**Sourcecode der Seminararbeit**

**Zwischen Ordnung und Chaos:**

**Das komplexe dynamische Verhalten nichtlinearer Gleichungen**

*# Sourcecode zu Seminararbeit: Zwischen Ordnung und Chaos  
# Titel: Logistische Gleichung: Iterationsfolge und Fixpunktdiagramm  
# Verwendet für Abb. 2, 3, 4, 5, 6, 7  
# 2017-09-05: J. Gollreiter***import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
*# 1) Iterationsfolge berechnen und plotten  
  
# Parametereingabe*x = 0.1 *# Startwert*r = 3.01 *# Wachtumsfaktor 1..4*n = 100 *# Anzahl der Iterationen  
  
# Ergebnisliste berechnen*xlist = [x] *# Initialisiere die Ergebnisliste***for** i **in** range(n):  
 x = x \* r \* (1 - x) *# Logistische Gleichung* xlist.append(x) *# Ergebnisliste erweitern* **print**(**"[UI]: x= {}"**.format(x))  
  
*# Ausgabe des Diagramms***print**(xlist)  
  
ilist = np.arange(0, n + 1, 1)  
  
plt.subplot(1,2,1)  
plt.plot(ilist, xlist)  
plt.scatter(ilist, xlist, color = **"red"**, s = 14)  
  
plt.xlabel(**'Iterationsschritt'**)  
plt.ylabel(**'Populationswert'**)  
  
*# 2) Iteration an den Fixpunkt berechnen und plotten  
  
# Parametereingabe*n = 100 *# Anzahl der Samples  
  
# Ergebnisliste berechnen*flist = [] *# Initialisiere die Ergebnisliste***for** i **in** range(n):  
  
 x = float(i)/ float(n)  
 flist.append(r \* x \* (1 - x)) *# Logistische Gleichung als Funktion  
  
 #print("[UI]: xx= {}".format(x))  
  
# Ausgabe des Diagramms  
  
#print(flist)*plt.subplot(1,2,2)  
  
ilist = np.arange(0, n, 1)  
ilist = ilist / float(n)  
plt.plot(ilist, flist)  
plt.scatter(xlist[0:25], xlist[1:26],color = **"red"**)  
plt.scatter(0.03, 0.1, color = **"red"**, s = 35)  
plt.plot(ilist, ilist, color = **"blue"**, linestyle=**":"**)  
  
plt.xlabel(**'x'**)  
plt.ylabel(**'F(x) = xr(1-x)'**)  
plt.show()  
  
plt.show()

*# Sourcecode zu Seminararbeit: Zwischen Ordnung und Chaos  
# Titel: Berechnung und Darstellung der Feigenbaum-Menge  
# Verwendet für Abb. 8, 9, 10  
# 2017-09-06: J. Gollreiter***import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
*# Parametereingabe*n = 100 *# Anzahl der angezeigten Iterationen*m = 1000 *# Anzahl der initialen Iterationen bis zum moegl. Einschwingen  
  
# 1. Iterationsfolge zur log. Gleichung berechnen und plotten***for** r **in** np.arange(1,5,0.001): *#r = Wachtumsfaktor 1..4* x = 0.1 *# Startwert* xlist = [] *# Initialisiere die Ergebnisliste* **for** i **in** range(m):  
 x = x \* r \* (1-x) *# Logistische Gleichung* **for** i **in** range(n):  
 x = x \* r \* (1-x) *# Logistische Gleichung* xlist.append(x) *# Ergebnisliste erweitern; y-Werte Scatter Plot  
  
 # Plotte die Ausgabe* rlist = [] *# x-Werte im Scatter Plot sind die Wachtumsfaktoren* **for** i **in** range(n):  
 rlist.append(r) *# Liste mit Wert r fuellen* plt.scatter(rlist, xlist, c=**'black'**, edgecolors=**'none'**, s=0.1)  
  
*# 2. Einzelne Iterationsergebnisse für den Plot hervorheben***for** r **in** [1, 2, 3.01, 3.5, 4]: *#r = Wachtumsfaktor 1..4* x = 0.1 *# Startwert* xlist = [] *# Initialisiere die Ergebnisliste* **for** i **in** range(m):  
 x = x \* r \* (1-x) *# Logistische Gleichung* **for** i **in** range(n):  
 x = x \* r \* (1-x) *# Logistische Gleichung* xlist.append(x) *# Ergebnisliste erweitern; y-Werte Scatter Plot  
  
 # Plot Ausgabe* rlist = [] *# x-Werte im Scatter Plot sind die Wachtumsfaktoren* **for** i **in** range(n):  
 rlist.append(r) *# Liste mit Wert r fuellen* plt.scatter(rlist, xlist, c=**'red'**, edgecolors=**'none'**, s=30)  
  
*# Beschriftung des Diagramms*plt.xlabel(**'Wachstumsfaktor r'**)  
plt.ylabel(**'Verteilung des Populationswerts'**)  
  
plt.show()

*# Sourcecode zu Seminararbeit: Zwischen Ordnung und Chaos  
# Titel: Orbitplot und Betragsplot zur Iteration der Mandelbrotmenge  
# Verwendet fuer Abb. 12, 13  
# 2017-09-06: J. Gollreiter***import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** math  
  
*# Eingabeparameter waehlen  
  
#c = complex(0.3, 0.5) # Parameter c zur Iteration   
#c = complex(0.4, 0.4)*c = complex(-0.5, 0.2)  
  
z = complex(0, 0) *# Startwert der Iteration  
  
#maxiter = 11*maxiter = 30  
  
reallist = [0]  
imaglist = [0]  
  
*# 1. Mandelbrotmenge berechnen und in Listen einfuegen***for** n **in** range(maxiter):  
 z = z \* z + c *# Mandelbrot-Iterationsformel  
  
 # Fehlerfaelle abfangen* **if** (math.isnan(z.real)) **or** (math.isnan(z.imag)):  
 **break** *# Listen fuellen* reallist.append(z.real)  
 imaglist.append(z.imag)  
 **print**(**"re = "**, z.real, **" im = "**, z.imag)  
 **pass***# 2. Orbitplot erstellen*plt.subplot(1, 2, 1)  
plt.plot(reallist, imaglist, **'r'**, zorder=1, lw=1)  
plt.scatter(reallist, imaglist, edgecolors=**'face'**, zorder=2, c=**'teal'**, s=3)  
plt.xlabel(**'Realteil'**)  
plt.ylabel(**'Imaginaerteil'**)  
plt.title(**'Iterationsfolge von z(n) ("Orbit")'**)  
  
*# 3. Betragsplot erstellen*plt.subplot(1, 2, 2)  
  
ylist = []  
**for** n **in** range(maxiter+1):  
 y = np.sqrt(reallist[n] \* reallist[n] + imaglist[n] \* imaglist[n])  
 ylist.append(y)  
 **print**(**"n="**,n,**" re="**, reallist[n],**" im="**,imaglist[n], **"abs="**, y)  
  
*#print(ylist)  
  
# Ausgabe der Diagramme*ilist = np.arange(0, maxiter+1, 1)  
plt.plot(ilist, ylist, **'r'**, zorder=1, lw=1)  
plt.scatter(ilist,ylist, color = **"blue"**, s=7 )  
plt.xlabel(**'Iteration n'**)  
plt.ylabel(**'Betrag |z|'**)  
plt.title(**'Iterationsfolge von |z(n)|'**)  
  
plt.show()

*# Sourcecode zu Seminararbeit: Zwischen Ordnung und Chaos  
# Titel: Mandelbrotmenge berechnen und plotten  
# Verwendet für Abb. Titelbild, 14, 15  
# 2017-09-05: Erstellt auf Basis des Python-Softwarecodes aus dem Buch:  
# Rashid, Tariq, Make your own Mandelbrot, CreateSpace Independent  
# Publishing Platform, 2014, 1. Auflage, Kapitel Fractals***import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
*#xvalues = np.linspace (-2.25, 0.75, 1000) # Mandelbrot gesamt, Abb 14  
#yvalues = np.linspace (-1.5, 1.5, 2000)  
  
#xvalues = np.linspace (-0.0, 0.6, 1000)   
#yvalues = np.linspace (-0.9, -0.3, 1000)  
  
#xvalues = np.linspace (-0.1, -0.0, 1000)  
#yvalues = np.linspace (-1, -0.5, 1000)  
  
#xvalues = np.linspace (-0.22, -0.21, 1000) # ex3/ Titelbild  
#yvalues = np.linspace (-0.7, -0.69, 1000)  
  
#xvalues = np.linspace (-0.22, -0.215, 1000)  
#yvalues = np.linspace (-0.7, -0.695, 1000)  
  
#xvalues = np.linspace (-0.217, -0.213, 1000)  
#yvalues = np.linspace (-0.697, -0.693, 1000)*xvalues = np.linspace(-0.7496875, -0.7296875, 1000) *# selbstgewaehltes Intervall*yvalues = np.linspace(0.1439375, 0.1639375, 1000)   
  
xlen = len(xvalues)  
ylen = len(yvalues)  
  
*# Funktion der Mandelbrotmenge***def** mandel(c, maxiter): *# maxiter = maximale Wiederholung an Iterationen* z = complex(0, 0)  
 **for** iteration **in** range(maxiter):  
  
 z = (z \* z) + c  
  
 **if** abs(z) > 4:  
 *# return 0 # schwarz / weiss* **break  
 pass  
 pass  
  
 return** iteration  
  
*# Plot der Mandelbrotmenge*atlas = np.empty((xlen, ylen))  
**for** ix **in** range(xlen):  
 **for** iy **in** range(ylen):  
 cx = xvalues[ix]  
 cy = yvalues[iy]  
 c = complex(cx, cy)  
 *#atlas[ix, iy] = mandel(c, 40) # Parameter bzgl. Aufloesung / Pixeldichte   
 # fuer geringen Zoom  
 #atlas[ix, iy] = mandel(c, 120) #* atlas[ix, iy] = mandel(c, 700) *# fuer hohen Zoom / hohe Rechenzeit beachten!!!* **pass  
 pass***#plt.figure(figsize=(20, 20))  
  
#Auswahl der Farbgebung  
  
#plt.imshow(-atlas.T, cmap = "viridis", interpolation="nearest") # farben  
#plt.imshow(-atlas.T, cmap = "terrain", interpolation="nearest") # farben  
#plt.imshow(-atlas.T, cmap = "gnuplot", interpolation="nearest") # farben  
#plt.imshow(-atlas.T, cmap = "jet", interpolation="nearest") # farben*plt.imshow(-atlas.T, cmap = **"nipy\_spectral"**, interpolation=**"nearest"**) *# farben  
# imshow(atlas.T, cmap = "binary",interpolation= "nearest") #schwarz weiss  
  
#Ausgabe des Diagramms*plt.show()