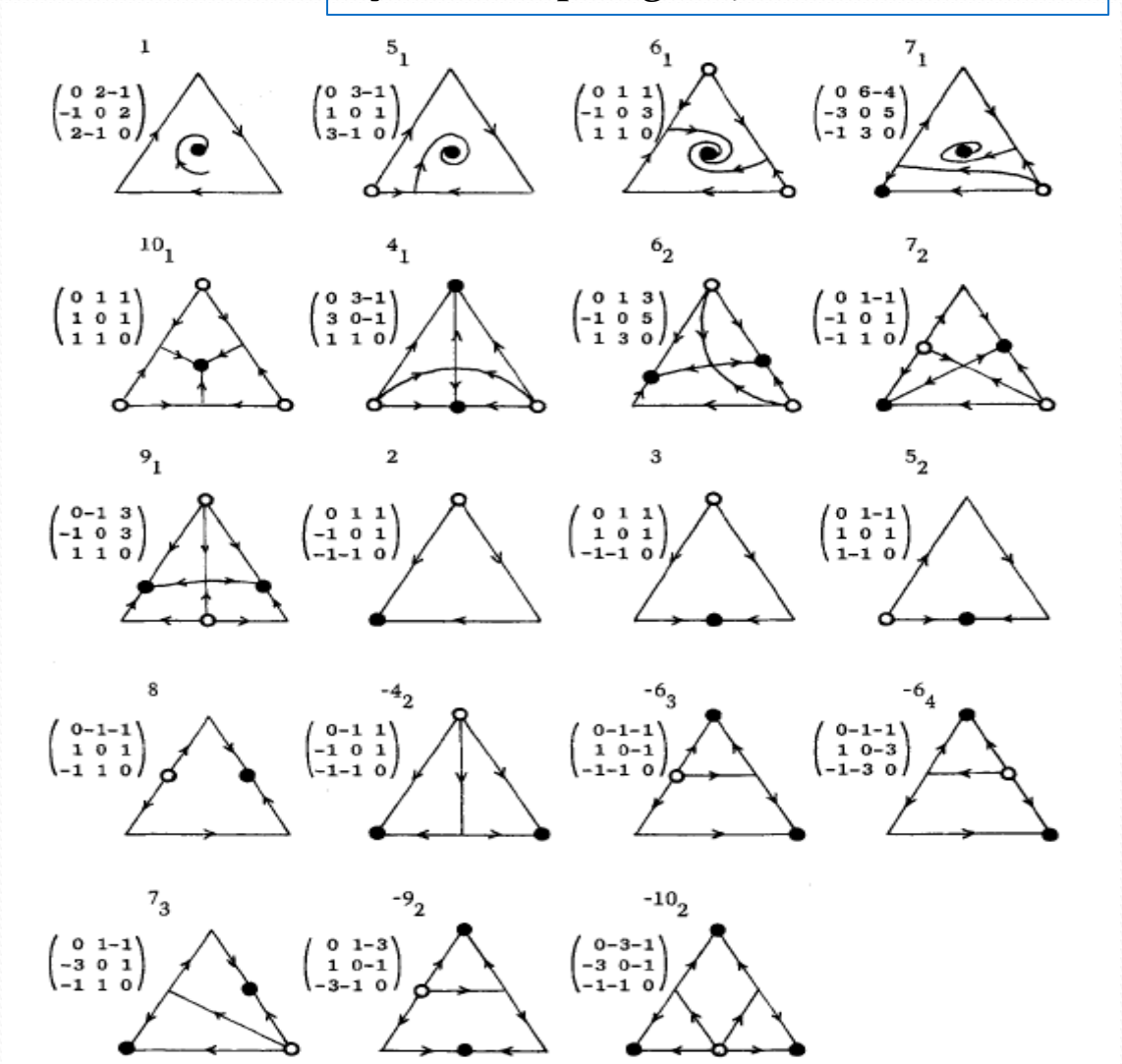


Replikatordynamik

(Klassifizierung symmetrische (2x3)-Spiele)

E. C. Zeeman, *POPULATION DYNAMICS FROM GAME THEORY, In:* Global Theory of Dynamical Systems, Springer 1980

E. C. Zeeman zeigt in seinem im Jahre 1980 veröffentlichten Artikel, dass man evolutionäre, symmetrische (2x3)-Spiele in 19 Klassen einteilen kann. Die Abbildung rechts zeigt das evolutionäre Verhalten dieser 19 Spieltypen. Die ausgefüllten schwarzen Punkte markieren die evolutionär stabilen Strategien der jeweiligen Spiele. Es gibt Spielklassen, die besitzen lediglich eine ESS und Klassen die sogar drei ESS besitzen.



Jupyter Notebook
Evolutionspiel4.ipynb

Physik der sozio-ökonomischen Systeme mit dem Computer

(Physics of Socio-Economic Systems with the Computer)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main

(Wintersemester 2025/26)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 22.08.2025

Erster Vorlesungsteil:

Die 19 Klassen der evolutionären symmetrischen (2×3)-Spiele

Einführung

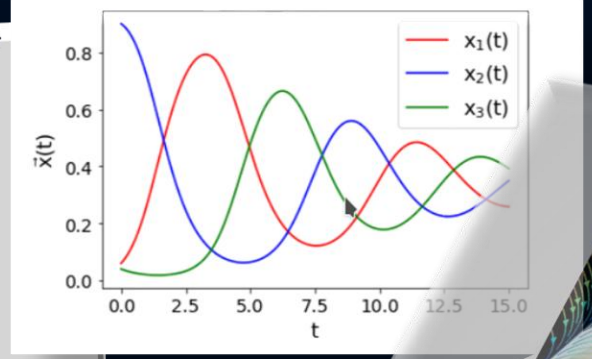
In diesem Unterkapitel werden die evolutionären symmetrischen (2×3)-Spiele analysiert. Symmetrische $(2 \times m)$-Spiele werden durch die folgende Differenzialgleichung beschrieben:

$$\frac{d\vec{x}}{dt} = \vec{x}\left(\hat{\$}\vec{x}\right) - \left(\left(\hat{\$}\vec{x}\right)^T \vec{x}\right)\vec{x}$$

Aufgrund der Symmetrie der Auszahlungsmatrix $\hat{\$} = \hat{\$}^A = \left(\hat{\$}^B\right)^T$ kann die zeitliche Entwicklung solcher Spiele durch nur einen Populationsvektor ($\vec{x}(t) = (x_1(t), x_2(t), \ldots, x_m(t))$ beschrieben werden.

Wir beschränken uns im Folgenden auf den Fall $m = 3$ und setzen die Auszahlungsmatrix wie folgt an:

$$\hat{\$} = \begin{pmatrix} \$_{11} & \$_{12} & \$_{13} \\ \$_{21} & \$_{22} & \$_{23} \\ \$_{31} & \$_{32} & \$_{33} \end{pmatrix}$$



Auf der Internetseite der Vorlesung

- Folien der 5.Vorlesung
- Vorlesungsaufzeichnung der 5.Vorlesung: WS 2022/23 bzw. WS 2021/22
- View Jupyter Notebook: Die 19 Klassen der evolutionären symmetrischen (2x3)-Spiele
- Download Jupyter Notebook: Die 19 Klassen der evolutionären symmetrischen (2x3)-Spiele
- Download Python Programm: Evolutionäre Spieltheorie symmetrischer (2x3)-Spiele (2x3-Spiel_ng.py)
- View Jupyter Notebook: Die Räuber-Beute Gleichung im Kontext der evolutionären Spieltheorie
- Download Jupyter Notebook: Die Räuber-Beute Gleichung im Kontext der evolutionären Spieltheorie
- Maple Worksheet: Äquivalenz der Räuber-Gleichung für N-Populationen mit der Replikatorgleichung der evolutionären Spieltheorie für (N+1)-Strategien
- Download Jupyter Notebook VPSOC_DGL_1.ipynb
- View Jupyter Notebook VPSOC_DGL_1.html
- Download C++ Programm evol1.cpp

Python Programm 2x3-Spiel_ng.py

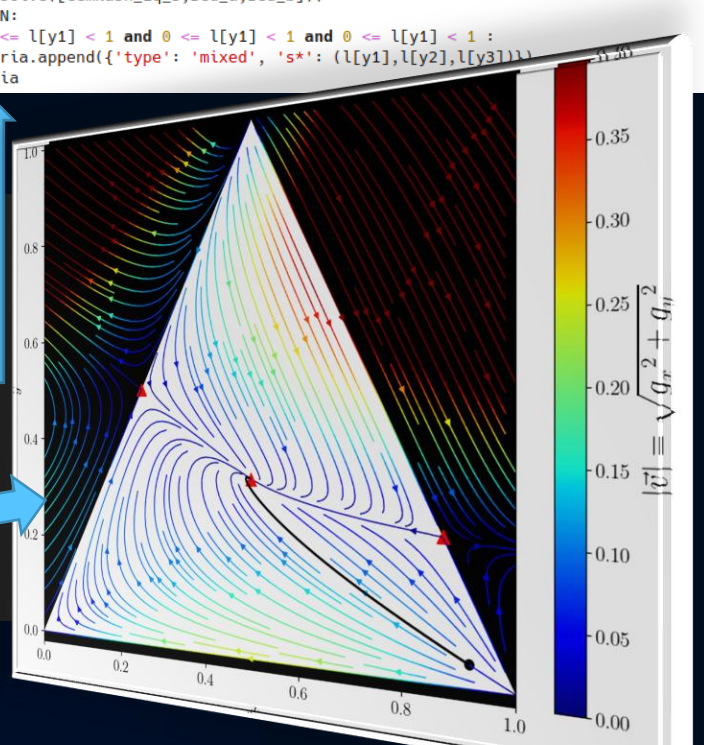
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
import matplotlib.colors as colors
from scipy.integrate import solve_ivp
from sympy import *

# baryzentrisches Dreiecks-Koordinatensystem
def xy(vx):
    return np.array([vx[1]+vx[2]/2,vx[2]])

# Definition der Funktionen g_x und g_y
def g(xy, D):
    m = 3
    x = np.array([1-xy[0]-xy[1]/2,xy[0]-xy[1]/2,xy[1]])
    dx_dt = []
    for i in range(m):
        dx_dt.append(sum(D[i,j]*x[i]*x[j] for j in range(m)) - sum(sum(D[k,j]*x[k]*x[j] for j in range(m)) for k in range(m))*x[i])
    return [dx_dt[1]+dx_dt[2]/2,dx_dt[2]]

# Vektorisiertes DGL-System mit numpy
def DGLsys(t, x, D):
    x = np.asarray(x)
    Dx = D @ x
    u = np.dot(x, Dx)
    return x * (Dx - u)

# Alternative Definition des DGL-Systems
#def DGLsys(t,x,D):
#    m=3
#    dx_dt = []
#    for i in range(m):
#        dx_dt.append(sum(D[i,j]*x[i]*x[j] for j in range(m)) - sum(sum(D[k,j]*x[k]*x[j] for j in range(m)) for k in range(m))*x[i])
#    return dx_dt

# Loesen der DGL und plotten der Populationsentwicklung
def solve_and_plot(D, x_init, t_end=15, n_points=1000):
    # Groessenfestlegung der Labels usw. im Bild
    rcParams.update({
        'text.usetex'       : True,
        'figure.figsize'    : [8,6],
        'axes.titlesize' : 14,
        'axes.labelsize' : 16,
        'xtick.labelsize' : 14 ,
        'ytick.labelsize' : 14
    })

    # Weitere Festlegungen
    fehler = 10**(-13)
    t_eval = np.linspace(0, t_end, n_points)

    # Loesung der DGL fuer eine Anfangspopulation x_init
    Loes = solve_ivp(DGLsys, [0, t_end], x_init, args=(D, ), t_eval=t_eval, rtol=fehler, atol=fehler)

    # Fuer die Darstellung des Feldliniendiagramms streamplot
    Y, X = np.mgrid[0:1:100j, 0:1:100j]
    gXY = g([X,Y],D)
    # Die Farbe wird die Geschwindigkeit der Aenderung des Populationsvektors anzeigen
    colorspeed = np.sqrt(gXY[0]**2 + gXY[1]**2)

    # Plotten des Bildes
    fig, ax = plt.subplots(figsize=(10, 8))
    ax.set_xlabel(r"$x$")
    ax.set_ylabel(r"$y$")
    strm = ax.streamplot(X, Y, gXY[0], gXY[1], linewidth=1,density=[2, 2],norm=colors.Normalize(vmin=0, vmax=0.4), color=colorspeed, cmap=plt.cm.jet)
```

```python
# Funktion zur Berechnung der Nashgleichgewichte
def find_nash_equilibria(D):
    equilibria = []
    # Berechnung der reinen Nash-Gleichgewichte
    for i in range(D.shape[0]):
        for j in range(D.shape[0]):
            if D[i, j] == max(D[:, j]) and D[j, i] == max(D[:, i]):
                equilibria.append({'type': 'pure', 's': (i+1,j+1)})
    #Berechnung der gemischten Nash-Gleichgewichte (Interior-gemischte und Boundary-gemischte)
    Loes_GN = []
    x1,x2,x3,y1,y2,y3 = symbols('x_1,x_2,x_3,y_1,y_2,y_3')
    xs = Matrix([x1,x2,x3])
    ys = Matrix([y1,y2,y3])
    Dollar_A = transpose(xs)*D*ys
    Dollar_As = Dollar_A.subs(x3,1-x1-x2).subs(y3,1-y1-y2)[0]
    Dollar_As_1 = Dollar_A.subs(x1,0).subs(x3,1-x2).subs(y3,1-y1-y2)[0]
    Dollar_As_2 = Dollar_A.subs(x2,0).subs(x3,1-x1).subs(y3,1-y1-y2)[0]
    Dollar_As_3 = Dollar_A.subs(x3,0).subs(x2,1-x1).subs(y3,1-y1-y2)[0]
    GemNash_Eq1 = Eq(Dollar_As.diff(x1),0)
    GemNash_Eq2 = Eq(Dollar_As.diff(x2),0)
    GemNash_Eq_1 = Eq(Dollar_As_1.diff(x2),0)
    GemNash_Eq_2 = Eq(Dollar_As_2.diff(x1),0)
    GemNash_Eq_3 = Eq(Dollar_As_3.diff(x1),0)
    Bed=Eq(1,y1+y2+y3)
    Loes_GN.append(solve([GemNash_Eq1,GemNash_Eq2,Bed]))
    Bed_a=Eq(0,y1)
    Bed_b=Eq(1,y2+y3)
    Loes_GN.append(solve([GemNash_Eq_1,Bed_a,Bed_b]))
    Bed_a=Eq(0,y2)
    Bed_b=Eq(1,y1+y3)
    Loes_GN.append(solve([GemNash_Eq_2,Bed_a,Bed_b]))
    Bed_a=Eq(0,y3)
    Bed_b=Eq(1,y1+y2)
    Loes_GN.append(solve([GemNash_Eq_3,Bed_a,Bed_b]))
    for l in Loes_GN:
        if l and 0 <= l[y1] < 1 and 0 <= l[y1] < 1 and 0 <= l[y1] < 1 :
            equilibria.append({'type': 'mixed', 's*': (l[y1],l[y2],l[y3])})
    return equilibria
```

Modularisierung des Programms mittels unterschiedlicher Funktionen

Berechnung der Nash-Gleichgewichte

Automatische Kennzeichnung der Nash-Gleichgewichte im Bild

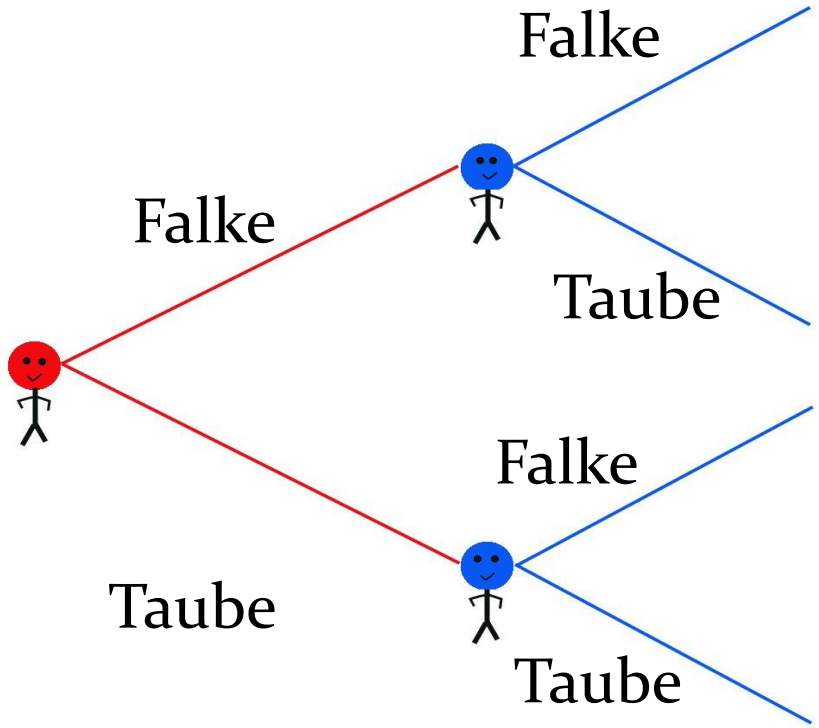


Anwendungsfelder Spieltheorie

- Anwendungsfelder in den Wirtschafts- Sozialwissenschaften und Biologie
  - Experimentelle Ökonomie
  - Die Finanzkrise als Falke-Taube Spiel
  - Die Entstehung einer dritten Strategie im Elfmeter-Spiel (Nesken Effekt)
  - Evolutionäre Entwicklung einer Eidechsen Population als symmetrisches (2x3)-Spiel
  - Das Räuber-Beute Spiel und die Lotka-Volterra-Gleichung
  - Die Klimakrise als Populationsdilemma

Das Falke-Taube Spiel

	Falke	Taube
Falke	$((p_h-d)/2 , (p_h-d)/2)$	$(p_h , 0)$
Taube	$(0 , p_h)$	$(p_m/2, p_m/2)$



Das Falke-Taube-Spiel modelliert ursprünglich den Wettkampf um eine Ressource (z.B. Nistplatz). Das Spiel wird jedoch oft auch auf andere Systeme angewendet, wobei die Taube-Strategie eine friedliche Verhaltensweise symbolisiert und die Falke-Strategie ein aggressives Verhalten.
Im folgenden Artikel wird das Falke-Taube-Spiel auf den Immobilien-Investmentmarkt angewendet (Spieler-Population: Investmentbanker).

Parameter setting	Risk of destabilisation	d	p_h	p_m
P1	LOW	6	5	3
P2	MEDIUM	10	5	3
P3	HIGH	20	5	3

TABLE II: Parameters of the three different sets of the underlying payoff matrix used to model the investment market of the Hawk-Dove game.

TABLE I: Payoff matrix for investment bankers A and B within the Hawk-Dove game. The parameters are defined as follows: p_h: high selling premium, d: disutility resulting from fighting and p_m: moderate selling premium.

Physica A 389 (2010) 5084–5102

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Doves and hawks in economics revisited: An evolutionary quantum game theory based analysis of financial crises

Matthias Hanauske [a,*], Jennifer Kunz [b], Steffen Bernius [a], Wolfgang König [c]

[a] Institute of Information Systems, Goethe-University, Grüneburgplatz 1, 60323 Frankfurt/Main, Germany

[b] Chair of Controlling & Auditing, Goethe-University, Grüneburgplatz 1, 60323 Frankfurt/Main, Germany

[c] House of Finance, Goethe-University, Grüneburgplatz 1, 60323 Frankfurt/Main, Germany

ARTICLE INFO

Article history:
Received 14 April 2009
Received in revised form 22 April 2010
Available online 15 June 2010

Keywords:
Evolutionary game theory
Quantum game theory
Hawk–dove game
Financial crisis

ABSTRACT

The last financial and economic crisis demonstrated the dysfunctional long-term effects of aggressive behaviour in financial markets. Yet, evolutionary game theory predicts that under the condition of strategic dependence a certain degree of aggressive behaviour remains within a given population of agents. However, as a consequence of the financial crisis, it would be desirable to change the "rules of the game" in a way that prevents the occurrence of any aggressive behaviour and thereby also the danger of market crashes. The paper picks up this aspect. Through the extension of the well-known hawk–dove game by a quantum approach, we can show that dependent on entanglement, evolutionary stable strategies also can emerge, which are not predicted by the classical evolutionary game theory and where the total economic population uses a non-aggressive quantum strategy.

Wie entwickelt sich der Populationsvektor x(t) der Investmentbanker im Laufe der Zeit?

Benutzen Sie hierbei die drei unterschiedlichen Parametersets der vorigen Folie.

Das Spiel der Geldpolitik

Fiskalbehörde / Geldbehörde	Keine neuen Schulden	Weiter Schulden machen
Finanzierung des Staatsdefizits über inflationäre Geldschöpfung	(3 , 3)	(2 , 4)
Stabile Geldpolitik	(4 , 2)	(0, 0)

Eine nationale, oder auch europäische Geldpolitik ist stets in einem fiskalpolitischen Diskurs. Die Geldbehörde (Zentralbank), die z.B. durch eine Verknappung der Geldmenge (kontraktive/restriktive) Geldpolitik bzw. eine Ausdehnung der Geldmenge (expansive Geldpolitik), eine stabile bzw. unstabile Strategie wählen kann, ist bestrebt ihre geldpolitischen Ziele (z.B. Preisniveaustabilität) durchzusetzen. Sowohl die Entscheidungsträger der Geldpolitik als auch die Politiker, welche eine fiskalpolitische Entscheidung zu treffen haben, befinden sich in einem wiederholten Spiel. Laut Gerhard Illing (Theorie der Geldpolitik, Kapitel 10.2) ist das gesamte geldpolitische Spiel, in erster Näherung, wie in der obigen Spielmatrix zu approximieren. Zusätzlich wirkt das globale Finanznetzwerk, zusammengesetzt (unter anderem) aus einer Vielzahl von Spekulanten, auf die Regierung ein, indem sie durch spekulativen Devisenhandel Währungskurse attackieren. Näheres siehe: Hochschul-Sommerkurses 2011, „Money, Money, Money: Deutschlands Wirtschafts- und Finanzleben“ (https://itp.uni-frankfurt.de/~hanauske/new/HSK_2011/index.html)

Das Spiel der Geldpolitik

Fiskalbehörde / Geldbehörde	Keine neuen Schulden	Weiter Schulden machen
Finanzierung des Staatsdefizits über inflationäre Geldschöpfung	(3 , 3)	(2 , 4)
Stabile Geldpolitik	(4 , 2)	(0, 0)

Obwohl hier eine symmetrische Spielmatrix vorliegt, ist das zugrundeliegende Spiel als Bi-Matrix Spiel zu beschreiben. In welche Klasse von Bi-Matrix Spielen ist das Spiel einzuordnen? Beschreiben Sie die möglichen zeitlichen Entwicklungen.

Benutzen Sie hierbei das folgende Maple oder Python Programm:
1) Bi-Matrix Spiele (Maple): https://itp.uni-frankfurt.de/~hanauske/VPSOC/T1/maple/I-2-4/BiMatrix1.html
2) Bi-Matrix Spiele (Python): https://itp.uni-frankfurt.de/~hanauske/VPSOC/2025/jupyter/EvolutionSpiel3.html

Anwendungsfelder der Spieltheorie (I)

- **Biologie**
  - **Verteilung von Bakterien in Organismen**

    Siehe z.B.: Kerr, Feldmann, Nature 2002
  - **Kooperation von Virus-Populationen**

    Siehe z.B.: Turner, Chao, Nature 1999
  - **Paarungsstrategien von Eidechsen**

    Siehe z.B.: Sinervo, Hazard, Nature 1996
  - **Evolutionäre Entwicklung von Makromolekülen**

    Siehe z.B.: Eigen, Schuster, Naturwissenschaften 64, 1977

Evolutionäre Spieltheorie
Evolutionäre Entwicklung von biologischen Systemen
Quasispezies und die Fitness der Genom Sequenz

Viele der in diesem Unterkapitel behandelten Systeme sind dem Buch Martin A. Nowak, Evolutionary Dynamics - Exploring the Equations of Life, 2006 entnommen, welches eine sehr gute und allgemein verständliche Einführung in das Themengebiet der evolutionären Dynamik darstellt. Obwohl der Fokus dieses Buches im Bereich der Evolution von biologischen Systemen liegt (siehe Kapitel 10: HIV Infection, Kapitel 11: Evolution of Virulence, Kapitel 12: Evolutionary Dynamics of Cancer, und Kapitel 13: Language Evolution), sind die Kapitel 1-9 weitgehend allgemein formuliert. Die evolutionäre Dynamik unterschiedlicher Spezien einer Tierart und der Mechanismus wie Tierarten ineinander übergehen wurde von Charles Darwin bereits im Jahre 1840 beschrieben. Im Jahre 1973 stellte John Maynard Smith eine Verbindung zwischen den Populationsgleichungen der Biologie und der evolutionären Spieltheorie her. Das Konzept der *Quasi-Spezien* (Ensemble von ähnlichen Genomen Sequenzen (Erbgut eines Lebewesens) welches durch einen Prozess der Mutation und Selektion entstanden ist) wurde von Manfred Eigen und Peter Schuster entwickelt (siehe Kapitel 3.3: Martin A. Nowak, Evolutionary Dynamics - Exploring the Equations of Life). Die Struktur der Quasi-Spezien Differentialgleichung ist den Gleichungen der evolutionären Spieltheorie sehr ähnlich (siehe Bild 3.4 und 4.5: Martin A. Nowak, Evolutionary Dynamics - Exploring the Equations of Life). Die evolutionäre Vorteilhaftigkeit einer Genom Sequenz wird hierbei als die *Fitness* der Quasi-Spezie bezeichnet. *Quasi-Spezien* entsprechen den Strategien der Spieltheorie und die Fitness kann als der Auszahlungswert einer Strategie aufgefasst werden. Die zeitliche Entwicklung der *Quasi-Spezien* am Beispiel des Paarungsverhalten von Eidechsen wird z.B. in siehe Sinervo, Barry, and Curt M. Lively. 'The rock-paper-scissors game and the evolution of alternative male strategies.' Nature 380.6571 (1996): 240. analysiert (siehe auch Vorlesung 6). Die evolutionäre Dynamik hängt von der unterliegenden Netzwerkstruktur der beteiligten Akteure ab und skalenfreie Netzwerkstrukturen agieren hier als Verstärker der evolutionären Selektion (siehe Kapitel 8, Evolutionary Graph Theory: Martin A. Nowak, Evolutionary Dynamics - Exploring the Equations of Life). Im folgenden Unterpunkt werden sie sogenannten *Spatial Games* behandelt (eine ausführliche Einführung findet sich im Kapitel 9: Martin A. Nowak, Evolutionary Dynamics - Exploring the Equations of Life).

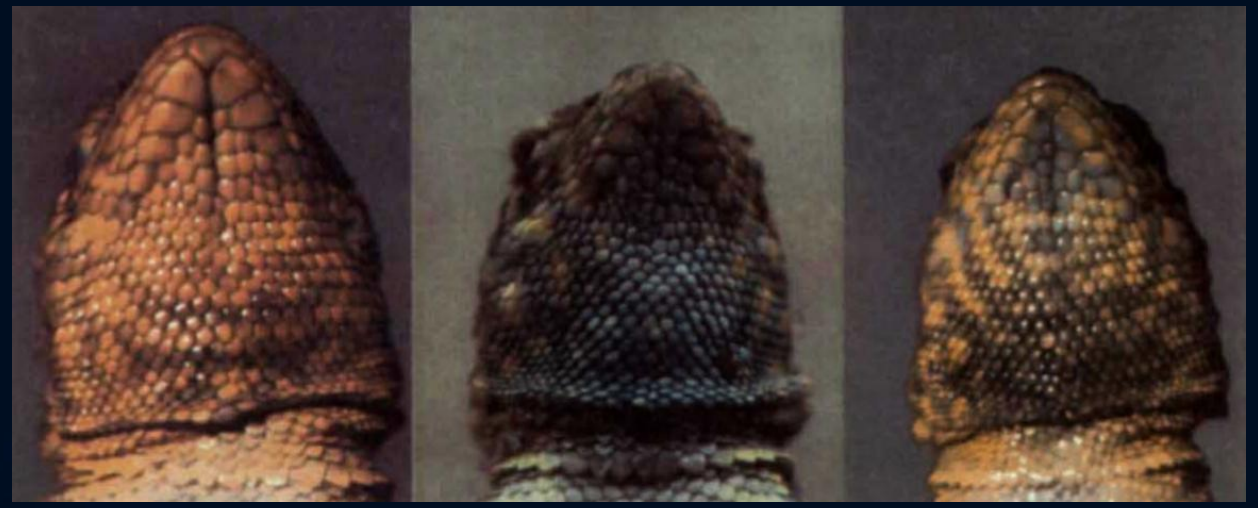
Siehe Teil III der Vorlesung

Beispiel

The rock–paper–scissors game and the evolution of alternative male strategies

B. Sinervo & C. M. Lively

Department of Biology and Center for the Integrative Study of Animal Behavior, Indiana University, Bloomington, Indiana 47405, USA



Evolutionäre Strategie (Quasi-Spezies)

Orange Blau Gelb

The Rock-Siccor-Paper Game
Replicatordynamics and ESS

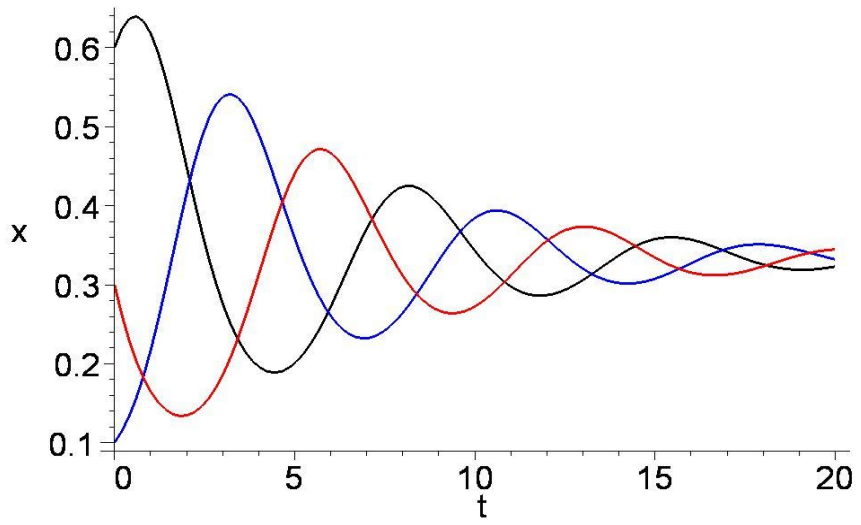
$$\frac{dx_1}{dt} = x_1 \cdot \left[2 \cdot x_2 - x_3 - \overline{\$} \right]$$

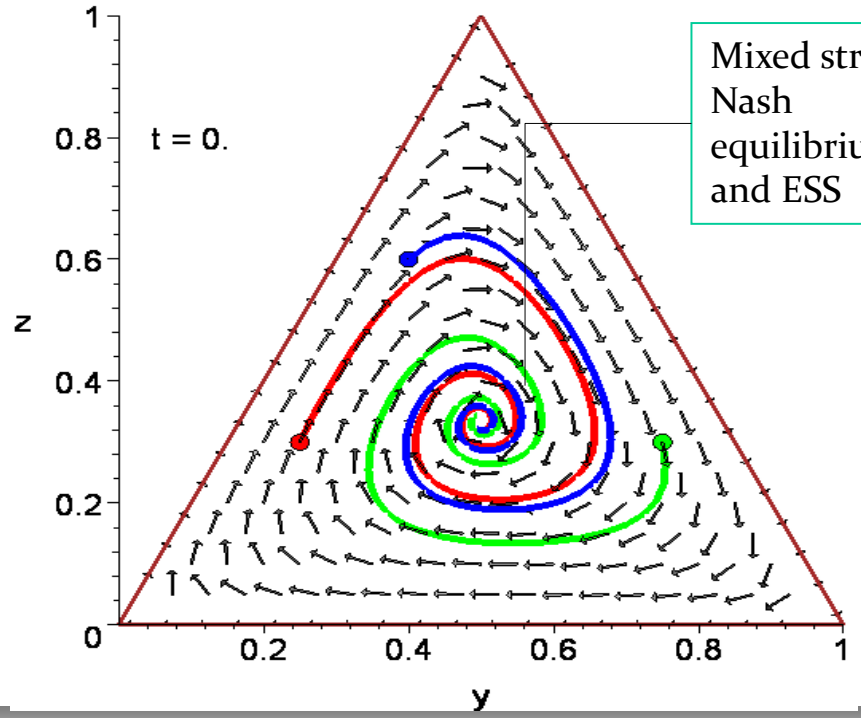
$$\frac{dx_2}{dt} = x_2 \cdot \left[-x_1 + 2 \cdot x_3 - \overline{\$} \right]$$

$$\frac{dx_3}{dt} = x_3 \cdot \left[2 \cdot x_1 - x_2 - \overline{\$} \right]$$

$$\text{with: } \overline{\$} = x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3$$

	Strategie 1	Strategie 2	Strategie 3
Strategie 1	(0, 0)	(1, -1)	(-1 , 1)
Strategie 2	(-1 , 1)	(0 , 0)	(1 , -1)
Strategie 3	(1 , -1)	(-1 , 1)	(0 , 0)





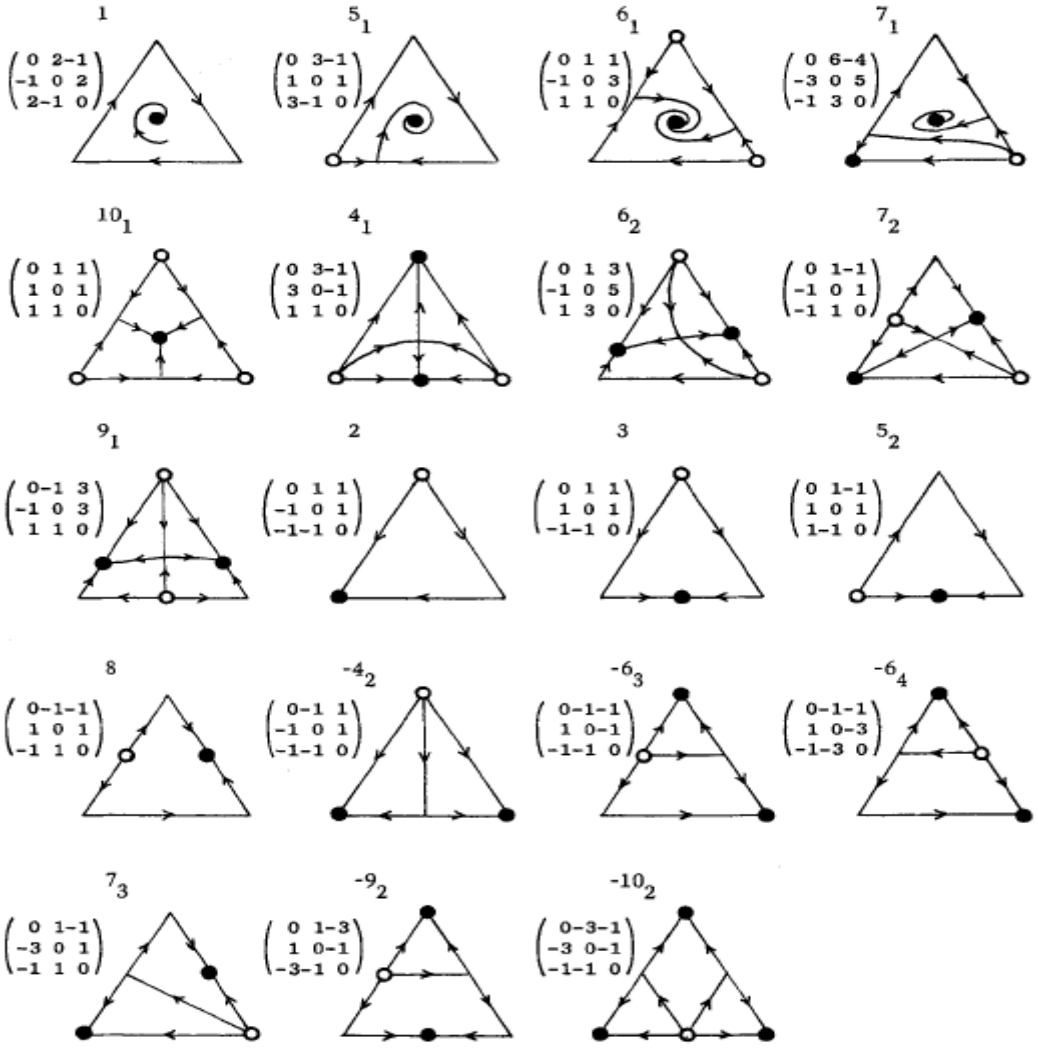
Mixed strategy Nash equilibrium and ESS

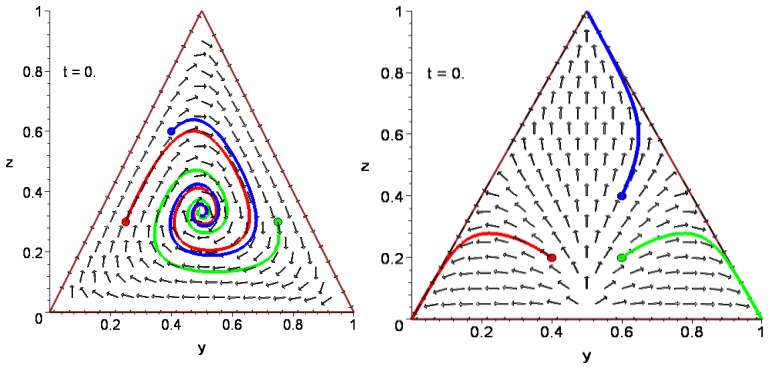
Using barycentric coordinates:

$$y := x_2 + \frac{x_3}{2}$$

$$z := x_3$$

Classes of symmetric (2x3) games

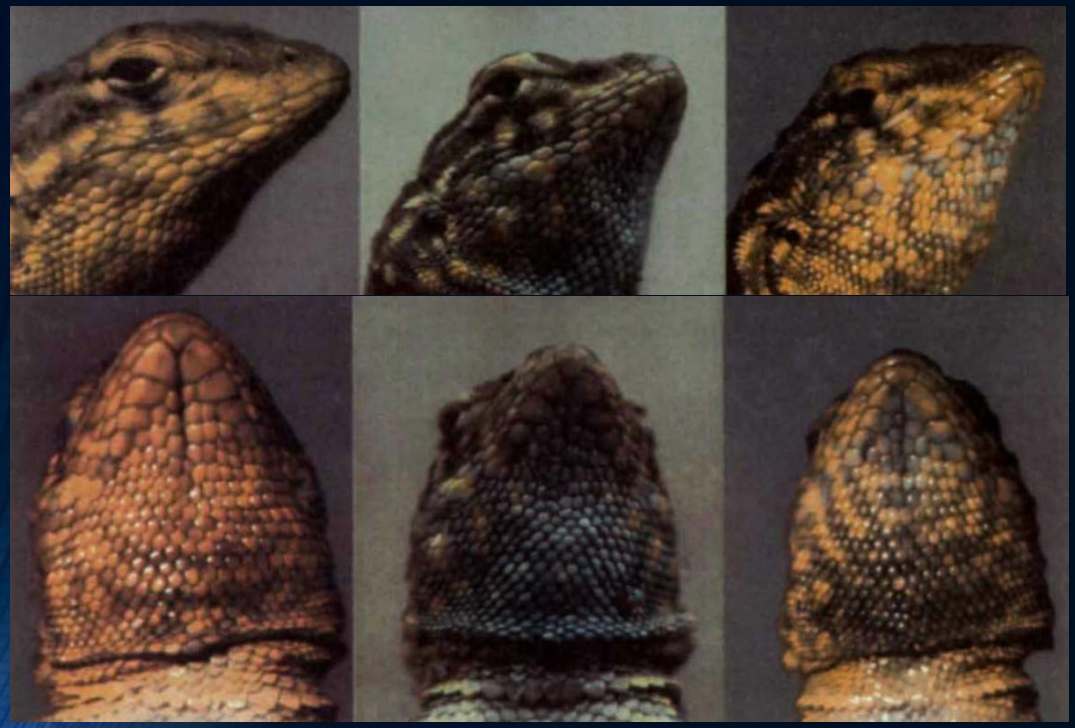




FE. C. Zeeman proved in his article that one can categorize symmetric evolutionary (2x3) games into 19 different classes. The figure on the left side shows that some classes have only one ESS (filled black circles), while others can have three ESSs.

E. C. Zeeman, *POPULATION DYNAMICS FROM GAME THEORY, In:* Global Theory of Dynamical Systems, Springer 1980

The Rock-Papers-Scissors Game and the Evolution of Sexual Selection



B.Sinervo and C.M.Lively focus within their article (The rock-paper-scissors game and the evolution of alternative male strategies, Nature, Vol.380 (1996)) on the sexual selection of male side-blotched lizards. From 1990-1995 they studied experimentally these animals and proposed an evolutionary model to explain their data.

Male fitness:
Number of monopolized + shared females

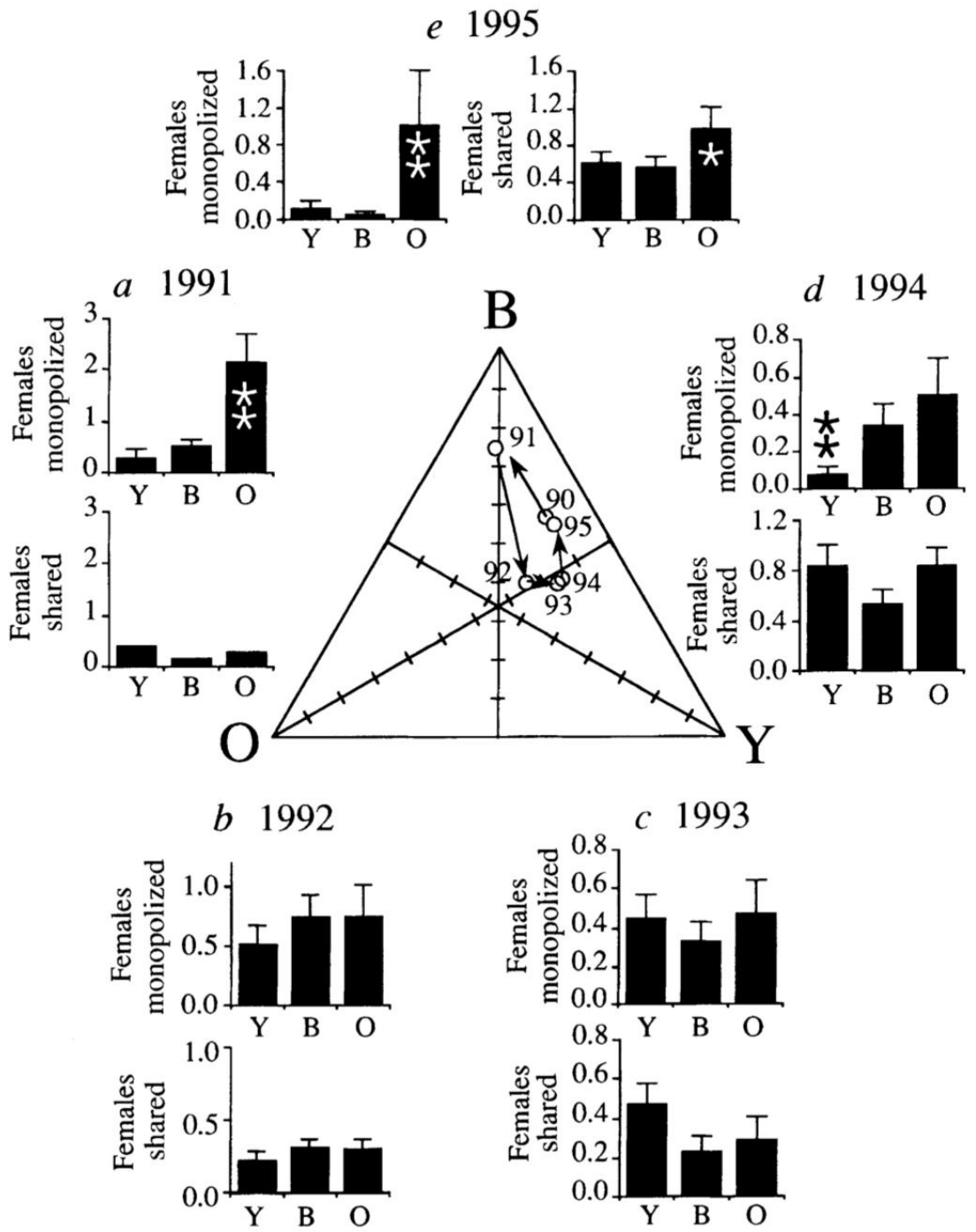
The male lizards have substantially three different colors, which are strongly connected to their behavior: **Orange** (very aggressive, defend large territories), **Blue** (less aggressive, defend small territories), **Yellow** (sneakers, look like females, not aggressive, do not defend territories). The payoff for the male lizards (their fitness) was estimated by the number of monopolized females (exclusively on his home range) and shared females (overlap to other territories).

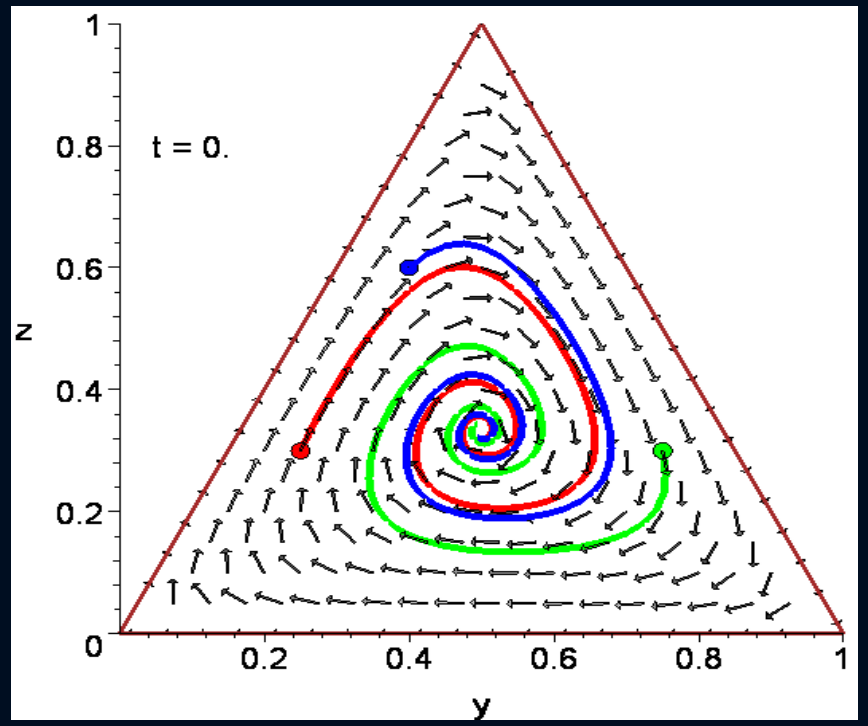
The Rock-Papers-Scissors Game and the Evolution of Sexual Selection

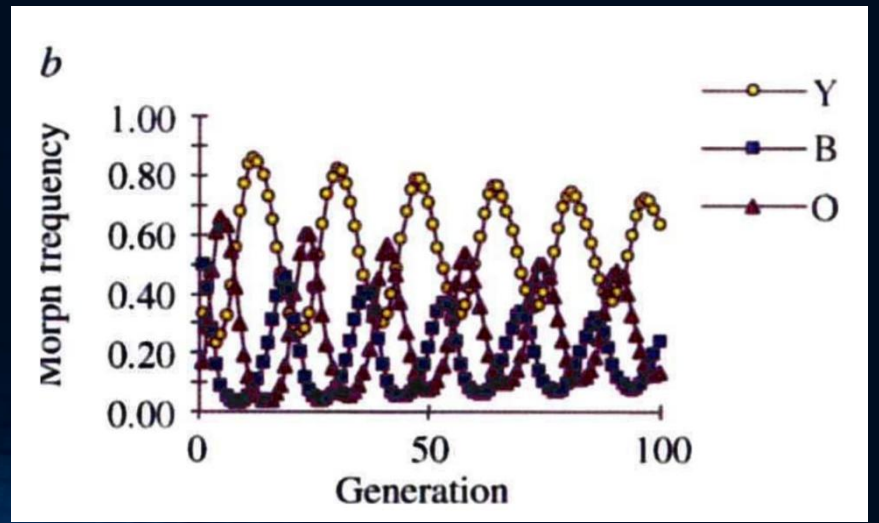
MANY species exhibit colour polymorphisms associated with alternative male reproductive strategies, including territorial males and 'sneaker males' that behave and look like females[1–3]. The prevalence of multiple morphs is a challenge to evolutionary theory because a single strategy should prevail unless morphs have exactly equal fitness[4,5] or a fitness advantage when rare[6,7]. We report here the application of an evolutionary stable strategy model to a three-morph mating system in the side-blotched lizard. Using parameter estimates from field data, the model predicted oscillations in morph frequency, and the frequencies of the three male morphs were found to oscillate over a six-year period in the field. The fitnesses of each morph relative to other morphs were non-transitive in that each morph could invade another morph when rare, but was itself invadable by another morph when common. Concordance between frequency-dependent selection and the among-year changes in morph fitnesses suggest that male interactions drive a dynamic 'rock–paper–scissors' game[7].

We have described the first biological example of a cyclical 'Rock–paper–scissors' game[7]. As in the game where paper beats rock, scissors beat paper, and rock beats scissors, the wide-ranging 'ultradominant' strategy of orange males is defeated by the 'sneaker' strategy of yellow males, which is in turn defeated by the mate-guarding strategy of blue males; the orange strategy defeats the blue strategy to complete the dynamic cycle. Frequency-dependent selection maintains substantial genetic variation in alternative male strategies, while at the same time prohibiting a stable equilibrium in morph frequency. □

The Rock-Papers-Scissors Game and the Evolution of Sexual Selection







III.3 Anwendungsfelder der Spieltheorie

III.3.1 Die Wissenschaft als komplexes Netzwerk (Models of Science Dynamics)

III.3.2 Sozio-ökonomische Labor- und Feldexperimente

III.3.3 Anwendungen in der Biologie

III.3.4 Anwendungen in den Politikwissenschaften

III.3.5 Spieltheorie und Auktionskonzepte

III.3.6 Finanzkrisen und evolutionäre Spiele

III.3.7 Sozio-ökonomische Netzwerke

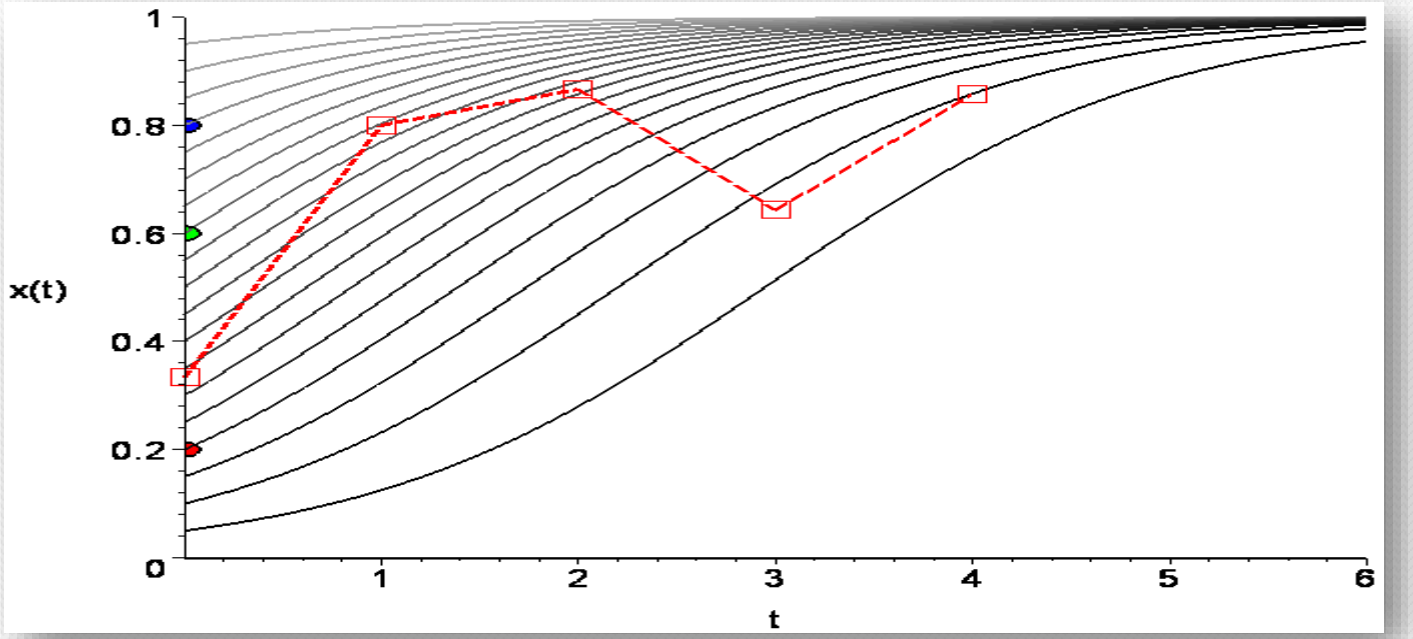
Anwendungsfelder der Spieltheorie (II)

- **Ökonomie**
  - **„Public Goods“- (Öffentliches Gut)- Spiele**
  - **Trust in Private and Common Property Experiments**, Elinor Ostrom, et al.
  - **Evolutionary Dynamics in Public Good Games**, CHRISTIANE CLEMENS and THOMAS RIECHMANN, Computational Economics (2006) 28: 399–420
  - **Institution Formation in Public Goods Games,** Michael Kosfeld, Akira Okada, and Arno Riedl, American Economic Review 2009, 99:4, 1335–1355
  - **Experimentelle Ökonomie**
  - **Cooperation in PD games: Fear, greed, and history of play,** T.K. AHN, ELINOR OSTROM, DAVID SCHMIDT, ROBERT SHUPP, Public Choice 106: 137–155, 2001.
  - **„Behavioral“- Verhaltensökonomie (Altruismus, Empathie, …)** z.B.: Fehr et al.
  - **Evolution von Informationsnetzwerken**

Theorie ⟺ Experiment

Experimentelle Ergebnisse des in Lyon gespielten Beispiels 1

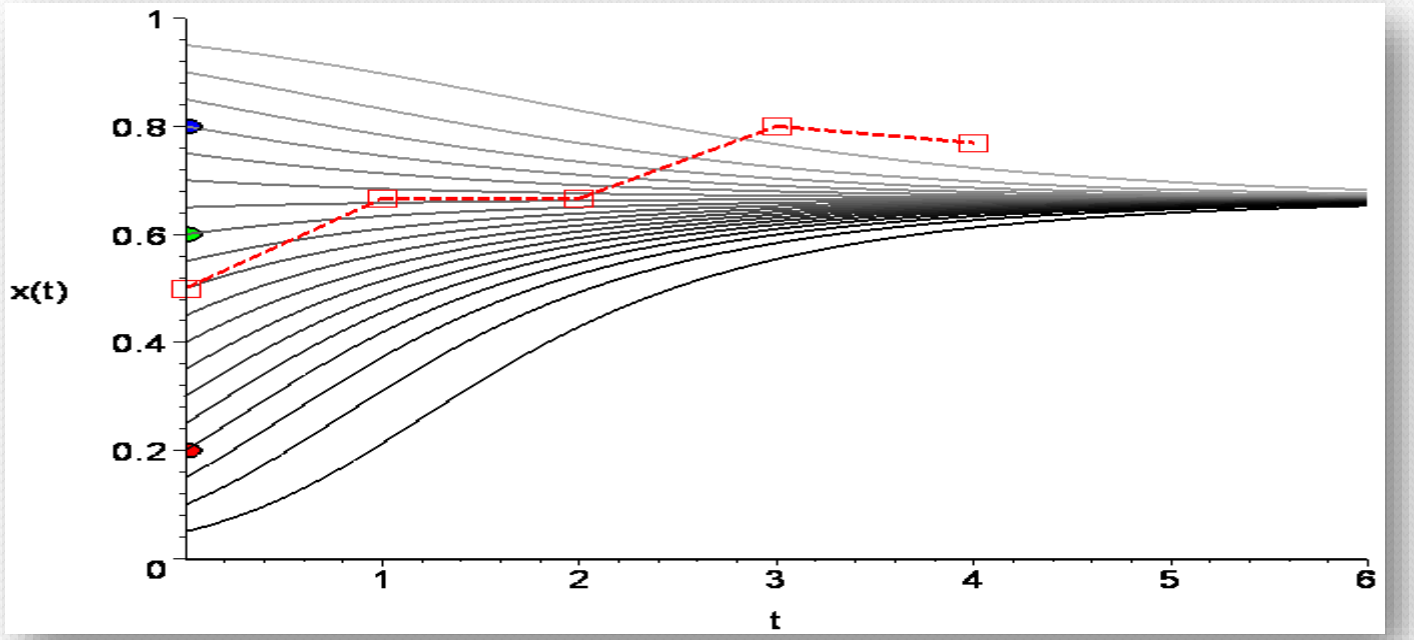
Experimentelle Ökonomie



Das erste Spiel besitzt nur ein Nash-Gleichgewicht das gleichzeitig die dominante Strategie des Spiels ist (Kugel, Kugel). Da es sich bei diesem Beispiel um ein dominantes, symmetrisches (2x2)-Spiel handelt und die Funktion g(x) im relevanten Bereich (x=[0,1]) immer größer-gleich Null ist, strebt der Populationsanteil der Kugel-Spieler unabhängig vom Anfangswert immer gegen die evolutionär stabile Strategie x=1. Die klassische evolutionäre Spieltheorie sagt demnach voraus, dass die Spieler innerhalb der betrachteten Population nach einer gewissen Zeit maßgeblich die Strategie Kugel wählen (x=1). Die rote Kurve in der obigen Abbildung zeigt die experimentellen Ergebnisse des im Vorlesungsteil 4 gespielten Beispiels 1.

Theorie ⟺ Experiment

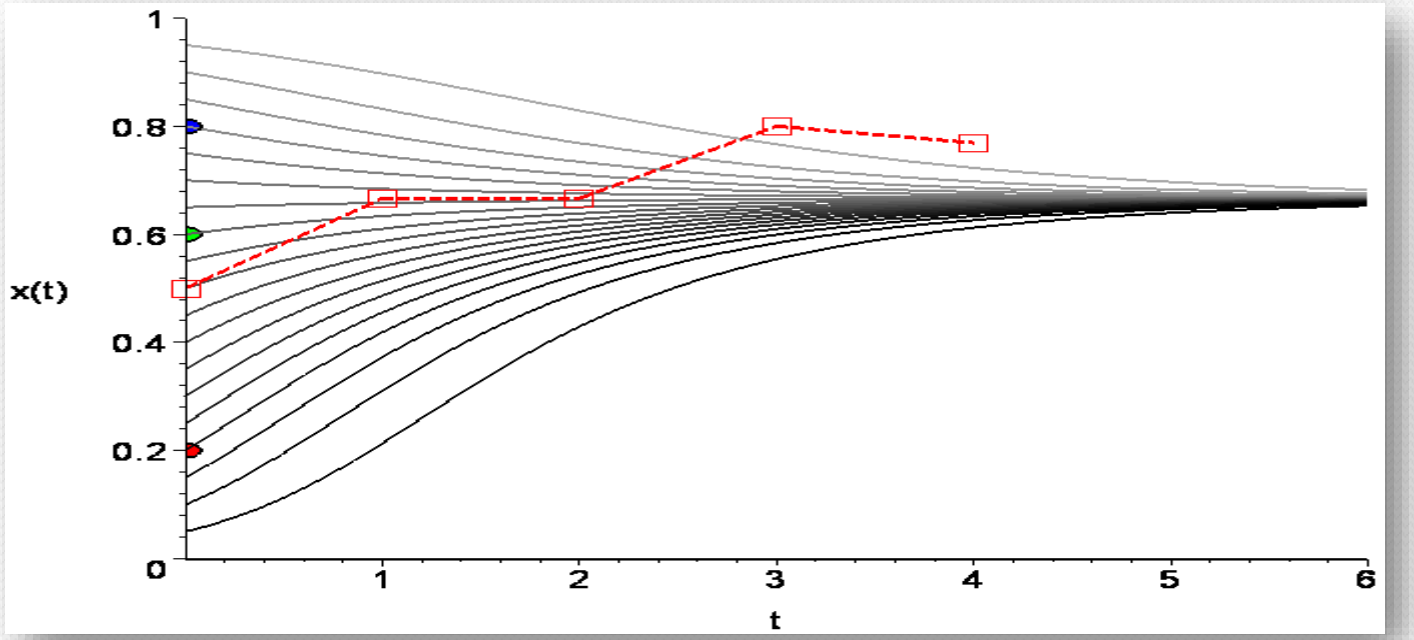
Experimentelle Ergebnisse des in Lyon gespielten Beispiels 2



Das zweite Spiel besitzt keine dominante Strategie, aber zwei unsymmetrische Nash-Gleichgewicht in reinen Strategien ((K,KK) und (KK,K)) und ein gemischtes Nash-Gleichgewicht (0.67 K , 0.33 KK). Da es sich bei diesem Beispiel um ein symmetrisches Anti-Koordinationsspiel handelt, strebt der Populationsanteil der Kugel-Spieler unabhängig vom Anfangswert immer zu dem gemischten Nash-Gleichgewicht (der einzigen evolutionär stabilen Strategie des Spiels), was identisch mit der mittleren Nullstelle der Funktion g(x) ist (x=0.67). Die rote Kurve in der obigen Abbildung zeigt die experimentellen Ergebnisse des im Vorlesungsteil 4 gespielten Beispiels 2.

Theorie ⟺ Experiment

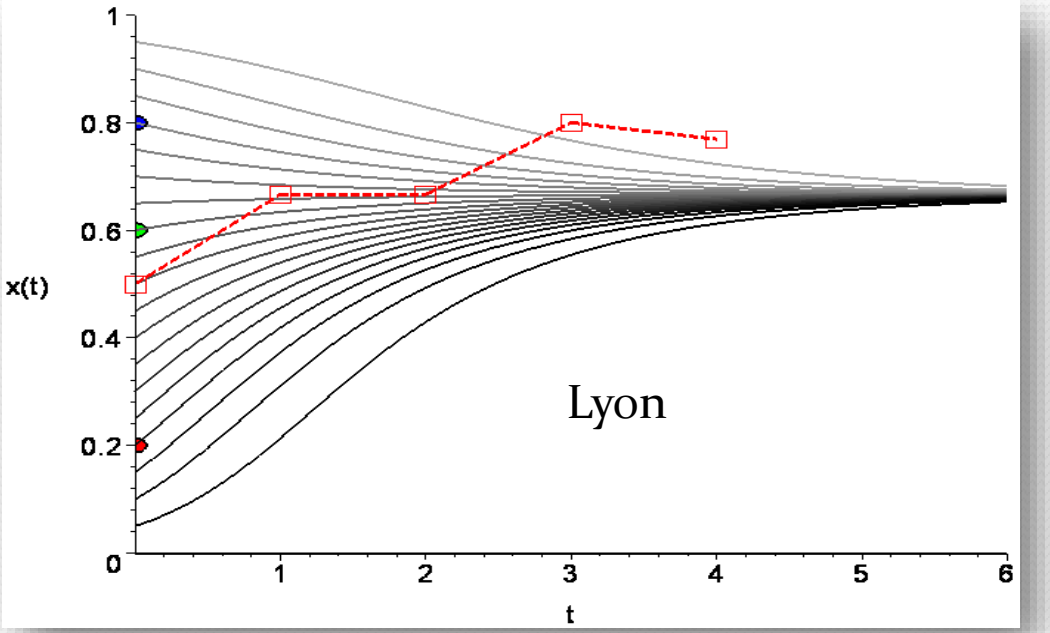
Experimentelle Ergebnisse des in Lyon gespielten Beispiels 2

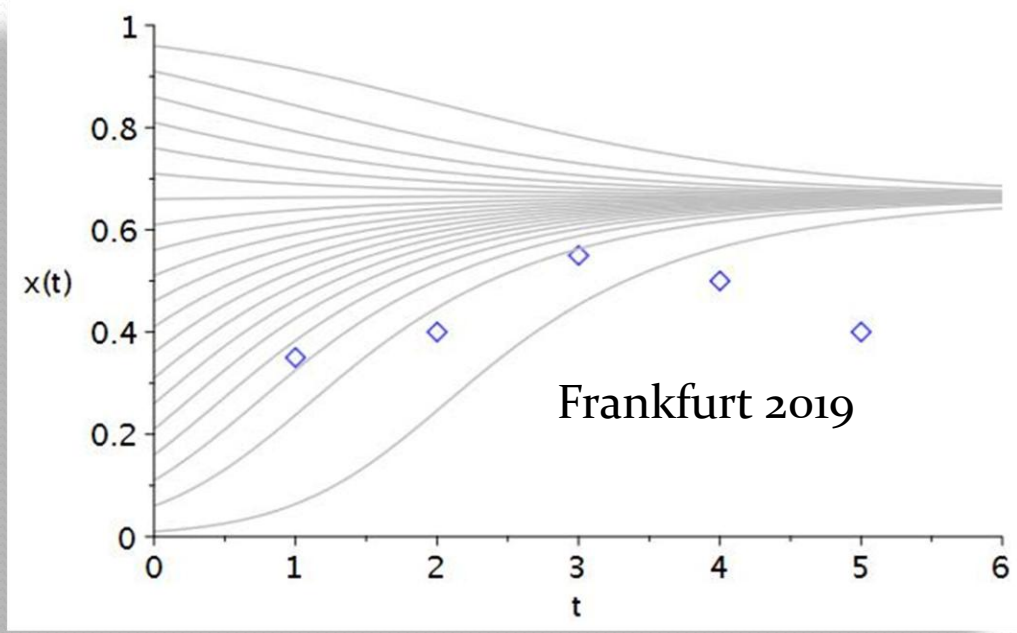


Das zweite Spiel besitzt keine dominante Strategie, aber zwei unsymmetrische Nash-Gleichgewicht in reinen Strategien ((K,KK) und (KK,K)) und ein gemischtes Nash-Gleichgewicht (0.67 K , 0.33 KK). Da es sich bei diesem Beispiel um ein symmetrisches Anti-Koordinationsspiel handelt, strebt der Populationsanteil der Kugel-Spieler unabhängig vom Anfangswert immer zu dem gemischten Nash-Gleichgewicht (der einzigen evolutionär stabilen Strategie des Spiels), was identisch mit der mittleren Nullstelle der Funktion g(x) ist (x=0.67). Die rote Kurve in der obigen Abbildung zeigt die experimentellen Ergebnisse des im Vorlesungsteil 4 gespielten Beispiels 2.

Theorie ⟺ Experiment

Experimentelle Ergebnisse des Beispiels 2

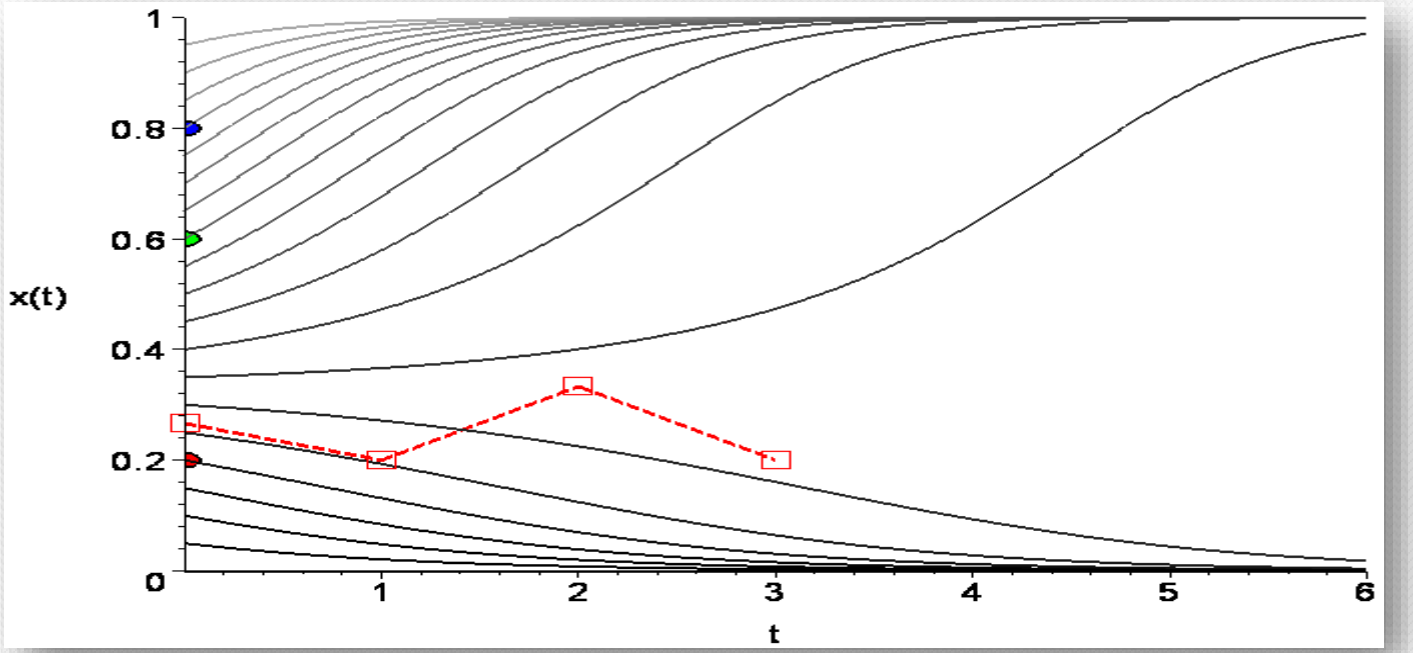




Das zweite Spiel besitzt keine dominante Strategie, aber zwei unsymmetrische Nash-Gleichgewicht in reinen Strategien ((K,KK) und (KK,K)) und ein gemischtes Nash-Gleichgewicht (0.67 K , 0.33 KK). Da es sich bei diesem Beispiel um ein symmetrisches Anti-Koordinationsspiel handelt, strebt der Populationsanteil der Kugel-Spieler unabhängig vom Anfangswert immer zu dem gemischten Nash-Gleichgewicht (der einzigen evolutionär stabilen Strategie des Spiels), was identisch mit der mittleren Nullstelle der Funktion g(x) ist (x=0.67). Die rote Kurve in der linken obigen Abbildung zeigt die experimentellen Ergebnisse des in Lyon gespielten Beispiels 2. Die blauen Punkte in der rechten obigen Abbildung zeigt die experimentellen Ergebnisse des in Frankfurt gespielten Beispiels 2 (zum Vergleich bitte die blauen Punkte um -1 auf der Zeitachse verschieben).

Theorie ⟺ Experiment

Experimentelle Ergebnisse des in Lyon gespielten Beispiels 3



Das dritte Spiel besitzt ebenfalls keine dominante Strategie, aber zwei symmetrische Nash-Gleichgewicht in reinen Strategien ((K,K) und (KK,KK)) und ein gemischtes Nash-Gleichgewicht (0.33 K , 0.67 KK). Da es sich bei diesem Beispiel um ein symmetrisches Koordinationsspiel handelt, strebt der Populationsanteil der Kugel-Spieler abhängig vom Anfangswert zu einem der beiden reinen Nash-Gleichgewichte (x=1 oder x=0). Die klassische evolutionäre Spieltheorie sagt demnach voraus, dass es zwei evolutionär stabile Strategien gibt (x=1 oder x=0). Die rote Kurve in der obigen Abbildung zeigt die experimentellen Ergebnisse des im Vorlesungsteil 4 gespielten Beispiels 3.

Anwendungsfelder der Spieltheorie (III)

- Sozialwissenschaft
  - **Kulturelle und moralische Entwicklungen**
  - **Evolution of social learning does not explain the origin of human cumulative culture**, Magnus Enquist, Stefano Ghirlanda, Journal of Theoretical Biology 246 (2007) 129–135
  - **EVOLUTION OF MORAL NORMS,** William Harms and Brian Skyrms, For Oxford Handbook on the Philosophy of Biology ed. Michael Ruse
  - **Evolution der Sprache**
  - **Finite populations choose an optimal language**, Christina Pawlowitsch**,** Journal of Theoretical Biology 249 (2007) 606–616
  - **Soziales Lernen**
  - **Evolution of social learning does not explain the origin of human cumulative culture**, Magnus Enquist, Stefano Ghirlanda, Journal of Theoretical Biology 246 (2007) 129–135
  - **Evolution von sozialen Normen**
  - **Collective Action and the Evolution of Social Norms,** Elinor Ostrom, The Journal of Economic Perspectives, Vol. 14, No. 3 (Summer, 2000), pp. 137-158
  - **Evolution von sozialen Netzwerken**
  - **GOVERNING SOCIAL-ECOLOGICAL SYSTEMS,** MARCO A. JANSSEN and ELINOR OSTROM
  - **A General Framework for Analyzing Sustainability of Social-Ecological Systems,** Elinor Ostrom, et al., Science 325, 419 (2009)

Deutsche Physikalische Gesellschaft DPG

Physics of Socio-Economic Systems Division

DPG Spring Meeting Berlin, March 25 - 30, 2012

Scope

- Financial Markets and Risk Management
- Economic Models and Evolutionary Game Theory
- Traffic Dynamics, Urban and Regional Systems
- Social Systems, Opinion and Group Dynamics
- Networks: From Topology to Dynamics

Keynote Talk

H. Eugene Stanley (Boston, USA)
"Interdependent Networks and Switching Phenomena"

Young Scientist Award for Socio- and Econophysics*

Keynote Speaker: **Stefan Thurner** (Wien, A)
"The Role of Agent Based Models in Understanding Human Societies"

* supported by d-fine

Tutorial "Scientific Writing"**

Hernan Rozenfeld (APS, USA)
Tim Smith (IOP Publishing, UK)

Invited Talks

Thilo Gross (Bristol, UK) "Adaptive Networks of Opinion Formation in Humans and Animals"

Marc Hütt (Bremen) "Common Design Principles of Metabolic Networks and Industrial Production"

Focus Session: Big Data**

Rosario Mantegna (Palermo, IT)
"Econophysics and Social Research with Large Sets of Data"

Philip Treleaven (London, UK)
"Experimental Computational Finance & Big Data Environment"

Tiziana Di Matteo (London, UK)
"Embedding High Dimensional Data on Networks"

Michael Batty (London, UK)
"Cities and Complexity"

Focus Session: Models of War, Conflict and Revolutions

Neil Johnson (Miami, USA)
"Escalation, Timing and Severity of Insurgent and Terrorist Events: Robust Patterns and a Generic Model"

Aaron Clauset (Boulder, USA)
"Fatality Dynamics and the Limits of Civil and Interstate Wars"

Ravinder Bhavnani (Geneva, CH)
"Group Segregation and Urban Violence"

**Sessions are organized with the jDPG

Registration via http://berlin12.dpg-tagungen.de/index.html?lang=en
Conference Languages: English and German
Deadline: December 1th 2011
Young Scientist Award: Call for nominations and applications at http://www.dpg-physik.de/dpg/gliederung/fv/soe/YSA/call.html
Deadline: December 1th 2011

Contact

Prof. Dr. Dirk Helbing, Dr. Jörg Reichardt and Dr. Tobias Preis,
Chairmen of the Physics of Socio-Economic Systems Division (Φ·SOE), http://www.phi-soe.de/

Tutorial

SOE 1.1 Sun 16:00–18:00 HSZ 04 **Collective Dynamics of Firms: A Statistical Physics Approach** — •Frank Schweitzer

Focus Session: Swarm Intelligence

SOE 2.1 Mon 10:15–10:45 GÖR 226 **Social Media and Attention** — •Bernardo Huberman
SOE 2.2 Mon 10:45–11:15 GÖR 226 **Mobilizing society with a red balloon** — •Riley Crane
SOE 2.3 Mon 11:15–11:45 GÖR 226 **Collective behaviour and swarm intelligence** — •Jens Krause

Focus Session: GPU-Computing (with DY)

SOE 5.1 Mon 14:00–14:30 GÖR 226 **Applications of GPU-Computing in Statistical Physics** — •Peter Virnau
SOE 5.2 Mon 14:30–15:00 GÖR 226 **Accelerating Monte Carlo Simulations in Statistical Physics with GPU's** — •David Landau

Focus Session: Experimental Methods

SOE 10.1 Tue 13:30–14:00 GÖR 226 **Complex Economic Systems in the Laboratory** — •Cars Hommes
SOE 10.2 Tue 14:00–14:30 GÖR 226 **Multiplicative Cascades: How to model trip within cities** — •Marta C. González
SOE 10.3 Tue 14:30–15:00 GÖR 226 **Human behavior on networks: lessons and perspectives from game theory** — •Angel Sánchez
SOE 10.4 Tue 15:00–15:30 GÖR 226 **Measuring Happiness** — •Peter S. Dodds

Young Scientist Award for Socio- and Econophysics

SOE 8.1 Mon 17:00–17:45 HSZ 02 **Dragon-kings versus black swans: diagnostics and forecasts for the on-going world financial crisis** — •Didier Sornette
SOE 8.1 Mon 18:00–18:30 HSZ 02 **Community structure in networks and statistical physics of social dynamics** — •Santo Fortunato

Joint Symposium on Foundations and Perspectives of Climate Engineering (with AKE, UP)

See SYCE for the full program of the symposium.

SYCE 1.1 Tue 10:30–11:00 HSZ 01 **Oceanic carbon-dioxide removal options: Potential impacts and side effects** — •Andreas Oschlies
SYCE 1.2 Tue 11:00–11:30 HSZ 01 **Climate Engineering through injection of aerosol particles into the atmosphere: physical insights into the possibilities and risks** — •Mark Lawrence
SYCE 1.3 Tue 11:30–12:00 HSZ 01 **Geoengineering - will it change the climate game?** — •Timo Goeschl
SYCE 1.4 Tue 12:00–12:30 HSZ 01 **The gamble with the climate - an experiment** — •Manfred Milinski

Plenary Talks related to SOE

PV X Thu 8:30– 9:15 H1 **Complex Networks: From Statistical Physics to the Cell** — •Albert Laszlo Barabasi

Tutorial

SOE 1.1 Sun 16:00–18:00 H10 **Time Series Analysis in Sociophysics and Econophysics** — •Johannes J. Schneider, •Tobias Preis

Invited Talks

SOE 2.1 Mon 9:30–10:15 H44 **Don't panic! - The physics of pedestrian dynamics and evacuation processes** — •Andreas Schadschneider
SOE 7.1 Tue 9:30–10:00 H44 **Humans playing spatial games** — •Arne Traulsen
SOE 12.1 Wed 9:30–10:15 H44 **The hidden complexity of open source software** — •Frank Schweitzer
SOE 17.1 Thu 9:30–10:15 H44 **Wave localization in complex networks** — •Jan W. Kantelhardt
SOE 22.1 Fri 9:30–10:15 H44 **Hypergraphs and social systems** — •Guido Caldarelli

Focus Session: Science of Science

SOE 4.1 Mon 13:30–14:00 H44 **Following the actors: individual and collective behavior in epistemic landscapes** — •Andrea Scharnhorst
SOE 4.2 Mon 14:00–14:30 H44 **Tracking science in real-time from large-scale usage data.** — •Johan Bollen
SOE 4.3 Mon 14:45–15:15 H44 **Mapping change in science** — •Martin Rosvall, Carl Bergstrom
SOE 4.4 Mon 15:15–15:45 H44 **Statistical physics of citation behavior** — •Santo Fortunato

Elfmeter im Fussball: Übergang von einem (2x2)-Spiel zu einem (2x3)-Spiel

Wolfgang Leininger and Axel Ockenfels*

The Penalty-Duel and Institutional Design: Is there a Neeskens-Effect?

Abstract

We document an increase in the scoring probability from penalties in soccer, which separates the time period before 1974 significantly from that after 1976: the scoring probability increased by 11%. We explain this finding by arguing that the *institution* of penalty-shooting before 1974 is best described as a *standard of behaviour* for striker and goal-keeper, which in game-theoretic terms represents a 2x2-game. In contrast to this, after 1976 the institution of the penalty-duel is best described by a 3x3 game form constrained by certain behavioural rules. Those rules can be parameterized by a *single* parameter, which nevertheless allows the theoretical reproduction (and hence explanation) of all the empirically documented regularities. The scoring probability in equilibrium of the latter institution is higher than in the former one. We present historical evidence to the effect, that this change in the perception of penalty-duels (as two different games), was caused by Johan Neeskens' shrewd and "revolutionary" penalty-taking during World-Cup 1974, when he shot a penalty in the first minute of the final between Germany and the Netherlands right into the *middle* of the goalmouth.

The following application is based on a working paper by W.Leininger and A.Ockenfels (CESIFO WORKING PAPER NO. 2187, 2008). The article focuses on the 'Penalty-Duel' in soccer and describes it as a simultanious two player game – a game between the goalkeeper and the kicker.

Neeskens Elfmeter:

https://www.youtube.com/watch?v=44HvFzhV9xI

Artikel:

https://www.econstor.eu/bitstream/10419/26769/1/528420186.PDF

The future of theoretical evolutionary game theory

PHILOSOPHICAL TRANSACTIONS B

royalsocietypublishing.org/journal/rstb

The future of theoretical evolutionary game theory

Arne Traulsen[1] and Nikoleta E. Glynatsi[2]

[1]Department of Evolutionary Theory, and [2]Max Planck Research Group: Dynamics of Social Behavior, Max Planck Institute for Evolutionary Biology, Plön 24306, Germany

AT, 0000-0002-0669-5267

Opinion piece

Cite this article: Traulsen A, Glynatsi NE.

Evolutionary game theory is a truly interdisciplinary subject that goes well
beyond the limits of biology. Mathematical minds get hooked up in simple
move into other parts of evolution-
alize how much they can learn from
nsfer insight that was originally gen-
an use their algorithms to explore a
rn from the environment, but also
and the focus on a few very popular
ice: several insights are re-discovered
with different heroes and modelling
atial structure are treated differently.
allel? Or can we converge to a single
ally a single software repository? Or
each other, learning from each other

Dynamic Games and Applications
https://doi.org/10.1007/s13235-023-00545-4

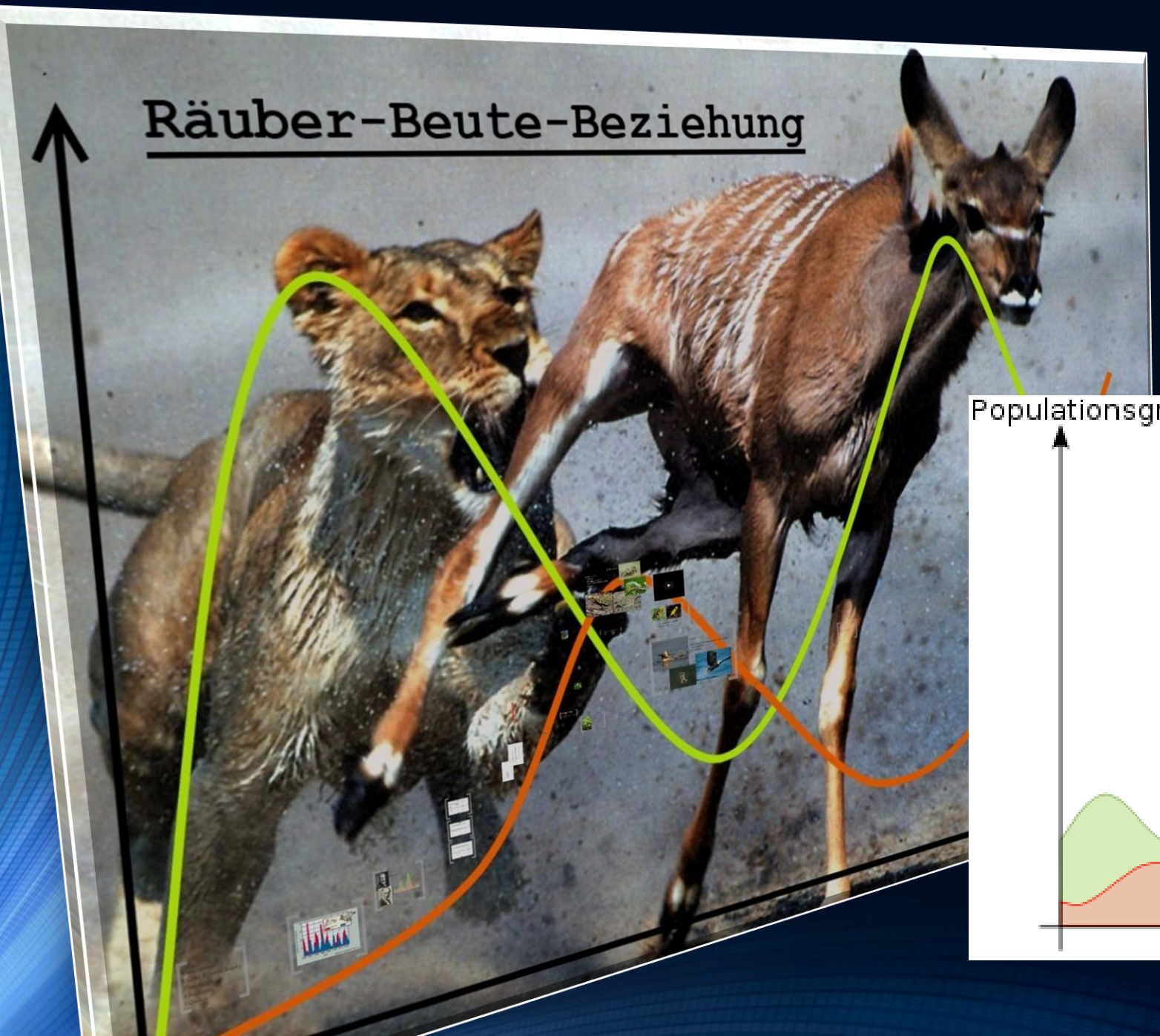
PREFACE

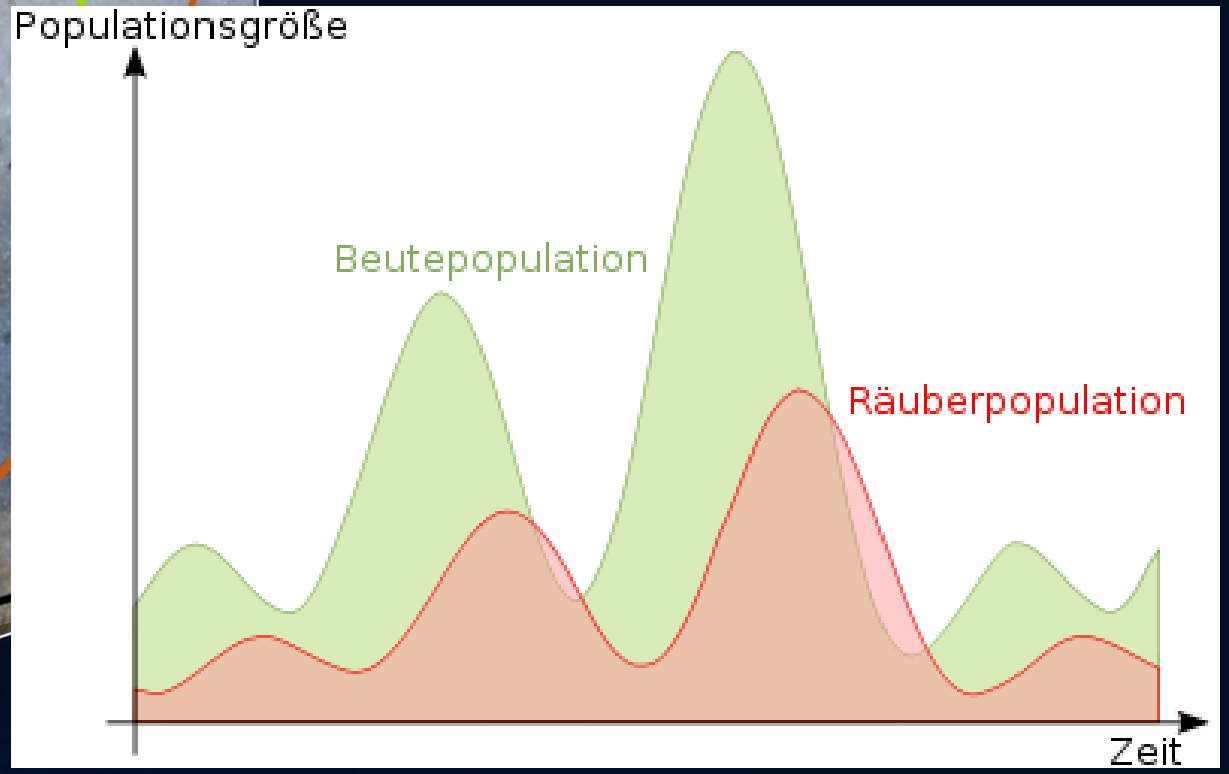
Evolutionary Games and Applications: Fifty Years of 'The Logic of Animal Conflict'

Evolutionary Games and Applications: Fifty Years of 'The Logic of Animal Conflict'

Christian Hilbe[1] · Maria Kleshnina[2] · Kateřina Staňková[3]

Das Räuber-Beute Spiel





Die Lotka-Volterra-Gleichung (Räuber-Beute-Gleichung) für N-Populationen

$$\frac{dx_i(t)}{dt} = \left(r_i + \sum_{j=1}^{N} b_{ij}\, x_j(t) \right) x_i(t)$$

Anzahl der Räuber/Beute Wesen der i-ten Population zur Zeit t

Reproduktions- bzw. Sterberaten

Interaktionsmatrix

Evolutionäre Spieltheorie und die Räuber-Beute-Gleichung

Wir wollen nun die Äquivalenz der Räuber-Beute-Gleichung (Lotka-Volterra Gleichung) für n-Populationen mit der Replikatordynamik der evolutionären Spieltheorie für $m = n + 1$ Strategien zeigen. Das System von Differentialgleichungen eines symmetrischen evolutionären Spiels mit m Strategien lautet:

Auf der Internetseite der Vorlesung

$$\frac{dx_i}{dt} = \left(\sum_{j=1}^{m} \$_{ij}\, x_j - \sum_{k=1}^{m} \sum_{j=1}^{m} \$_{kj}\, x_k\, x_j \right) x_i \qquad (4)$$

Die Interaktionsmatrix b_{ij} und die intrinsischen Reproduktion-Sterberaten r_i sind dabei mit der Auszahlungsmatrix $\$_{ij}$ des evolutionären Spiels in einer speziellen Weise verknüpft (siehe Sigmund/Hofbauer S:77 bzw. Novak "Evolutionary dynamics", p.68). Es gelten dabei die folgenden Zusammenhänge:

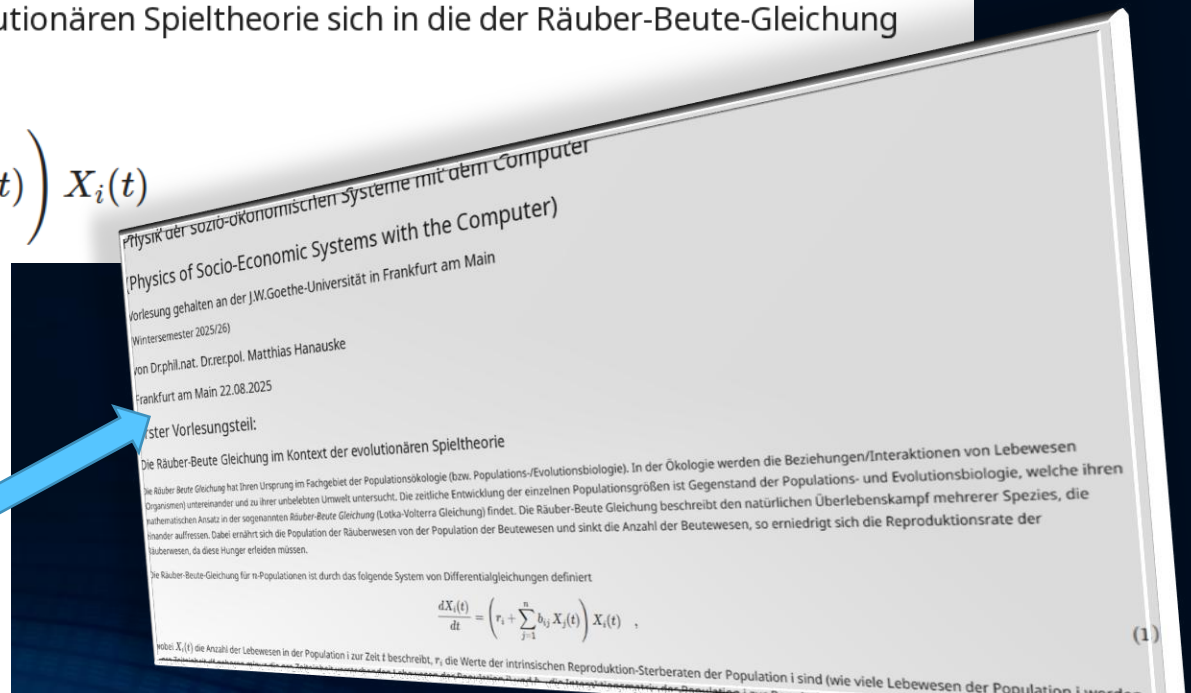
$$r_i = \$_{im} - \$_{mm}\ , \quad b_{ij} = \$_{ij} - \$_{mj}$$

Jupyter Notebook

Im Folgenden befassen wir uns mit dem Räuber-Beute-Spiel zweier Populationen (Anzahl der Strategien m=3, entspricht n=2 Populationen) und zeigen anhand der oben behandelten Beispiele, wie die Gleichungen der evolutionären Spieltheorie sich in die der Räuber-Beute-Gleichung überführen lassen.

$$\frac{dX_i(t)}{dt} = \left(r_i + \sum_{j=1}^{n} b_{ij}\, X_j(t) \right) X_i(t)$$

- Folien der 5.Vorlesung
- Vorlesungsaufzeichnung der 5.Vorlesung: WS 2022/23 bzw. WS 2021/22
- View Jupyter Notebook: Die 19 Klassen der evolutionären symmetrischen (2x3)-Spiele
- Download Jupyter Notebook: Die 19 Klassen der evolutionären symmetrischen (2x3)-Spiele
- Download Python Programm: Evolutionäre Spieltheorie symmetrischer (2x3)-Spiele (2x3-Spiel_ng.py)
- View Jupyter Notebook: Die Räuber-Beute Gleichung im Kontext der evolutionären Spieltheorie
- Download Jupyter Notebook: Die Räuber-Beute Gleichung im Kontext der evolutionären Spieltheorie
- Maple Worksheet: Äquivalenz der Räuber-Gleichung für N-Populationen mit der Replikatorgleichung der evolutionären Spieltheorie für (N+1)-Strategien
- Download Jupyter Notebook VPSOC_DGL_1.ipynb
- View Jupyter Notebook VPSOC_DGL_1.html
- Download C++ Programm evol1.cpp


Physik der sozio-ökonomischen Systeme mit dem Computer
(Physics of Socio-Economic Systems with the Computer)
Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main
(Wintersemester 2025/26)
von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske
Frankfurt am Main 22.08.2025
Erster Vorlesungsteil:
Die Räuber-Beute Gleichung im Kontext der evolutionären Spieltheorie
Die Räuber Beute Gleichung hat Ihren Ursprung im Fachgebiet der Populationsökologie (bzw. Populations-/Evolutionsbiologie). In der Ökologie werden die Beziehungen/Interaktionen von Lebewesen (Organismen) untereinander und zu ihrer unbelebten Umwelt untersucht. Die zeitliche Entwicklung der einzelnen Populationsgrößen ist Gegenstand der Populations- und Evolutionsbiologie, welche ihren mathematischen Ansatz in der sogenannten Räuber-Beute Gleichung (Lotka-Volterra Gleichung) findet. Die Räuber-Beute Gleichung beschreibt den natürlichen Überlebenskampf mehrerer Spezies, die einander auffressen. Dabei ernährt sich die Population der Räuberwesen von der Population der Beutewesen und sinkt die Anzahl der Beutewesen, so erniedrigt sich die Reproduktionsrate der Räuberwesen, da diese Hunger erleiden müssen.
Die Räuber-Beute-Gleichung für n-Populationen ist durch das folgende System von Differentialgleichungen definiert
$$\frac{dX_i(t)}{dt} = \left(r_i + \sum_{j=1}^{n} b_{ij}\, X_j(t) \right) X_i(t) \ , \qquad (1)$$
wobei $X_i(t)$ die Anzahl der Lebewesen in der Population i zur Zeit t beschreibt, r_i die Werte der intrinsischen Reproduktion-Sterberaten der Population i sind (wie viele Lebewesen der Population i werden