WalezuLogistik

November 23, 2020

1 Von den Walen zur logistischen Gleichung

1.1 Herleitung

1.1.1 Recap des Walpopulationsmodells

Wir haben für die Entwicklung der Walpopulation ein Modell vom Character

$$W_{k+1} = W_k + \rho W_k(K - W_k)$$

wobei W_k die Anzahl Wale zum Zeitpunkt k, $\rho > 0$ einen Konstante, die unter anderem vom Wachstum der Population abhängig ist und K die Kapazitätsgrenze der Populationsgrösse bezeichnen. Auf gym math gibts das Video zur Walpopulation auch zum Nachschauen.

Man kann nachrechnen, dass 0 und K Fixpunkte dieses Iterationsmodells sind und für gewisse, vernünftige ρ und K der Fixpunkt 0 instabil und der Fixpunkt K stabil ist.

Übung : Rechne nach.

Experimentiert man ein bisl, kann man beim Modell gucken, was passiert, wenn man seine Grenzen auslotet. Wir setzen das Modell noch mal auf. Dazu bestimmen wir zuerst den Parameter ρ zu einem vermuteten Wachstum von $\tilde{q} = 8\%$ jährlich, Startwert $W_0 = 1\,200$ und Kapazitätsgrenze $K = 20\,000$.

[1]:
```
import sympy as sy

W0 = 1200  # starting value
K = 20_000  # capacity
q = 1 + 0.08  # growth factor
rho = sy.symbols('rho') # set r als sympy symbol

eqrho = sy.Eq(q*W0, W0 + rho*(K-W0)*W0) # set equation
solrho = sy.solve(eqrho) # solve for rho
numrho = solrho[0] # get float
print(r'rho = ' + str(numrho))
```

```
rho = 4.25531914893617e-6
```

[2]:
```
numrho
```

[2]:
$$4.25531914893617 \cdot 10^{-6}$$

[3]:
```
%%latex
\normalfont
```

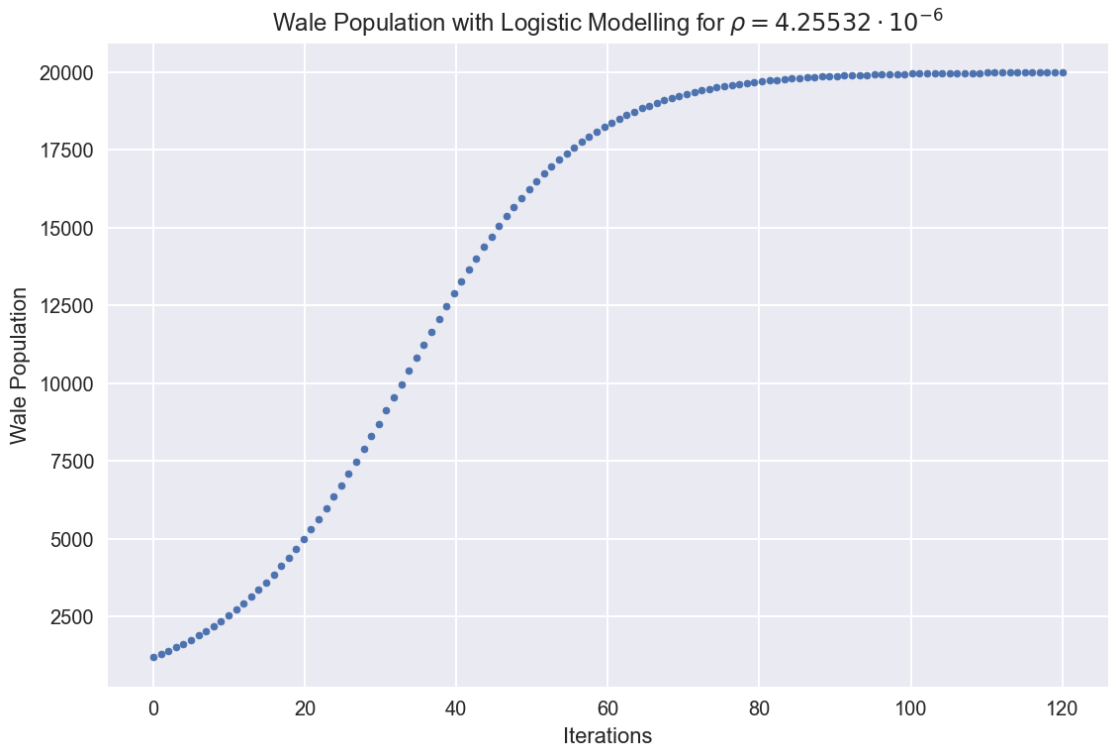
Nun iterieren wir mit den oben genannten Werten, nehmen aber diesmal einen Scatterplot, der die einzelnen Punkte nicht verbindet. Ihr seht im Folgenden wieso. Setzen wir zuerst die Iterationsgleichung als Funktion an:

[4]:
```
def walepopfree(W0=1200, K=20_000, rho=4.25532*10**(-6), n=100):
    # initialize iteration list
    walepop = []
    walepop.append(W0)
    for k in range(n+1):
        walepop.append(walepop[k] + rho * (K - walepop[k]) * walepop[k])
    return walepop
```

Und plotten damit eine generierte Prognose mit 120 Einträgen.

[5]:
```
walepop=walepopfree(1200, 20_000, numrho, 120)
```

[6]:
```
%matplotlib inline
#%matplotlib notebook
%config InlineBackend.figure_format = 'retina'

import matplotlib.pyplot as plt
from numpy import linspace

plt.style.use('seaborn')

x = linspace(0, 120, 122)
plt.scatter(x, walepop, marker='.')
plt.xlabel('Iterations')
plt.ylabel('Wale Population')
plt.title(r'Wale Population with Logistic Modelling for $\rho=4.25532\cdot␣
 ↪10^{-6}$')

plt.tight_layout()

plt.show()
#plt.savefig('plot.svg')
```

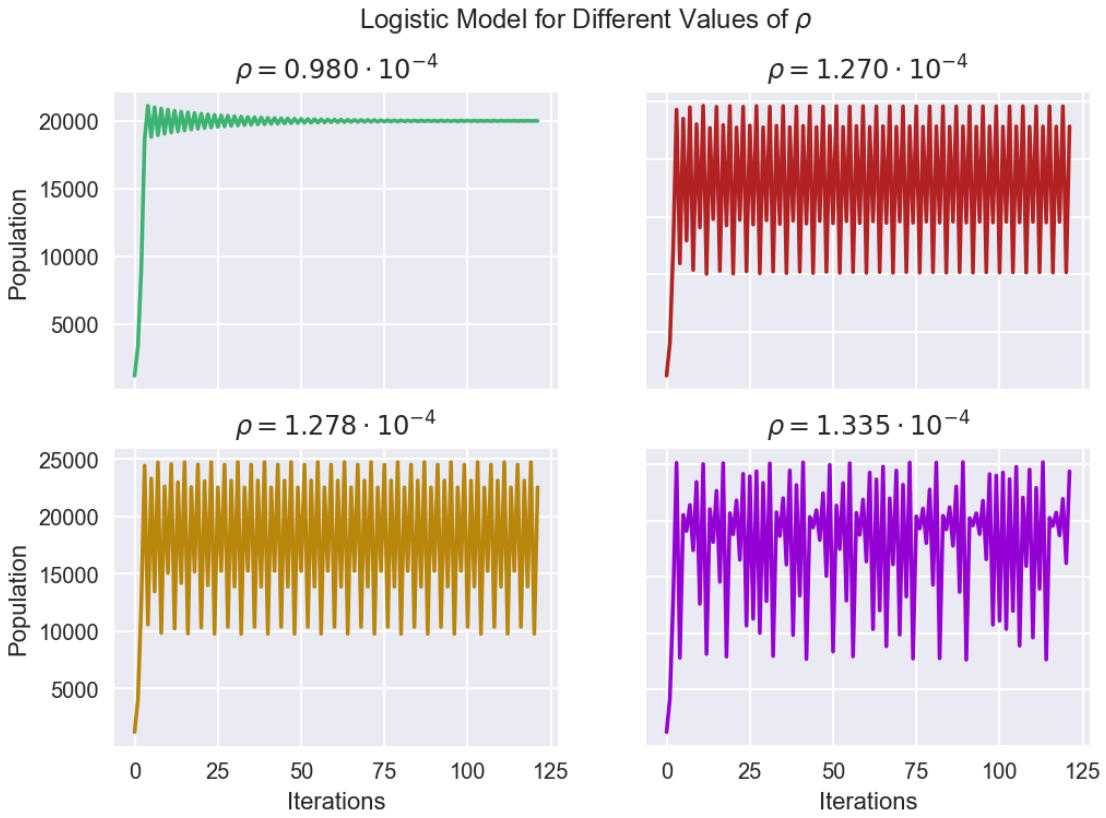


Mit dem oben aus dem Kontext berechneten ρ ergibt sich was wir erwarten: eine logstische Wachstumskurve mit Startwert 1200 und Kapazitätsgrenze K. Jetzt aber variieren statt wie im vorangegangenen Modul die Startwerte W_0 oder die Fangzahlen/-quoten den Parameter ρ; und staunen:

[7]:
```
walepop1=walepopfree(1200, 20_000, 0.980*10**(-4), 120)
walepop2=walepopfree(1200, 20_000, 1.270*10**(-4), 120)
walepop3=walepopfree(1200, 20_000, 1.278*10**(-4), 120)
walepop4=walepopfree(1200, 20_000, 1.335*10**(-4), 120)
```

[8]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(walepop1, color='mediumseagreen', label='empfänglich')
axs[0, 0].set_title(r'$\rho=0.980\cdot 10^{-4}$')
axs[0, 1].plot(walepop2, color='firebrick')
axs[0, 1].set_title(r'$\rho=1.270\cdot 10^{-4}$')
axs[1, 0].plot(walepop3, color='darkgoldenrod')
axs[1, 0].set_title(r'$\rho=1.278\cdot 10^{-4}$')
axs[1, 1].plot(walepop4, color='darkviolet')
axs[1, 1].set_title(r'$\rho=1.335\cdot 10^{-4}$')

fig.suptitle(r'Logistic Model for Different Values of $\rho$')
#l1, = ax1.plot(t, E, color='mediumseagreen', label='empfänglich')
```

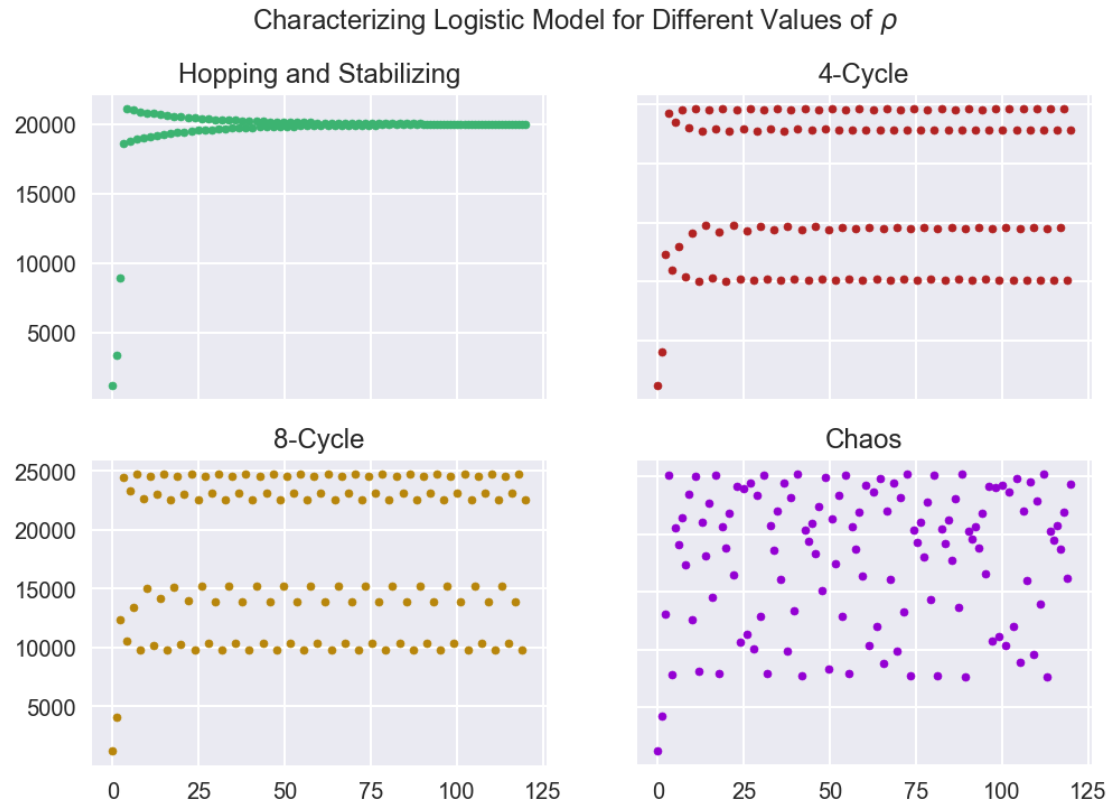
```
#fig.legend()
#plt.xlabel('Tage')
#plt.ylabel('Kinder')
#plt.show()

for ax in axs.flat:
    ax.set(xlabel='Iterations', ylabel='Population')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```



Hier sieht man bloss, dass die Punkte mit fort laufender Iteration "nervös" hin und her springen. Gut ersichtlich ist der zeitliche Verlauf. Um aber die "Punkteverteilung" besser zu sehen, stellen wir auf Scatterplot um, welcher die Punkte nicht verbindet. In der Tat handelt es sich ja auch um einzelne, berechnete Punkte W_k, und nicht um stetige Funktionen.

[9]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(x, walepop1, marker='.', color='mediumseagreen',␣
 ↪label='empfänglich')
axs[0, 0].set_title(r'Hopping and Stabilizing')
```

```
axs[0, 1].scatter(x, walepop2, marker='.', color='firebrick')
axs[0, 1].set_title(r'4-Cycle')
axs[1, 0].scatter(x, walepop3, marker='.', color='darkgoldenrod')
axs[1, 0].set_title(r'8-Cycle')
axs[1, 1].scatter(x, walepop4, marker='.', color='darkviolet')
axs[1, 1].set_title(r'Chaos')

fig.suptitle(r'Characterizing Logistic Model for Different Values of $\rho$')
#l1, = ax1.plot(t, E, color='mediumseagreen', label='empfänglich')


#fig.legend()
#plt.xlabel('Tage')
#plt.ylabel('Kinder')
#plt.show()

#for ax in axs.flat:
    #ax.set(xlabel='Iterations', ylabel='Population')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```



1.2 Analyse

1.2.1 Funktionenschreibweise

Wir wollen für weitere Analysen nun weg von der iterativen Schreibweise hin zur Funktionenschreibweise. Dazu wird die Iteration linear skaliert und normiert, so dass nur noch ein Parameter übrig bleibt.

Übung : Gehe von der Iteration der Wale

$$W_{k+1} = W_k + \rho W_k(K - W_k)$$

aus.

a) Zeige, dass daraus $W_{k+1} = (1 + \rho K)W_k - \rho W_k^2$ folgt.

b) Setze $r := 1 + \rho K$ und $x_k := \frac{\rho}{r} W_k$ und zeige, dass daraus die mit dem Parameter r normierte Iterationsgleichung

$$x_{k+1} = r \cdot x_k(1 - x_k)$$

folgt.

In diesem Video, Wale zu logistische Funktion, wird der Inhalt obiger Übung noch einmal erläutert.

1.2.2 Die logistische Funktion

Wir arbeiten also jetzt mit der zugehörigen Funktion

$$f_r(x) = rx(1 - x),$$

welche ich **logistische Funktion** nenne. Diese quadratische Funktion ist grundsätzlich auf ganz $\mathbb{R}$ definiert. Jetzt wird sich das Geschehen eine Weile um diese Funktion f_r drehen.

Übung : Berechne die beiden Nullstellen von f_r und beachte, dass diese unabhängig vom Parameter r sind.

Damit die Funktion "handlicher" wird, schränken wir erstmal den Definitionsbereich $\mathbb{D}$ der freien Variablen x ein, was aber de facto wegen der Isomorphie $\mathbb{R} \simeq (0, 1)$ keiner Einschränkung im mathematischen Sinn gleich kommt. Wählen wir also $\mathbb{D} := [0, 1]$. Nun möchten wir die Wertemenge ebenfalls $\mathbb{W} = [0, 1]$ haben.

Übung : Ein *Isomorphismus* ist vereinfacht gesagt eine bijektive Abbildung zwischen zwei Mengen. Findest du einen Isomorphismus $\varphi : \mathbb{R} \to (0, 1)$? Kannst du auch die Inversfunktion φ^{-1} angeben?

Übung : Wieso wollen wir $\mathbb{W} = [0, 1]$?

Übung : Für welche Werte von r gilt sicher $\mathbb{W} = [0, 1]$?

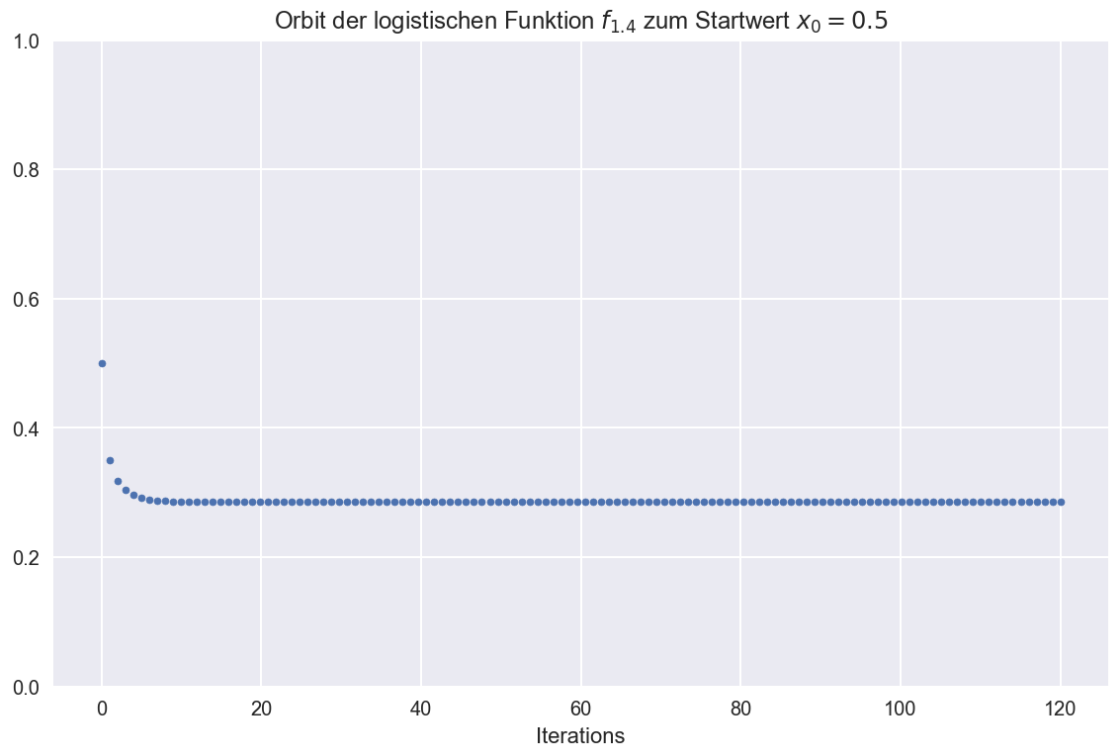
Wiederum ist es an der Zeit für etwas Action. Lassen wir einige Orbits zu ausgewählten rs plotten. Wir installieren die logistische Funktion f_r:

[10]:
```
def fr(x,r):
    return r*x*(1-x)
```

und schreiben eine Funktion, die uns für die logistische Funktion f_r mit Startwert x_0 einen Orbit generiert.

[11]:
```
def frorbit(x0=0.5, r=1.4, n=120):
    orbit = []
    orbit.append(x0)
    for k in range(n+1):
        orbit.append(fr(orbit[k],r))
    return orbit
```

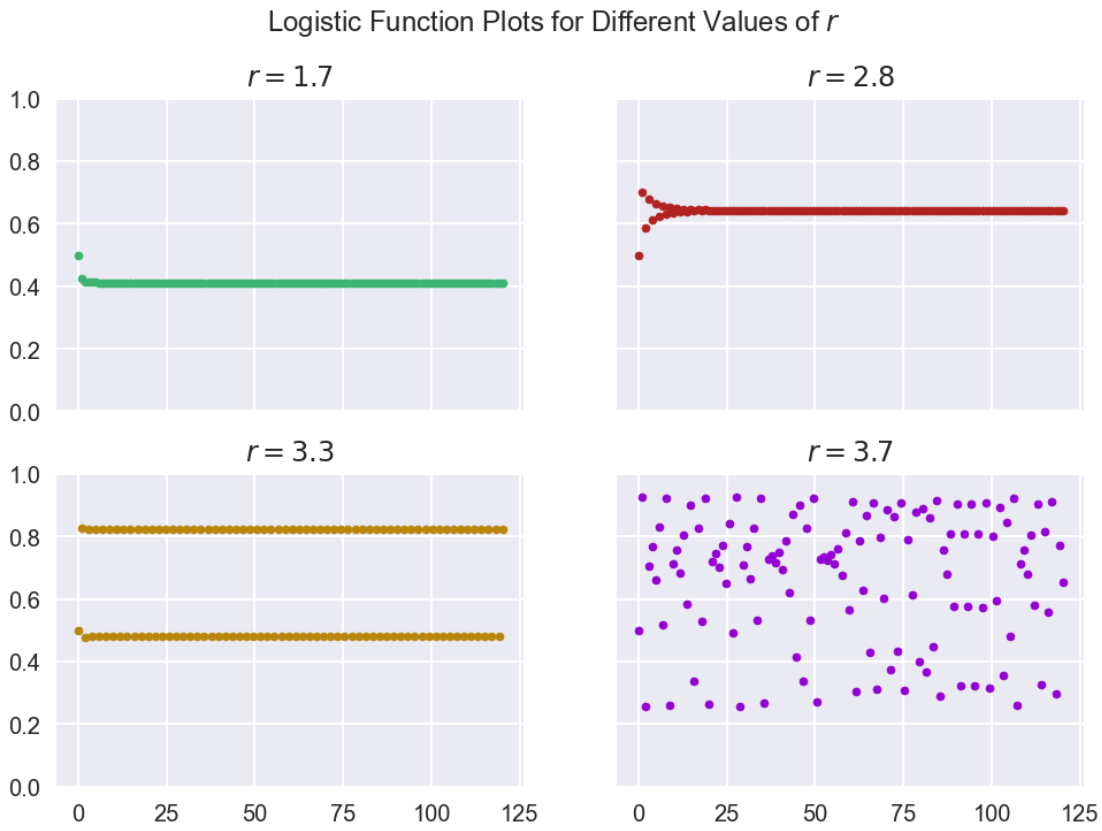
[12]:
```
orb = frorbit()

plt.scatter(x, orb, marker='.')
plt.xlabel('Iterations')
plt.title(r'Orbit der logistischen Funktion $f_{1.4}$ zum Startwert $x_0=0.5$')

plt.ylim(0,1)
plt.tight_layout()
plt.show()
```



Da die Funktion mit $\mathbb{D} = [0, 1]$ für $0 < r < 4$ uneingeschränkt iteriert werden kann, schauen wir uns ein paar Beispiele an. Unten sind die Plots für $r = 1.7, 2.8, 3.3, 3.7$. Man vergleiche mit dem Walmodell.

[13]:
```
orbit1 = frorbit(0.5, 1.7, 120)
orbit2 = frorbit(0.5, 2.8, 120)
orbit3 = frorbit(0.5, 3.3, 120)
orbit4 = frorbit(0.5, 3.7, 120)
```

[14]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(x, orbit1, marker='.', color='mediumseagreen',␣
 ↪label='empfänglich')
axs[0, 0].set_title(r'$r=1.7$')
axs[0, 1].scatter(x, orbit2, marker='.', color='firebrick')
axs[0, 1].set_title(r'$r=2.8$')
axs[1, 0].scatter(x, orbit3, marker='.', color='darkgoldenrod')
axs[1, 0].set_title(r'$r=3.3$')
axs[1, 1].scatter(x, orbit4, marker='.', color='darkviolet')
axs[1, 1].set_title(r'$r=3.7$')

fig.suptitle(r'Logistic Function Plots for Different Values of $r$')

axs[0, 0].set_ylim([0, 1])
axs[0, 1].set_ylim([0, 1])
axs[1, 0].set_ylim([0, 1])
axs[1, 1].set_ylim([0, 1])

for ax in axs.flat:
    ax.label_outer()
```



Übung : Welchen r-Werten entsprächen obige Plots den im Walmodell gezeigten Plots?

1.2.3 Fixpunkte von f_r

Übung : Bestimme die Fixpunkte der logistischen Funktion

$$f_r(x) = rx(1-x).$$

Wir interessieren uns in Kürze über Attraktivität dieser; vorerst noch etwas Geduld.

1.2.4 Werte von r

0<r<1 Ich verwende die Funktionen- und die Iterationsschreibweise grad passend für meine Argumentationslinie, welche aber in jedem Fall mit beiden Ansätzen vollzogen werden kann. Dieser Fall, wenn $0 < r < 1$ ist, lässt sich bequem handhaben. Man kann zudem $0 \leq r < 1$ nehmen, da die Fixpunktbedingung über diesen Bereich gelten. Die Iterationsfolge konvergiert für jeden Startwert x_0, da man x_k nach oben abschätzen kann. Betrachte:

$$x_k = rx_{k-1}(1-x_{k-1}) < rx_{k-1} < r^2x_{k-2} < \cdots < r^kx_0$$

was wegen $0 \leq r < 1$ klar gegen 0 geht:

$$\lim_{k\to\infty} r^k x_0 = 0.$$

Also konvergiert die Iterationsfolge für diese r für jeden Startwert x_0 gegen den Fixpunkt 0.

Übung : Bestimme die Bereiche für r, für welche die beiden Fixpunkte (nota bene 1. Ordnung) attaktiv sind.

1<r<3 Man könnte jetzt unmittelbar weiterstöbern. Ich möchte aber kurz einen Einschub vorziehen, den wir eigentlich bereits zu Beginn absolviert haben. Also entweder als Recap oder jetzt zum Ersten...

1.2.5 Fixpunktlemma

Es gilt:

> Seien $f : \mathbb{D} \to \mathbb{W}$ mit $\mathbb{W} \subset \mathbb{D} \subset \mathbb{R}$, sowie $x_0 \in \mathbb{D}$ und $x_k = f(x_{k-1}) \quad \forall k \in \mathbb{N}$. Wenn die Iterationsfolge $< x_k >$ gegen x_p konvergiert, dann ist x_p ein Fixpunkt von f.

Der Beweis ist nahezu trivial, wie es auch der Satz erscheint. Konvergiere $< x_k >$ gegen x_p, d.h. $\lim_{k\to\infty} x_k = x_p$, dann folgt

$$x_p = \lim_{k\to\infty} x_k = \lim_{k\to\infty} f(x_{k-1}) = f(x_p).$$

So trivial der Satz scheint, jedoch gilt die Umkehrung im Allgemeinen nicht!

Übung : Finde ein Gegenbeispiel, in dem die Umkehrung nicht gilt.

Da wir uns aber für die Umkehrung — oder zumindest eine abgeschwächte Form davon — interessieren, sind wir etwas vorsichtig. Wir formulieren folgenden Satz.

> Sei $I \subset \mathbb{D}$ ein Intervall und f habe in I genau einen Fixpunkt x_p. Ferner gebe es ein M mit $0 < M < 1$ für das $|f'(x)| < M \quad \forall x \in I$.
> Dann gilt für jede Iteration mit Startwert $x_0 \in I$:
> Die Folge $< x_k >$ konvergiert und der Grenzwert ist ihr Fixpunkt x_p.

Der Beweis wurde bereits zu Beginn der Iterationen einmal illustiert.

Übung : Recap des Beweises. Er setzt sich aus dem Fixpunktsatz von Hahn-Banach mit Hilfe des Mittelwertsatzes zusammen.

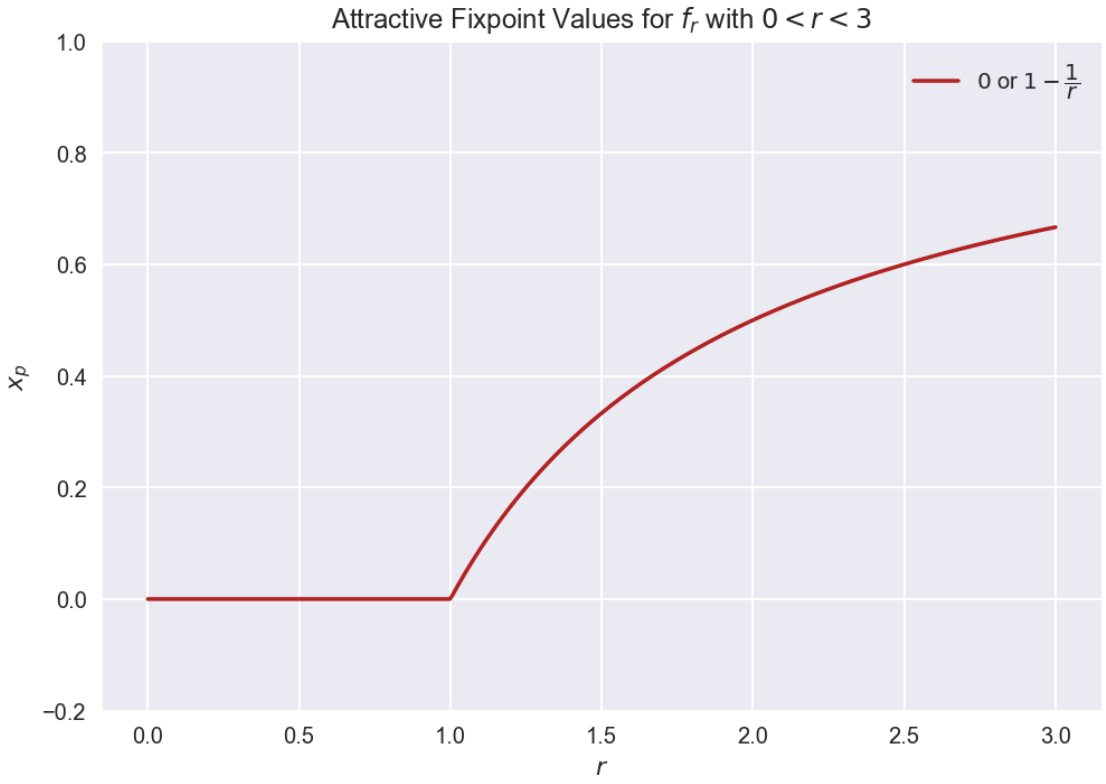
Wenn wir uns für Startwerte x_0 interessieren, welche anziehend sind, für welchen Bereich lassen wir dann Startwerte zu, wenn wir den obigen Satz berücksichtigen?

Übung : Zeige, dass wir die Startwerte im Bereich $(\frac{r-1}{2r}, \frac{r+1}{2r})$ haben wollen.

1.2.6 Attraktorwerte von f_r

Übung : Bestätige, dass der Graph der attraktiven Fixpunkte der logistischen Funktion f_r für die Parameterwerte $0 < r < 3$ wie folgt aussehen muss:

[15]:
```
import numpy as np

xl = np.linspace(0, 3, 500)
y = np.piecewise(xl, [xl < 1, xl >= 1], [lambda xl: 0, lambda xl: 1-1/xl])

fig = plt.figure()
plt.plot(xl,y, color='firebrick', label=r'$0$ or $1-\dfrac{1}{r}$')
plt.ylim(-0.2,1)
plt.title(r"Attractive Fixpoint Values for $f_r$ with $0<r<3$")
plt.legend()
plt.xlabel(r'$r$')
plt.ylabel(r'$x_p$')
plt.show()
```



Wir haben also noch durch obigen Satz so was wie ein “sicheres” Attraktorintervall

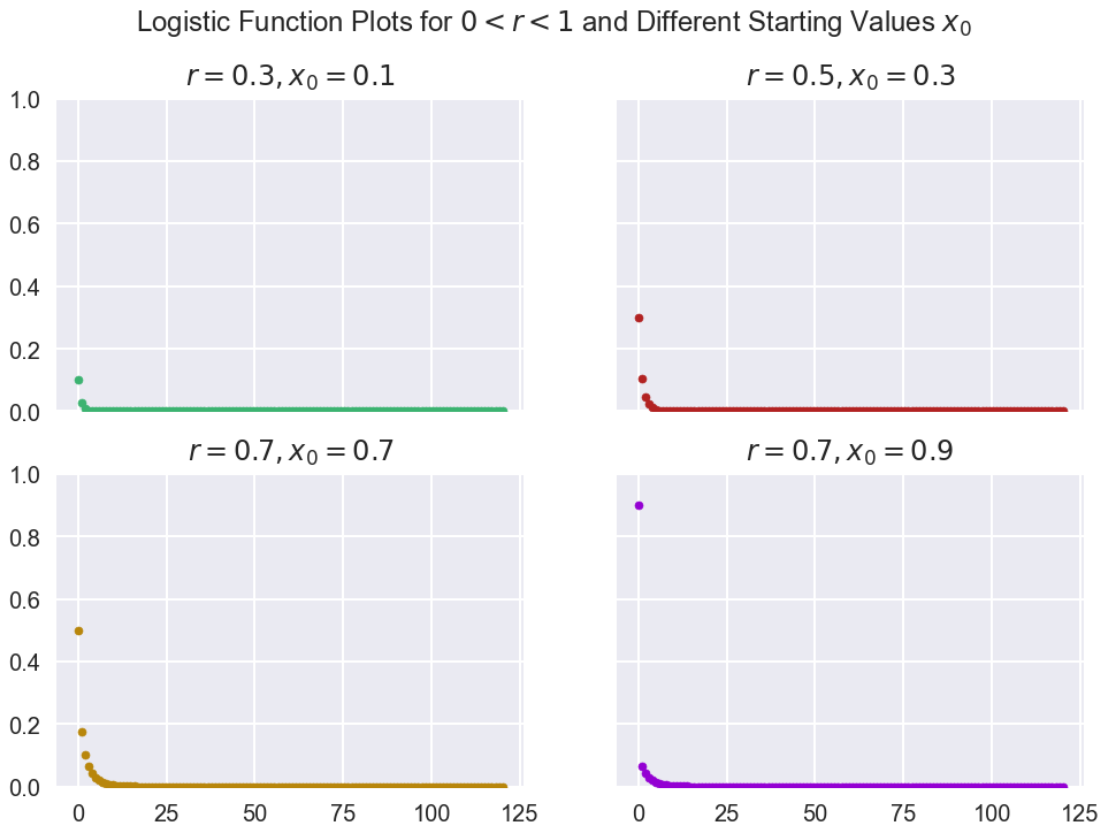
$$I_r := \left(\frac{1}{2} - \frac{1}{2r}, \frac{1}{2} + \frac{1}{2r}\right)$$

erhalten.

Betrachtet man für $0 < r < 1$ den attraktiven Fixpunkt $x_p = 0$, so konvergiert wegen $[0, 1] \subset I_r$ die Iterationsfolge für jeden Startwert x_0 gegen 0. Die Population stirbt immer aus.

[16]:
```
orbit5 = frorbit(0.1, 0.3, 120)
orbit6 = frorbit(0.3, 0.5, 120)
orbit7 = frorbit(0.5, 0.7, 120)
orbit8 = frorbit(0.9, 0.7, 120)
```

[17]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(x, orbit5, marker='.', color='mediumseagreen',␣
 ↪label='empfänglich')
axs[0, 0].set_title(r'$r=0.3, x_0=0.1$')
axs[0, 1].scatter(x, orbit6, marker='.', color='firebrick')
axs[0, 1].set_title(r'$r=0.5, x_0=0.3$')
axs[1, 0].scatter(x, orbit7, marker='.', color='darkgoldenrod')
axs[1, 0].set_title(r'$r=0.7, x_0=0.7$')
axs[1, 1].scatter(x, orbit8, marker='.', color='darkviolet')
axs[1, 1].set_title(r'$r=0.7, x_0=0.9$')

fig.suptitle(r'Logistic Function Plots for $0<r<1$ and Different Starting Values␣
 ↪$x_0$')

axs[0, 0].set_ylim([0, 1])
axs[0, 1].set_ylim([0, 1])
axs[1, 0].set_ylim([0, 1])
axs[1, 1].set_ylim([0, 1])

for ax in axs.flat:
    ax.label_outer()
```



Erstaunlich, wie rasch diese Folgen jeweils gegen 0 gehen!
Interessanter wird es jetzt für $1 < r < 3$, da dort der Wert des Fixpunktes für jedes r via $x_p = 1 - \frac{1}{r}$ berechnet wird; also individuell ist. Ich beziehe mich auf das Beispiel $r = 1.4$ weiter oben, von dem wir bereits einen Plot zum Startwert 0.5 haben. Für diesen Fall finden wir also den Fixpunkt bei

[18]:
```
from IPython.display import Latex, display, Math

fpvalue = 1-1/1.4
Latex("$x_p$ = %0.5e " % fpvalue)
```

[18]:
x_p = 2.85714e-01

[19]:
```
%%latex
\normalfont
```

Für das attraktive Startwertintervall rechnen wir

[20]:
```
rvalue = 1.4
dummy = 1/(2*rvalue)
starting = 1/2 - dummy
ending = 1/2 + dummy
```

```
print(rf'I_n = (',starting,',',ending,')')
```

```
I_n = ( 0.14285714285714285 , 0.8571428571428572 )
```

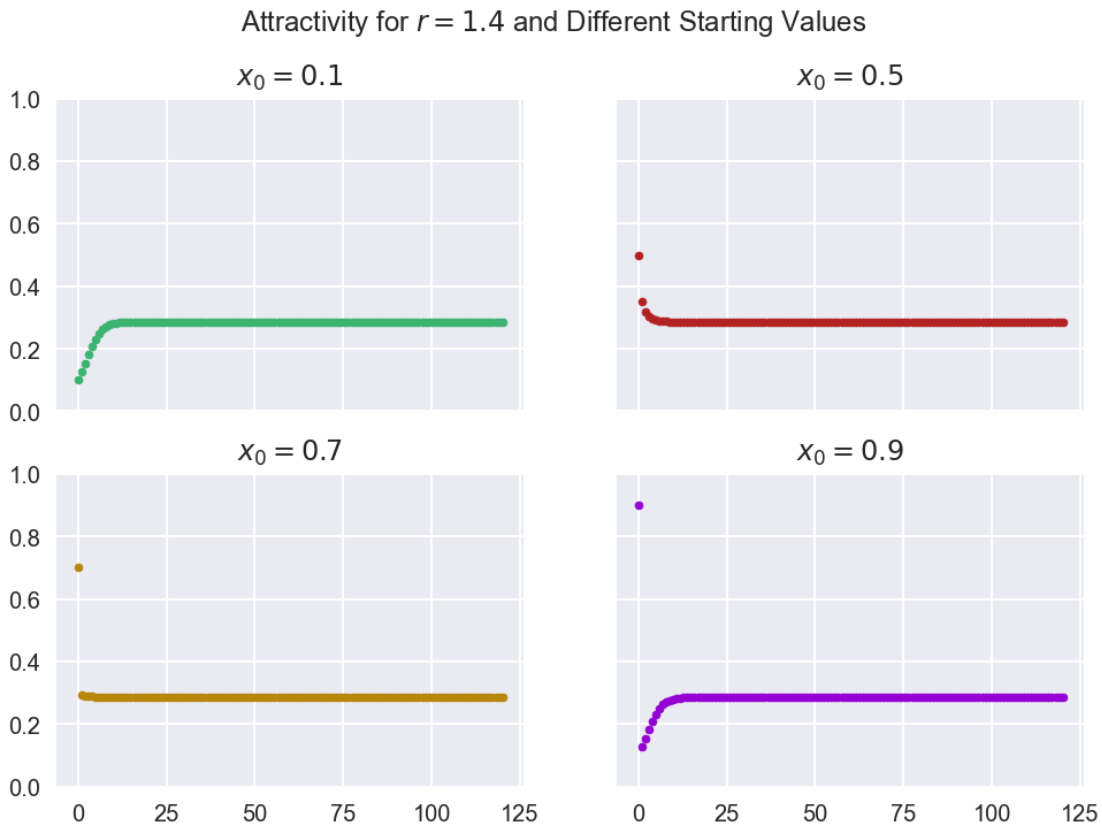
[21]:
```
from sympy import *
display(Math("I_n = (" + latex(starting) + "," + latex(ending) + ")"))
```

$I_n = (0.14285714285714285, 0.8571428571428572)$

[22]:
```
%%latex
\normalfont
```

[23]:
```
orbit9 = frorbit(0.1, 1.4, 120)
orbit10 = frorbit(0.5, 1.4, 120)
orbit11 = frorbit(0.7, 1.4, 120)
orbit12 = frorbit(0.9, 1.4, 120)
```

[24]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(x, orbit9, marker='.', color='mediumseagreen',␣
 ↪label='empfänglich')
axs[0, 0].set_title(r'$x_0=0.1$')
axs[0, 1].scatter(x, orbit10, marker='.', color='firebrick')
axs[0, 1].set_title(r'$x_0=0.5$')
axs[1, 0].scatter(x, orbit11, marker='.', color='darkgoldenrod')
axs[1, 0].set_title(r'$x_0=0.7$')
axs[1, 1].scatter(x, orbit12, marker='.', color='darkviolet')
axs[1, 1].set_title(r'$x_0=0.9$')

fig.suptitle(r'Attractivity for $r=1.4$ and Different Starting Values')

axs[0, 0].set_ylim([0, 1])
axs[0, 1].set_ylim([0, 1])
axs[1, 0].set_ylim([0, 1])
axs[1, 1].set_ylim([0, 1])

for ax in axs.flat:
    ax.label_outer()
```



Bemerke: Zweimal sind wir ausserhalb des sicheren Attroktorbereichs gestartet, trotzdem sind wir beim Fixpunkt gelandet. Das bedeutet, dass sich vielleicht dieser Attraktorbereich noch weiter fassen lässt.

1.2.7 Was passiert für $r > 3$?

Klar ist, dass für $r > 3$ die beiden Fixpunkte nicht mehr attraktiv sind; sie sind aber natürlich nach wie vor "da". Wenn man oben die Walplots anschaut, so legt $\rho = 0.98 \cdot 10^{-4}$ den Verdacht nahe, dass Fixpunkte zweiter Ordnung existieren könnten. Daher lösen wir folgende

Übung : Suche Fixpunkte zweiter Ordnung. Verwende Polynomdivision, um die Fixpunkte erster Ordnung aus der Bedingung "rauszudividieren" und so die "reinen" Fixpunkte der Ordnung 2 zu erhalten.

Wir kriegen also nebst $x_{p1} = 0$ und $x_{p2} = 1 - \frac{1}{r}$ nun zusätzlich

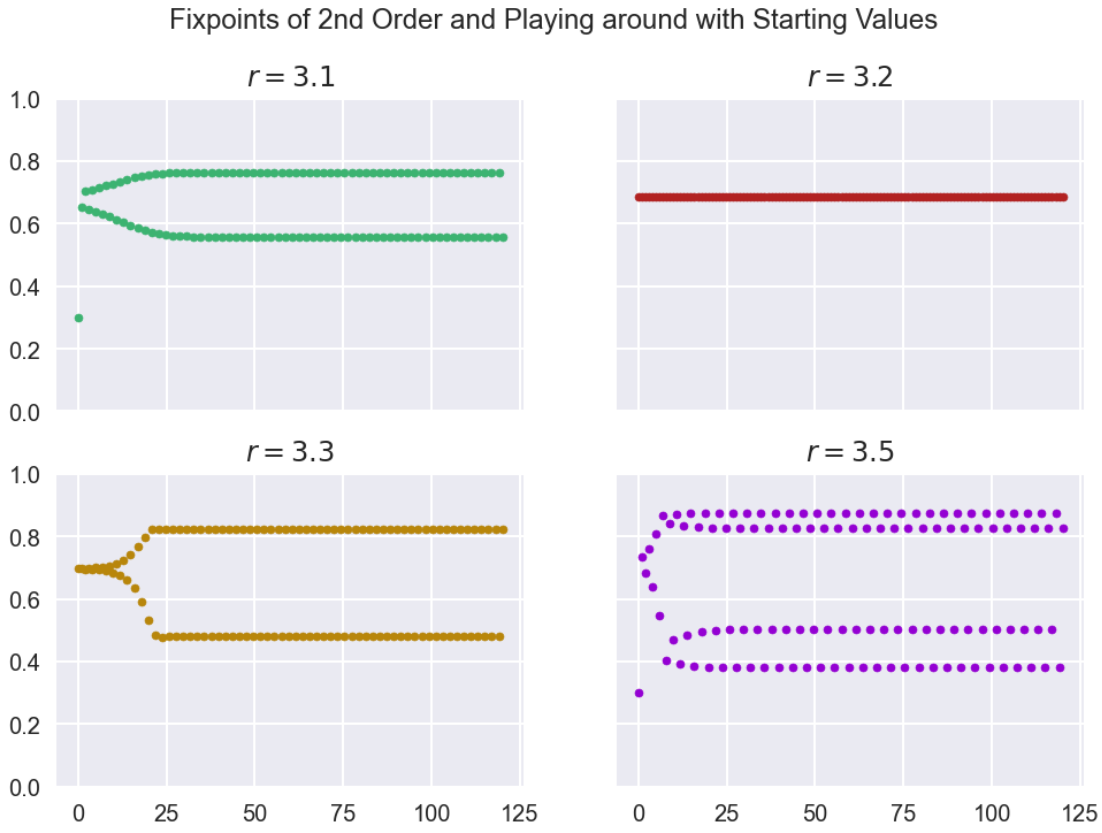
$$x_{p3,p4} = \frac{(r+1) \pm \sqrt{(r+1)(r-3)}}{2r}$$

was zusätzlich die Frage aufwirft

Übung : Gibt es einen Bereich für r, in dem diese Fixpunkte der Ordnung 2 stabil sind? Vermutlich ja, wenn man die folgenden Plots anschaut. Dabei habe ich mit den Startwerten x_0 gespielt.

Im Video Fixpunkte von f_r der Periode 2 wird alles erläutert.

[25]:
```
orbit13 = frorbit(0.3, 3.1, 120)
orbit14 = frorbit(1-1/3.2, 3.2, 120)
orbit15 = frorbit(1-1/3.29, 3.3, 120)
orbit16 = frorbit(0.3, 3.5, 120)

fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(x, orbit13, marker='.', color='mediumseagreen',␣
 ↪label='empfänglich')
axs[0, 0].set_title(r'$r=3.1$')
axs[0, 1].scatter(x, orbit14, marker='.', color='firebrick')
axs[0, 1].set_title(r'$r=3.2$')
axs[1, 0].scatter(x, orbit15, marker='.', color='darkgoldenrod')
axs[1, 0].set_title(r'$r=3.3$')
axs[1, 1].scatter(x, orbit16, marker='.', color='darkviolet')
axs[1, 1].set_title(r'$r=3.5$')

fig.suptitle(r'Fixpoints of 2nd Order and Playing around with Starting Values')

axs[0, 0].set_ylim([0, 1])
axs[0, 1].set_ylim([0, 1])
axs[1, 0].set_ylim([0, 1])
axs[1, 1].set_ylim([0, 1])

for ax in axs.flat:
    ax.label_outer()
```



Übung : Rechne die Fixpunkt-Werte nach.

Übung : Finde den Attraktivitätsbereich für r für die Fixpunkte zweiter Ordnung.

Weitere Fixpunkte sind nicht mehr lustig zum Rechnen. Man stellt rasch fest, dass der Aufwand von Hand höhere Fixpunkte auszurechnen ins Astronomische steigt.

Ich möchte vor der Fortsetzung noch ein schönes Resultat von `Nagashima/Baba` aus dem Jahr 1999 vorstellen. Dazu leite ich wieder durch Übungen...

Übung : Zeige: Seien x_1 und x_2 zwei Fixpunkte der Ordnung 2 von f mit $f(x_1) = x_2$ und f gutmütig. Dann gilt für die Ableitung $(f(f(x_1)))' = (f(f(x_2)))'$.

Übung : Mit einer Induktion zeige man nun, dass analog zu oben dieselbe Aussage für Fixpunkte der Periode k gilt. Das heisst: $>$ Alle Fixpunkte eines k Orbits haben dieselbe Steigung, werden also alle gleichzeitg stabil oder instabil.

Übung : Zeige: Wendet man obiges Ergebnis auf die logistische Funktion für die beiden Fixpunkte der Ordnung 2 an, so folgt Attraktivität für $3 < r < 1 + \sqrt{6}$.

Von den Walen zur logistischen Funktion