

COMPUTATIONAL PHYSICS

Jona Ackerschott, Julian Mayr

Problem set 8

Problem 1

A plot of the characteristic polynomial for different values of r is given below (see source code 2).

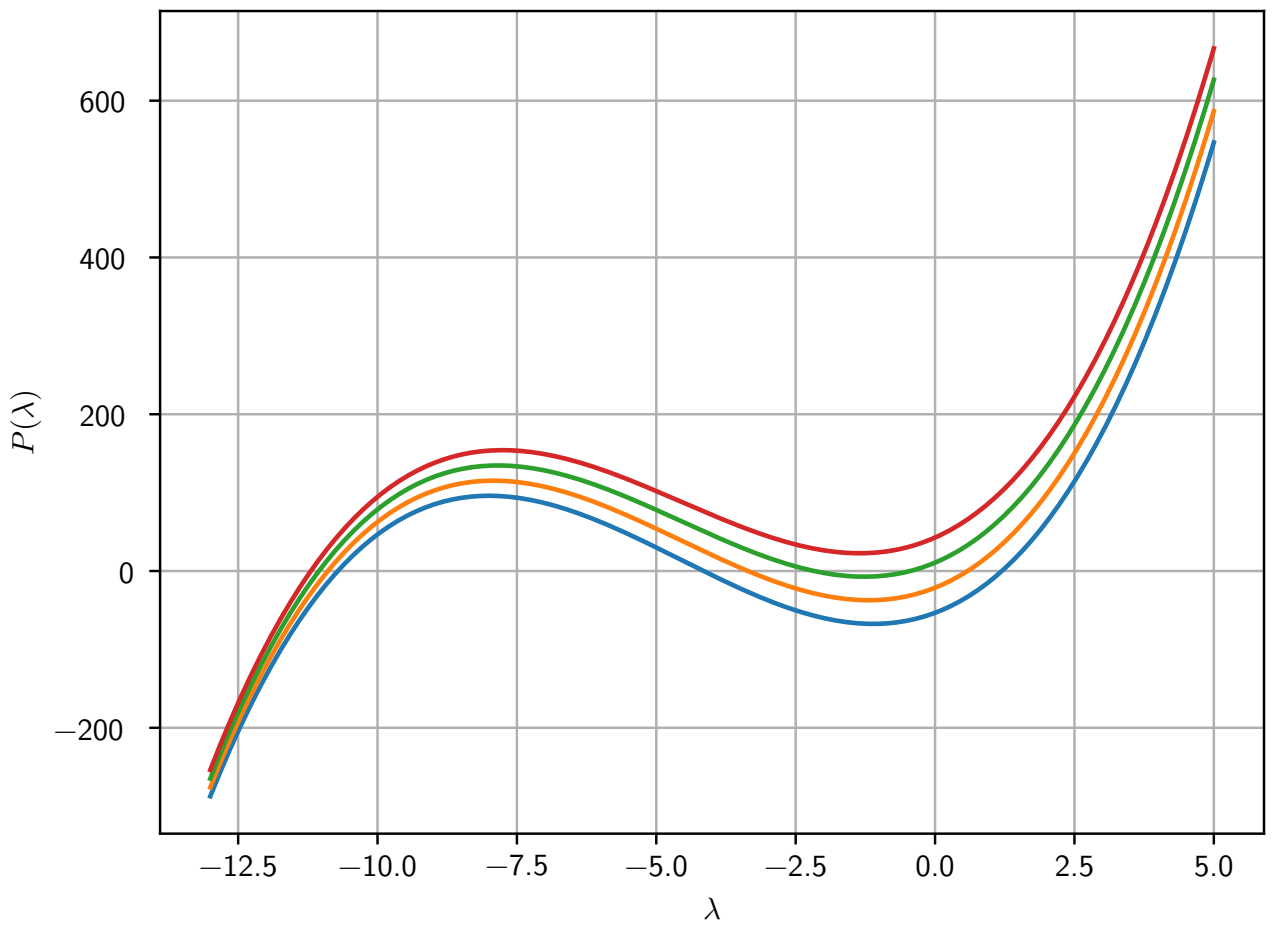


Figure 1: Plots of the characteristic polynomial $P(\lambda)$ for 4 equidistant values for r between 0.0 (blue graph) and 1.8 (red graph)

The zeros of $P(\lambda)$ are computed with the mathematica method `NRoots` as well as exported and plotted in python (see source code 1 and 3). The result is given below
Interpretation of the results is skipped here, due to time constraints.

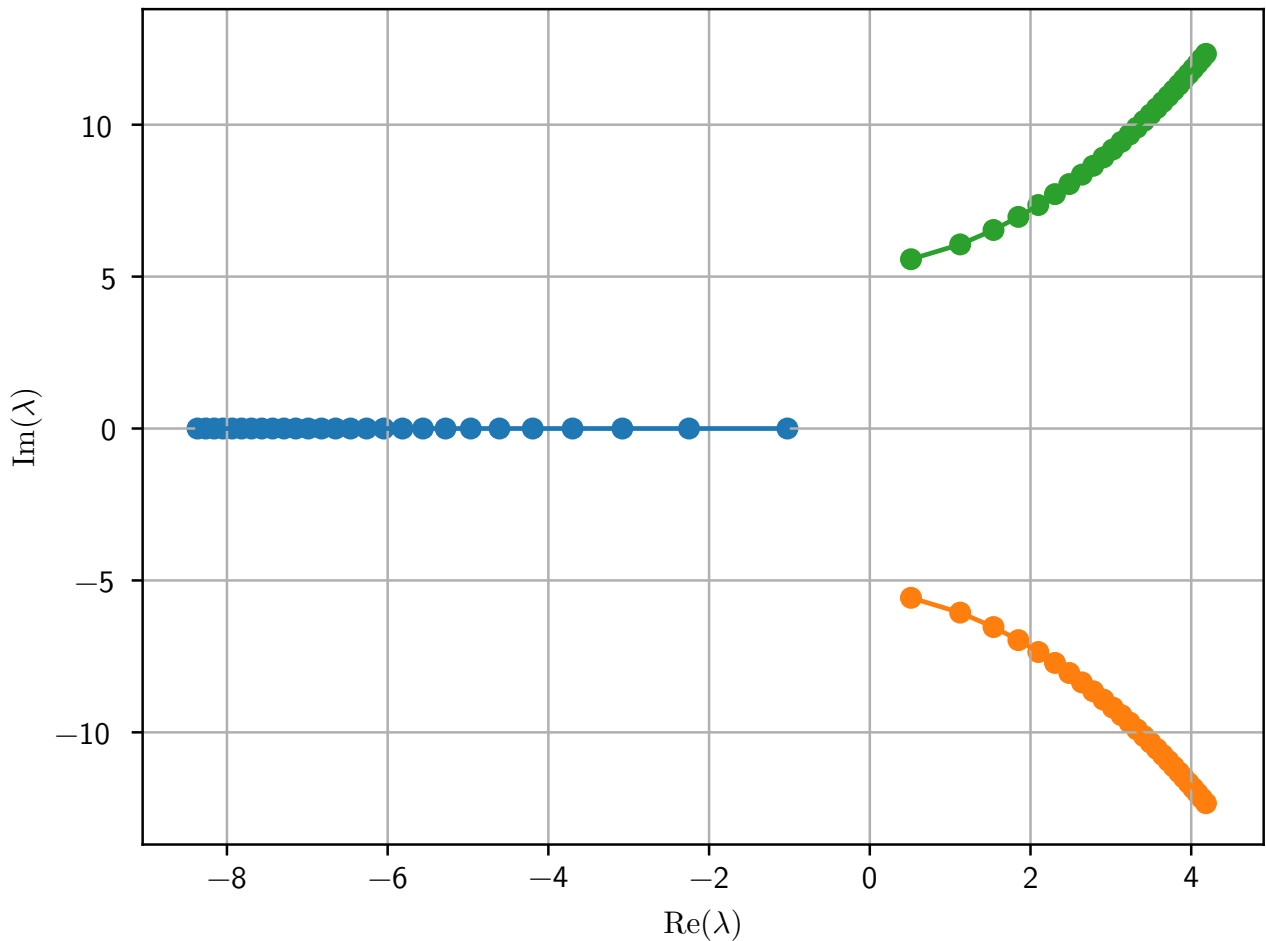


Figure 2: Zeros of the characteristic polynomial $P(\lambda)$, plotted in the complex plane, for r from approximately 1.34561 to 28 in steps of 1

```
In[1]:= sigma=10; b=8/3;
In[2]:= P[x_]:=x^3+(1+b+sigma)+b*(sigma+r)*x+2*sigma*b(r-1)
In[3]:= Zeros=Table[Apply[List,NRoots[P[x]==0,x][[All,2]]],{0,1}],{r,1.34561,28,1}]
In[4]:= Export["zeros.dat",Zeros]
```

Source code 1: Mathematica input for problem 1

```
import matplotlib.pyplot as plt
import numpy as np
import os

sigma = 10
b = 8 / 3

def P(x, r):
    return x**3 + (1 + b + sigma) * x**2 + b * (sigma + r) * x + 2 * sigma * b * (r - 1)

r_vals = np.linspace(0.0, 1.8, 4)

plt.subplots()
plt.xlabel(r'$\lambda$')
plt.ylabel(r'$P(\lambda)$')
```

```

for r in r_vals:
    x = np.linspace(-13, 5, 1000)
    plt.plot(x, P(x, r))

if not os.path.exists('pset8/figures'):
    os.makedirs('pset8/figures')
plt.savefig('pset8/figures/fig1.pgf', bbox_inches='tight', pad_inches=0.0)

```

Source code 2: Python code for problem 1

```

import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.optimize import root

with open('pset8/zeros.dat') as f:
    lines = f.readlines()

zeros = np.zeros((len(lines), 3), dtype=complex)
for i, line in enumerate(lines):
    line = line.replace('*I', 'j')
    zeros_str = np.array(line.split('\t'))
    zeros[i] = zeros_str.astype(complex)

plt.subplots()
plt.xlabel(r'$\mathrm{Re}(\lambda)$')
plt.ylabel(r'$\mathrm{Im}(\lambda)$')
plt.plot(zeros[:,0].real, zeros[:,0].imag)
plt.scatter(zeros[:,0].real, zeros[:,0].imag)
plt.plot(zeros[:,1].real, zeros[:,1].imag)
plt.scatter(zeros[:,1].real, zeros[:,1].imag)
plt.plot(zeros[:,2].real, zeros[:,2].imag)
plt.scatter(zeros[:,2].real, zeros[:,2].imag)

if not os.path.exists('pset8/figures'):
    os.makedirs('pset8/figures')
plt.savefig('pset8/figures/fig2.pgf', bbox_inches='tight', pad_inches=0.0)

```

Source code 3: Python code for problem 1

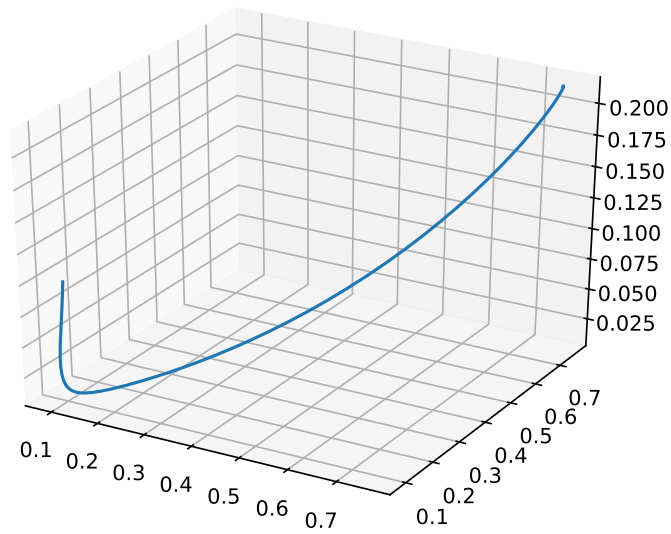


Figure 3: $r = 0.500$

Problem 2

Using the scipy-builtin RK45 solver, the solution of the Lorenz attractor problem was computed numerically for different values of r . The initial condition y_0 was chosen to be $(0.1, 0.1, 0.1) +$ the chosen fixed Point. The solutions were 3d-plotted with matplotlib.

For the stable values of r , the solutions converge oscillating around the nearest fix point (stable solutions). For the higher values, they move around it chaotically without ever converging (chaotic solutions).

Next, for $r=27$ and y_0 as chosen in a), a stable solution, the z coordinate of the local minima in z of the solution were plotted against the previous z value. This plot converges against the Point $(30, 30)$ on the diagonal, so $z = 30$. The slope of the funtion is less than 1, so the point seems to be stable, as the z coordinate also converges.

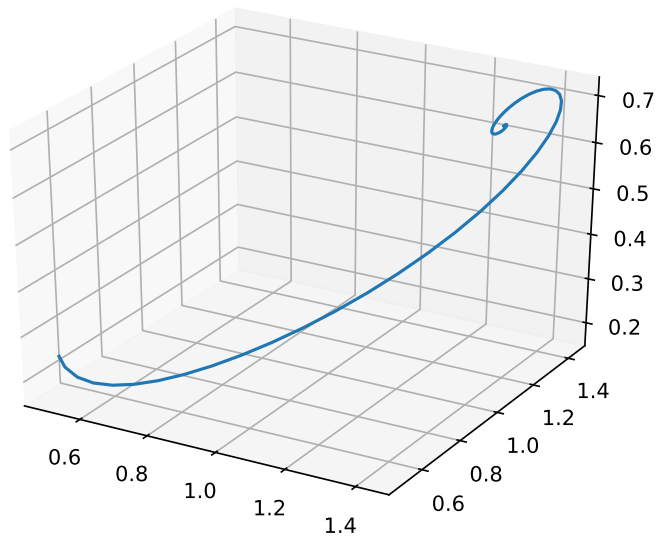


Figure 4: $r = 1.150$

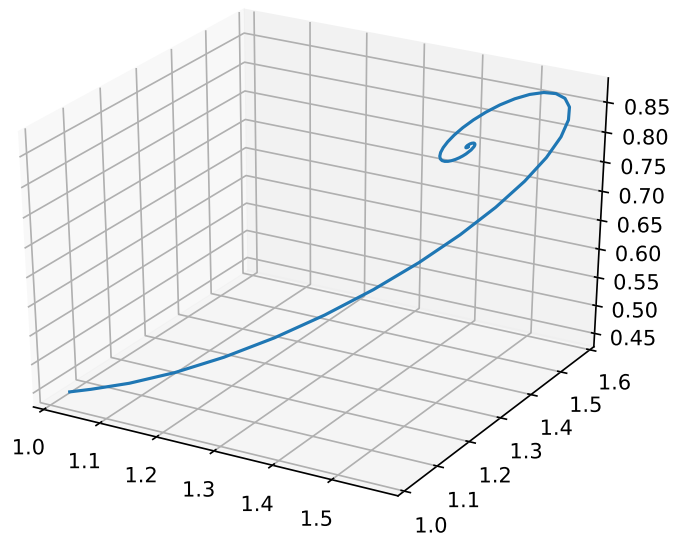


Figure 5: $r = 1.3456$

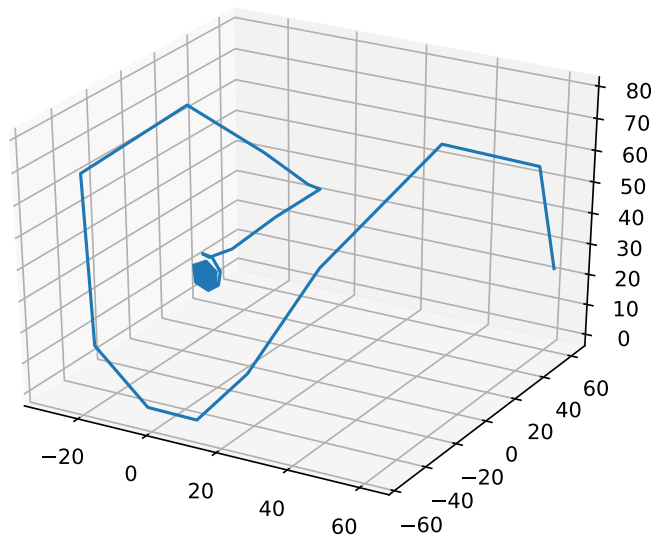


Figure 6: $r = 24.000$

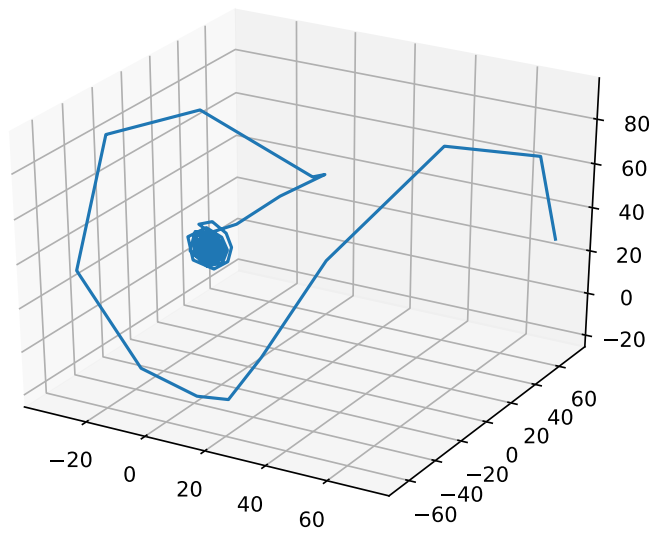


Figure 7: $r = 28.000$

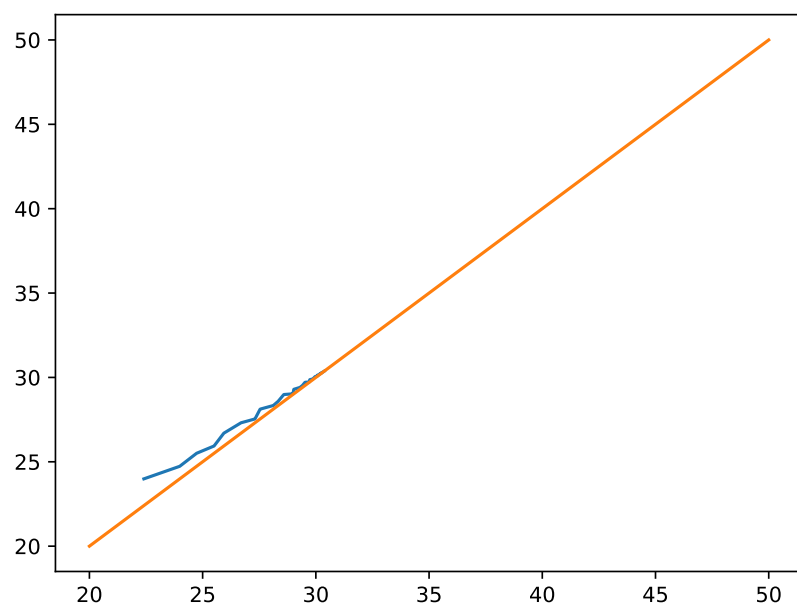


Figure 8: $z_k(z_{k+1})$ plotted from $k = 4$ to $k = 100$

```

import numpy as np
from scipy.integrate import solve_ivp
from scipy import signal
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
SIGMA=10
b=8/3.0

r=0.5
def x_dot(t,y):
    return -SIGMA*(y[0]-y[1])
def y_dot(t,y):
    return r*y[0]-y[2]-y[0]*y[2]
def z_dot(t,y):
    return y[0]*y[1]-b*y[2]

def fun(t,y):
    return [x_dot(t,y),y_dot(t,y),z_dot(t,y)]

t0=0
t_end=100
"""
for r in [0.5,1.15,1.3456,24.0,28.0]:
    if r<1:
        y0=[0.1,0.1,0.1]
    else:
        y0=[b*(r-1)+0.1, b*(r-1)+0.1, r-1+0.1]
    k = solve_ivp(fun, (t0, t_end), y0, method='RK45', max_step=0.1)
    y=k["y"]
    t=k["t"]
    fig=plt.figure()
    ax=plt.axes(projection='3d')
    ax.plot3D(y[0], y[1], y[2], label="r=%.4f"%r)
    plt.savefig("r=%.4f.pdf"%r)
plt.show()
"""
###(b)
r=27
y0=[b*(r-1)+0.1, b*(r-1)+0.1, r-1+0.1]
k = solve_ivp(fun, (t0, t_end), y0, method='RK45', max_step=0.1)
y=k["y"]
t=k["t"]
#plt.plot(t,y[2])
#plt.show()
minima=signal.argrelmin(y[2])
plt.plot(y[2][minima][3:-1], y[2][minima][4:])
plt.plot(np.linspace(20,50,10),np.linspace(20,50,10))
plt.savefig("zk.pdf")
plt.show()

```

Source code 4: Mathematica input for problem 2