

Exercise sheet 9

by Robin Heinemann (group 4) and Paul Rosendahl (group 4)

February 1, 2019

1 The Lorentz attractor

We investigate the system of differential equations

$$\begin{aligned}\dot{x} &= -\sigma(x - y) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz\end{aligned}$$

This has the fixed point $(0, 0, 0)$ and the for $r > 1$ two additional fixed points $C_{\pm} = (\pm a_0, \pm a_0, r - 1)$, $a_0 = \sqrt{b(r - 1)}$. For the further analysis we will fix $\sigma = 10$ and $b = 8/3$. We solve the set of differential equations numerically using the Runge-Kutta-Fehlberg algorithm, the *rust* program used can be found at the end. We solve the set of equations for $t < 100$ and $r \in \{0.5, 1.15, 1.3456, 23.5, 29\}$. A projection on the x - y -plane and the x - z -plane is shown for each of the cases. As initial values we choose $(0.01, 0.01, 0.01)$ for $r = 0.5$ and $C_+ + (0.01, 0.01, 0.01)$ where $r > 1$.

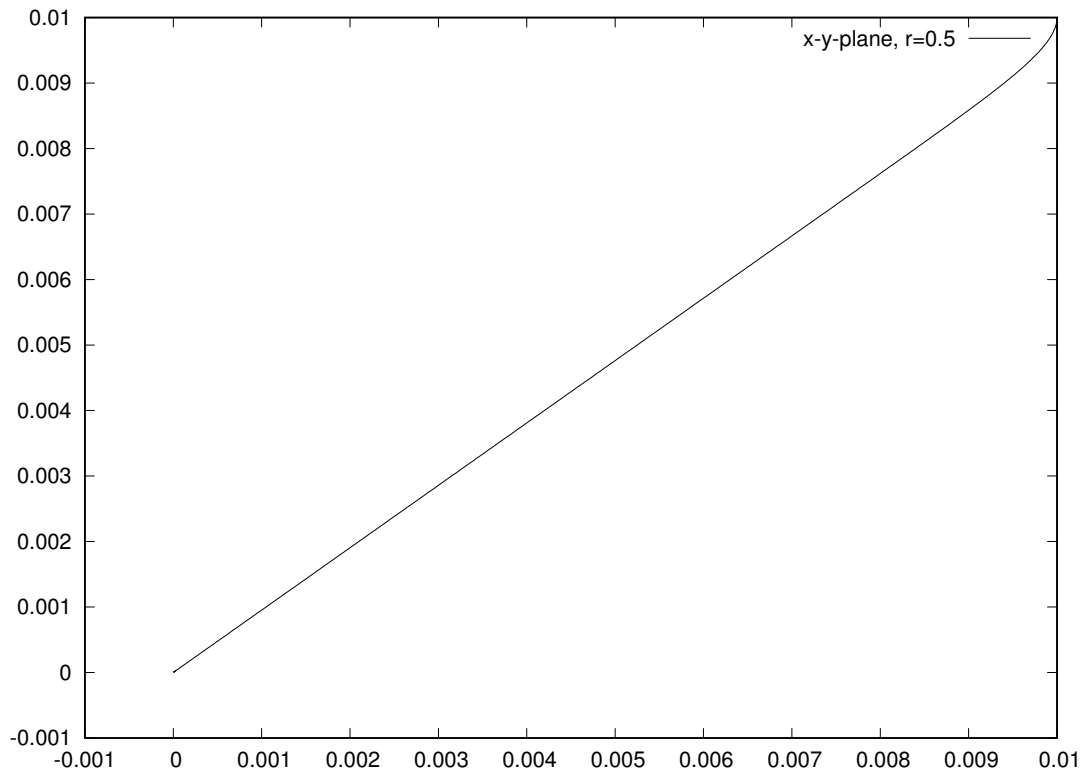


Figure 1: Plot of the projection on the x - y -plane for $r = 0.5$

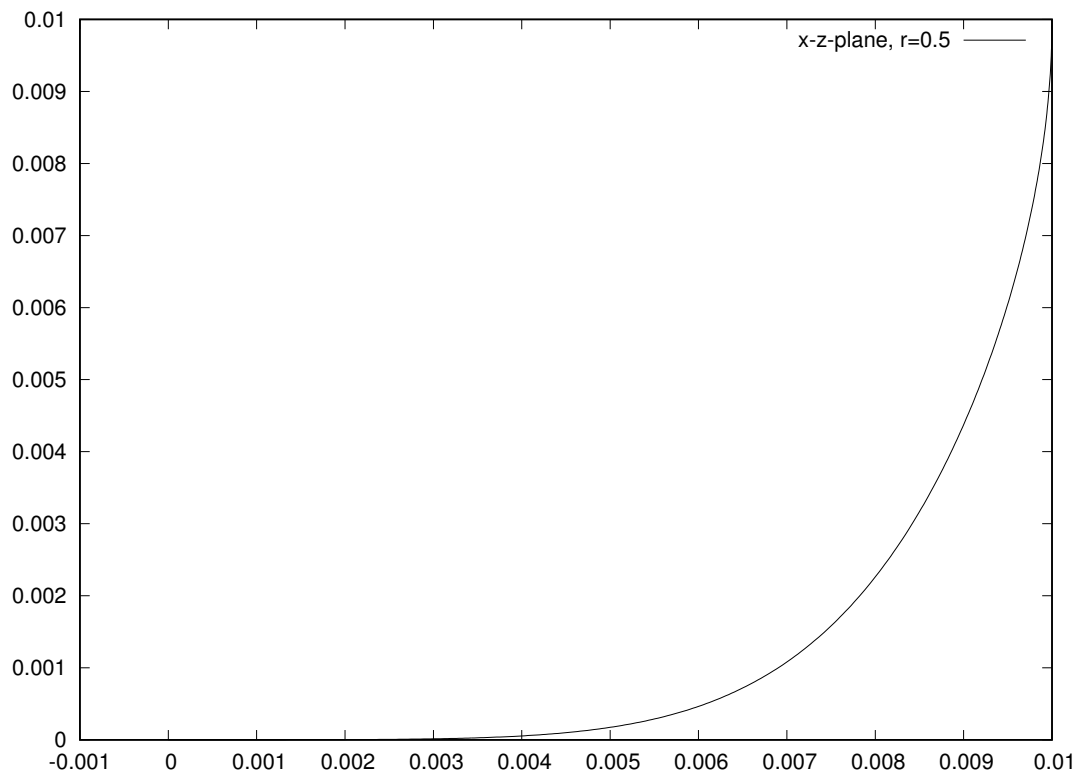


Figure 2: Plot of the projection on the x - z -plane for $r = 0.5$

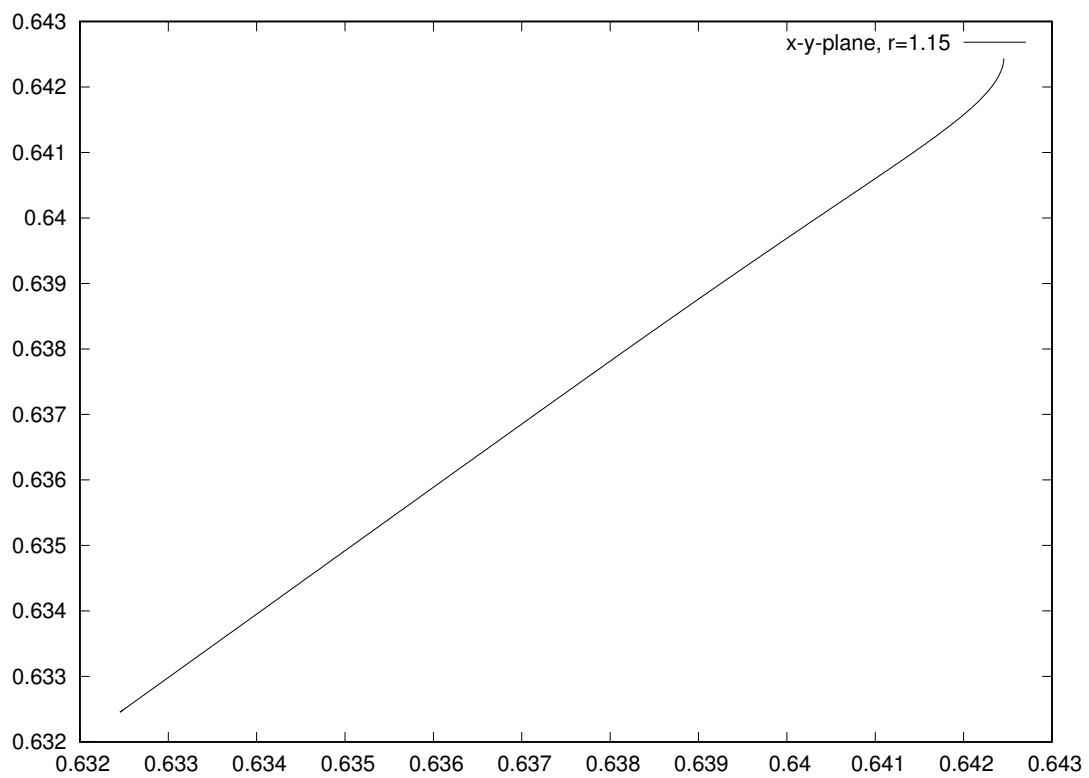


Figure 3: Plot of the projection on the x - y -plane for $r = 1.15$

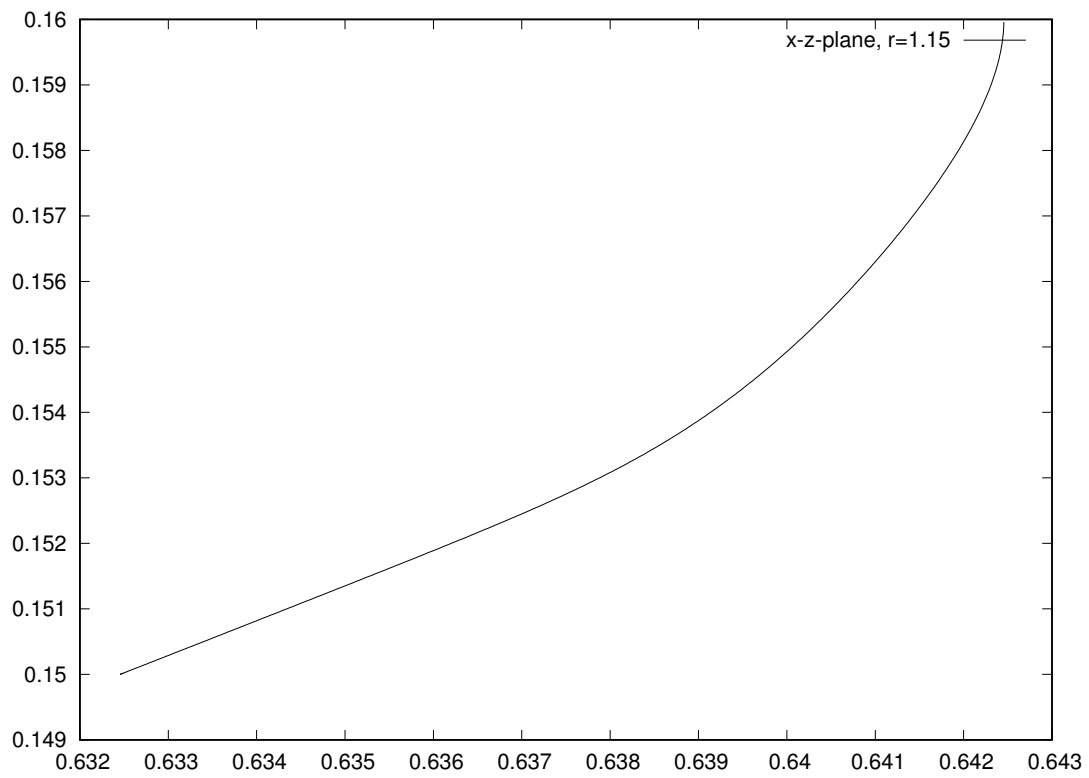


Figure 4: Plot of the projection on the x - z -plane for $r = 1.15$

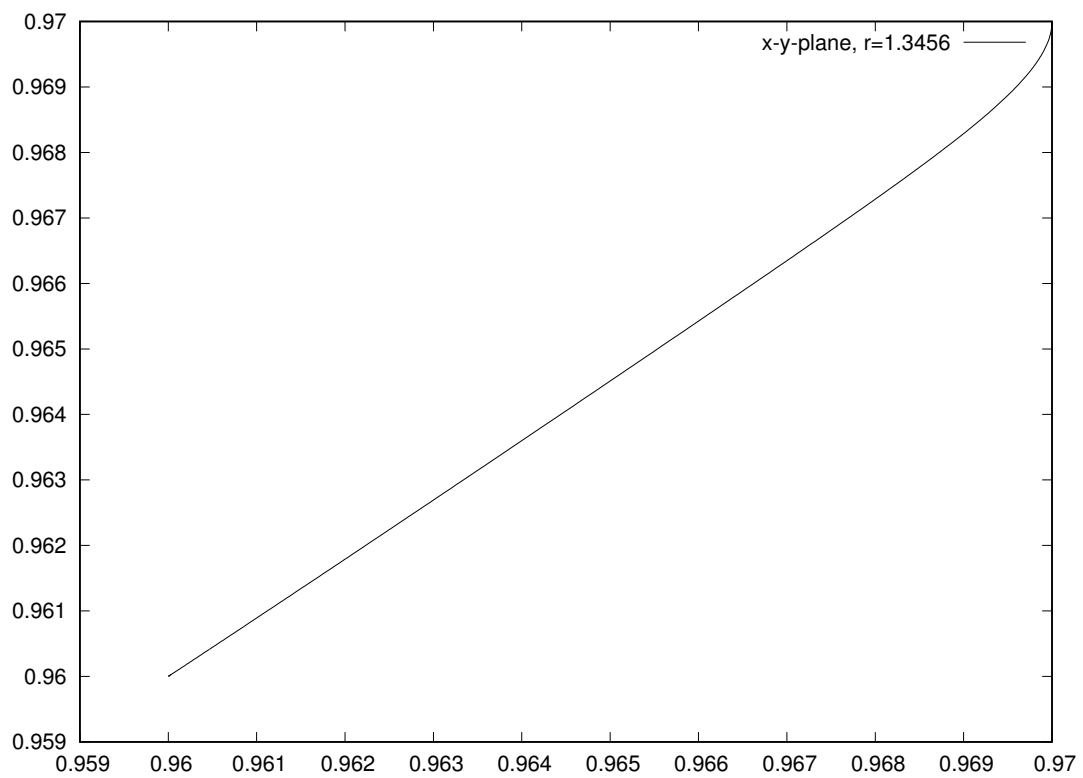


Figure 5: Plot of the projection on the x - y -plane for $r = 1.3456$

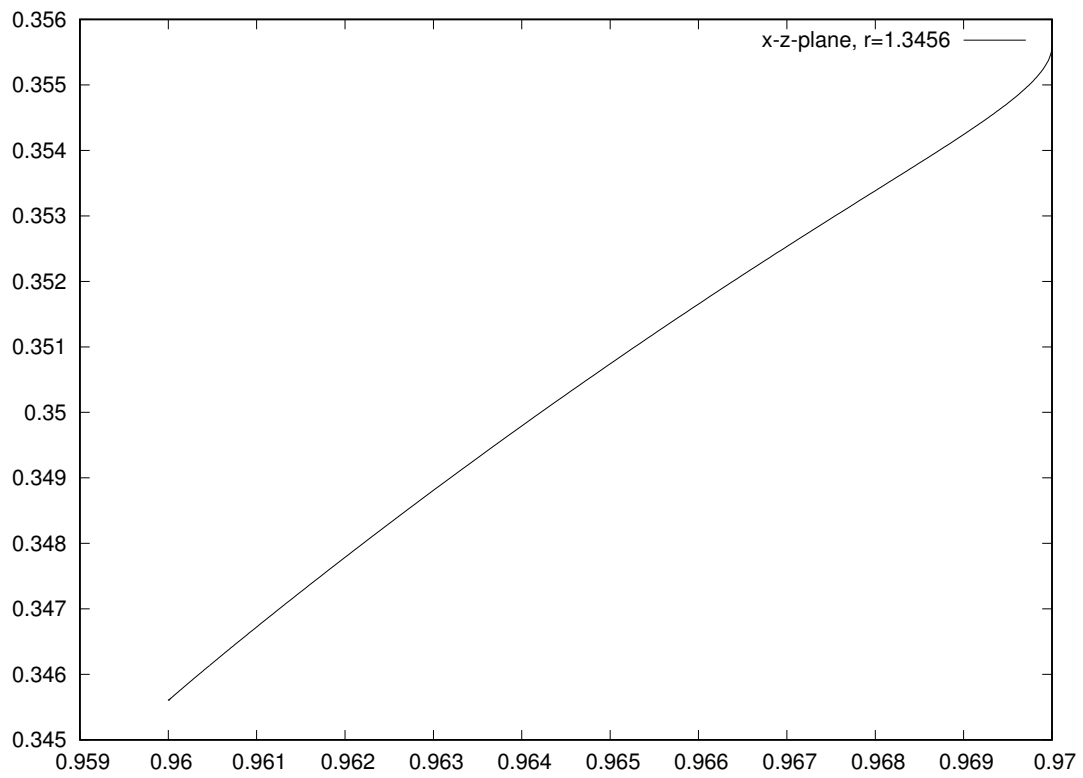


Figure 6: Plot of the projection on the x - z -plane for $r = 1.3456$

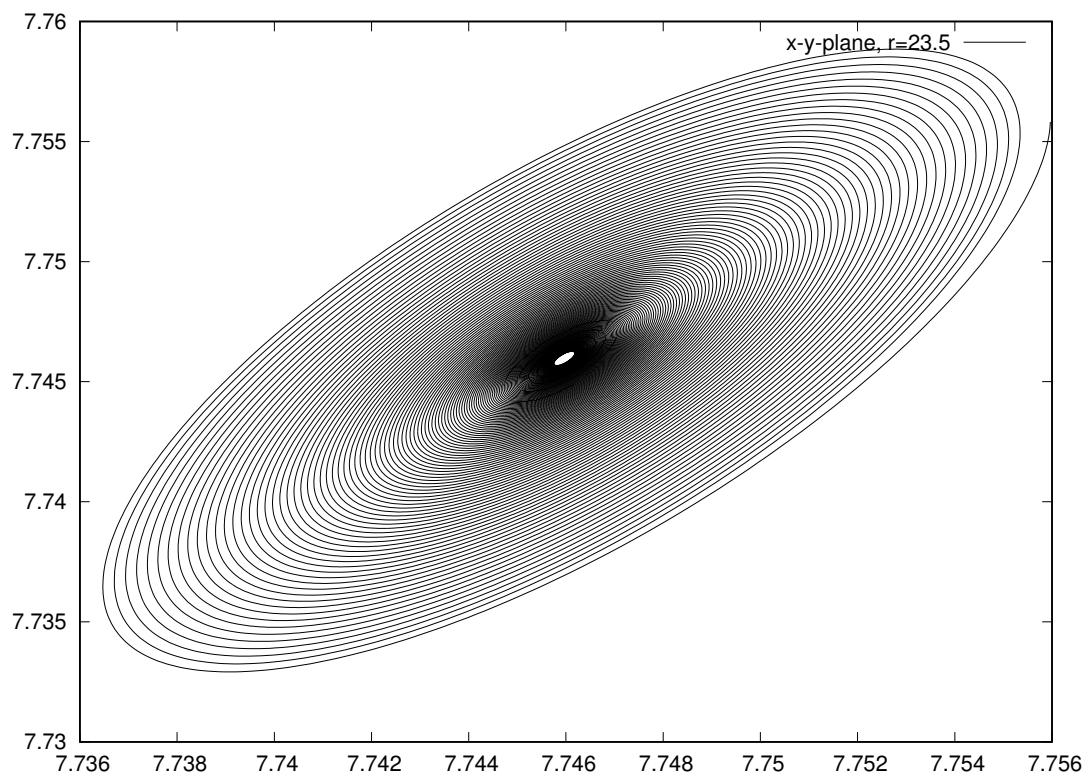


Figure 7: Plot of the projection on the x - y -plane for $r = 23.5$

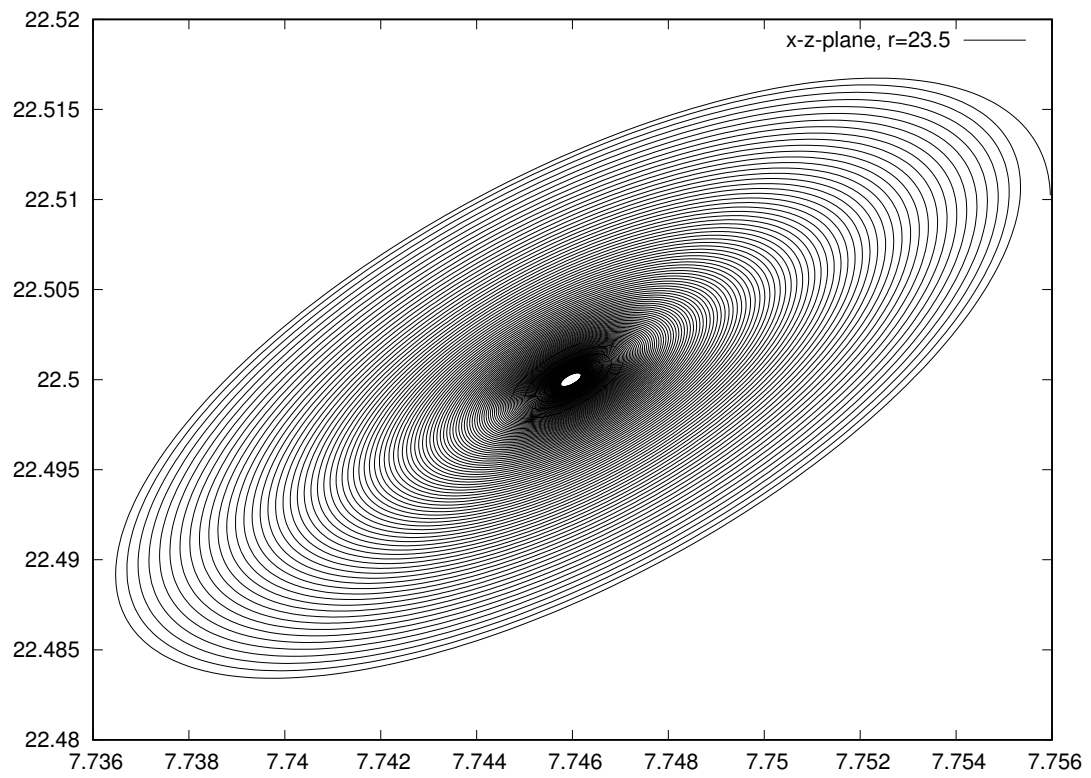


Figure 8: Plot of the projection on the x - z -plane for $r = 23.5$

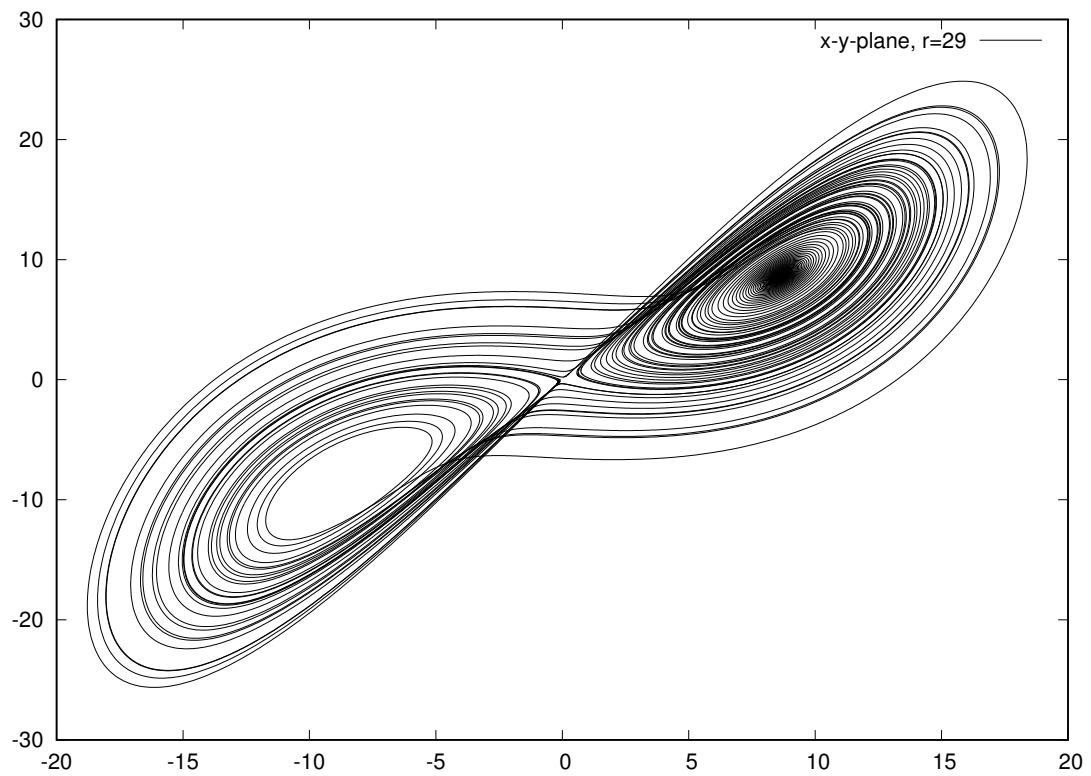


Figure 9: Plot of the projection on the x - z -plane for $r = 29$

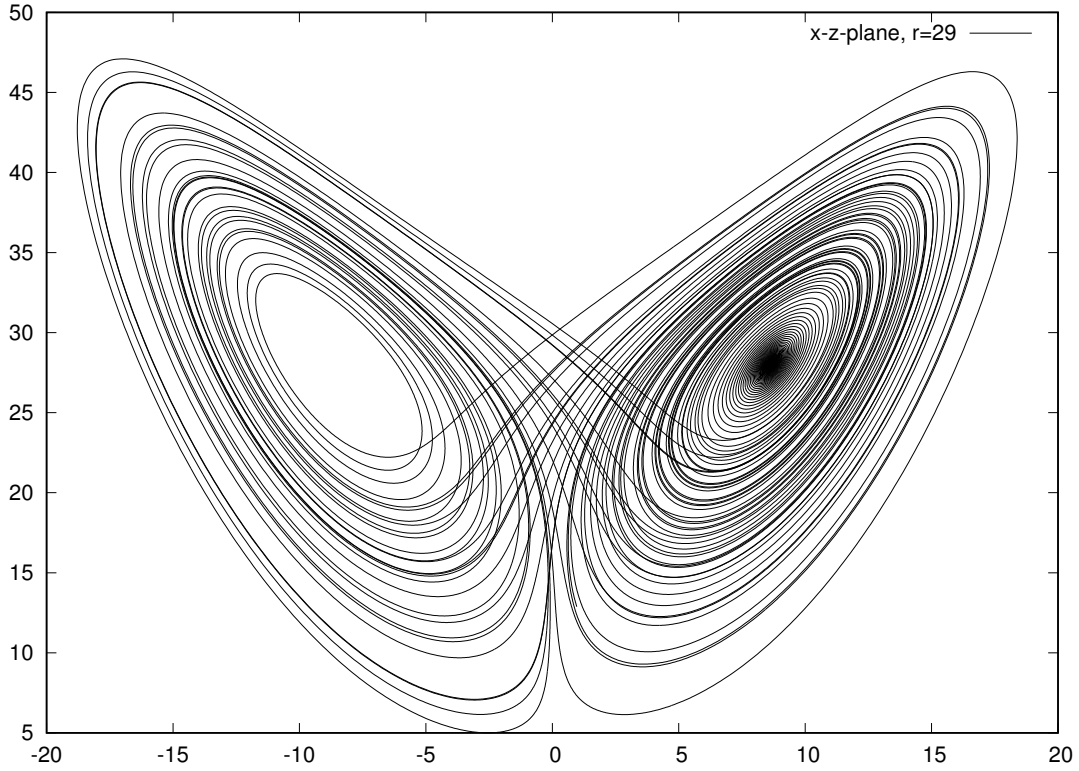


Figure 10: Plot of the projection on the x - z -plane for $r = 29$

The evolution of the system is vastly different depending on r . For $r = 0.5$, $r = 1.15$ and $r = 1.3456$ the system rapidly approaches the nearby stationary point. The evolution of a system of differential equations around a stationary point can be analysed using Taylor expansion around the stationary point and approximating the system as linear. This results in a linear system of differential equations given by the Jacobi matrix J around the stationary point (because the 0th order term is 0 for a stationary point). The general solution to a linear system of differential equations $\vec{y}'(t) = J\vec{y}(t)$ is given by $\vec{y}(t) = \exp(Jt)\vec{y}(0)$. The evolution of this system depends on the eigenvalues of the Jacobi matrix. For our the characteristic polynomial of J around C_{\pm} is given by

$$P(\lambda) = \lambda^3 + (1 + b + \sigma)\lambda^2 + b(\sigma + r)\lambda + 2\sigma b(r - 1)$$

The zeros (eigenvalues) can easily be calculated using for example *Mathematica*. For $r = 1.15$ and $r = 1.3456$ all zeros are real and negative. This means the stationary point is stable. This matches the numerically determined evolution near C_{+} , for $r = 1.15$ and $r = 1.3456$ they move towards the stationary point. For $r = 0.5$ also obtain negative eigenvalues matching the numerical calculation.

For $r = 23.5$ however the eigenvalues are no longer real, but are complex. They still have negative real parts. This means a stable oscillation around the stationary point with decreasing radius. This perfectly matches the calculated evolution, which shows an elliptic trajectory with decreasing radius around the stationary point.

For $r = 29$ the eigenvalues are also complex, but two of them have positive real parts. This causes an oscillation with increasing radius around the stationary point. This is exactly what happens in the calculated evolution, at first there is an elliptic trajectory around C_{+} with increasing radius. This trajectory gets continuously deformed until it even reaches the second stationary point. It wanders around these two stationary points.

Last we analyze $r = 26$. This time we determine the local minima of the z component. This is simply done by comparing the last three values and identifying the second last value as local minima, if it is smaller than the last value and smaller than the third last value. Let z_k denote the k th local minima. We then plot $f(z_k) = z_{k+1}$.

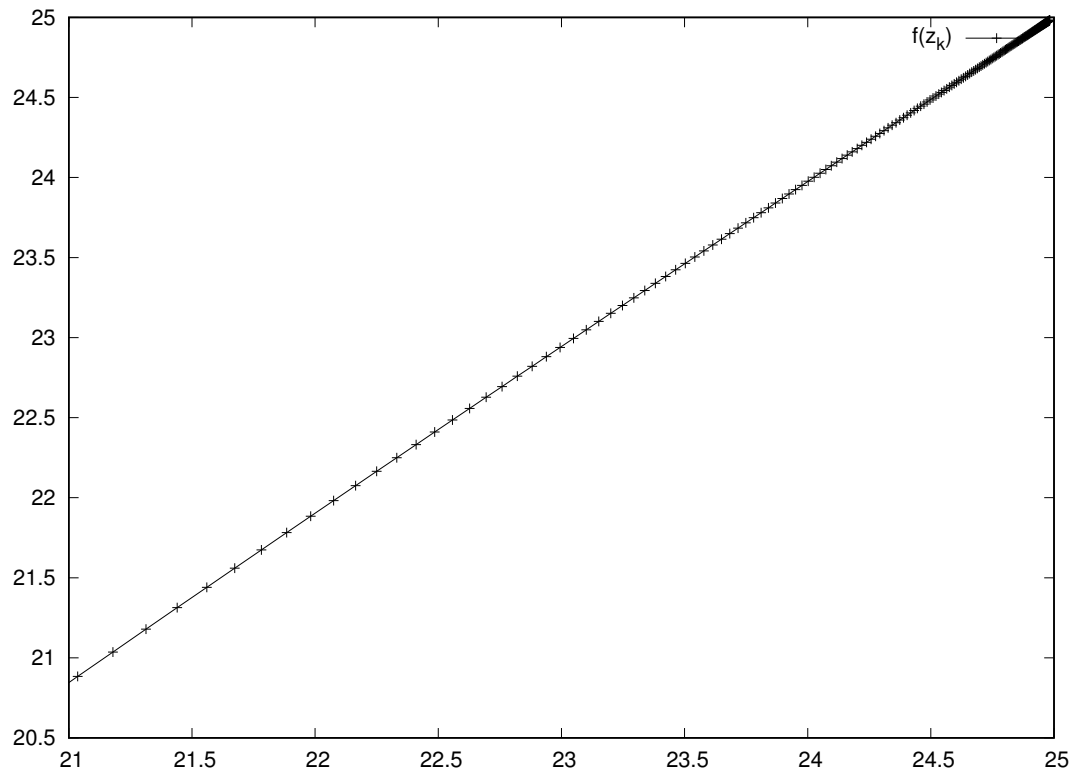
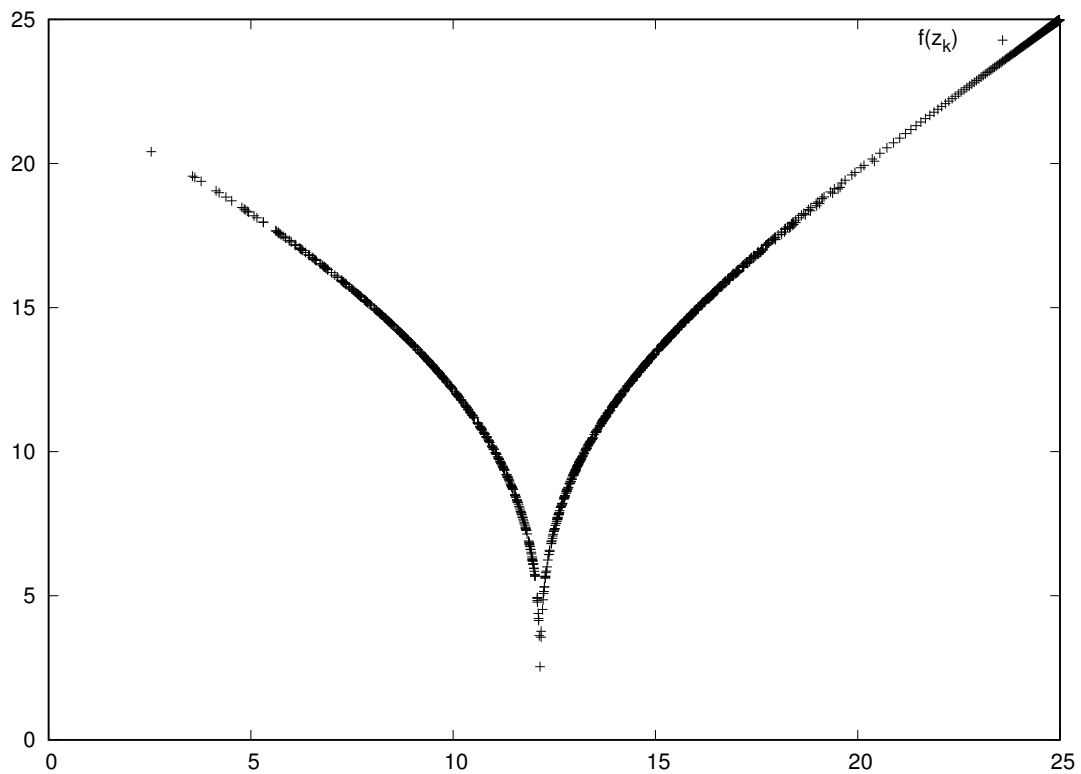


Figure 11: Plot of $f(z_k) = z_{k+1}$

By looking at the x and y coordinate of the first and the last shown point we can immediately see that the slope is positive and $|m| > 1$. Thus we can deduce by the theory of discrete maps that this solution is not periodic. Continuing the graph we can see that the system actually has trajectory around both stationary points (lines have been omitted for clarity).

Figure 12: Plot of $f(z_k) = z_{k+1}$

2 Rust implementation

```

1 // cargo-deps: rulinalg, num-traits
2 #![feature(core_float)]
3 #[macro_use]
4 extern crate rulinalg;
5 extern crate core;
6 extern crate num_traits;
7
8 use rulinalg::vector::Vector;
9 use std::ops::{Mul, Div};
10 use num_traits::pow::Pow;
11 use std::fs::File;
12 use std::path::Path;
13 use std::io::Write;
14
15 pub trait MulEx<RHS = Self> {
16     type Output;
17     fn mul(self, rhs: RHS) -> Self::Output;
18 }
19
20 pub trait AddEx<RHS = Self> {
21     type Output;
22     fn add(self, rhs: RHS) -> Self::Output;

```



```

23 }
24
25 impl MulEx<f64> for f64 {
26     type Output = f64;
27     fn mul(self, f: f64) -> f64 {
28         self * f
29     }
30 }
31
32 impl AddEx<f64> for f64 {
33     type Output = f64;
34     fn add(self, f: f64) -> f64 {
35         self + f
36     }
37 }
38
39 impl<T: Clone, TV: Clone + MulEx<T, Output = TV>> MulEx<T> for Vector<TV> {
40     type Output = Vector<TV>;
41     fn mul(mut self, f: T) -> Vector<TV> {
42
43         for val in self.mut_data().iter_mut() {
44             *val = val.clone().mul(f.clone());
45         }
46
47         self
48     }
49 }
50
51 impl<TV: Clone + AddEx<TV, Output = TV>> AddEx<Vector<TV>> for Vector<TV>
52 {
53     type Output = Vector<TV>;
54     fn add(mut self, f: Vector<TV>) -> Vector<TV> {
55         let mut i = 0;
56         for val in self.mut_data().iter_mut() {
57             *val = val.clone().add(f[i].clone());
58             i += 1;
59         }
60
61         self
62     }
63 }
64
65 fn runge_kutta_extended<T: Clone>(t: f64, x0: &Vector<T>, h: f64, f: &Fn(f64,
66     ↪ &Vector<T>) -> Vector<T>, rk_tab: &Vec<Vec<f64>>, tol: T) -> (Vector<T>, f64)
67 where Vector<T> : MulEx<f64, Output=Vector<T>>
68     , Vector<T> : AddEx<Vector<T>, Output=Vector<T>>
69     , T : core::num::Float
70     , T : Copy
71     , T : std::cmp::PartialOrd
72     , T : Div<T, Output=T>

```

```

72     , T : Div<f64, Output=T>
73     , T : Pow<f64, Output=T>
74     , T : Mul<f64, Output=f64>
75 {
76     let order = rk_tab.len() - 2;
77     let mut k = vec![x0.clone(); order];
78
79     for i in 0..order {
80         let mut x = x0.clone();
81         let rk_row = &rk_tab[i];
82
83         for j in 0..(order - 1) {
84             x = x.add(k[j].clone().mul(h * rk_row[j + 1]));
85         }
86
87
88         k[i] = f(t + h * rk_row[0], &x);
89     }
90
91     let mut xn = x0.clone();
92
93     let mut delta = x0.clone();
94     delta = delta.clone().add(delta.clone().mul(-1.0));
95
96     for i in 0..order {
97         delta = delta.add(k[i].clone().mul(rk_tab[order][i] - rk_tab[order + 1][i]));
98     }
99
100    delta = delta.mul(h);
101    for i in 0..delta.size() {
102        delta[i] = delta[i].abs();
103    }
104
105    let delta_max = delta.argmax().1;
106
107    if delta_max > tol {
108        return runge_kutta_extended(t, x0, (tol / delta_max).abs().pow(1.0 / 5.0) * h
↪ * 0.5, f, rk_tab, tol);
109    }
110
111    if delta_max < tol / 100.0_f64 {
112        return runge_kutta_extended(t, x0, (tol / delta_max).abs().pow(1.0 / 5.0) * h
↪ * 0.5, f, rk_tab, tol);
113    }
114
115    for i in 0..order {
116        xn = xn.add(k[i].clone().mul(h * rk_tab[order + 1][i]));
117    }
118
119    (xn, h)

```

```

120 }
121
122 fn lorentz(_t: f64, x: &Vector<f64>, sigma: f64, b: f64, r: f64) -> Vector<f64> {
123     let mut dx = x.clone();
124     dx[0] = - sigma * (x[0] - x[1]);
125     dx[1] = r * x[0] - x[1] - x[0] * x[2];
126     dx[2] = x[0] * x[1] - b * x[2];
127
128     return dx;
129 }
130
131 fn main() {
132     let rk45 = vec![vec![0.0,          0.0,          0.0,          0.0,
↪ 0.0,          0.0]
133         ,vec![1.0 / 4.0,    1.0 / 4.0,          0.0,          0.0,
↪ 0.0,          0.0]
134         ,vec![3.0 / 8.0,    3.0 / 32.0,          9.0 / 32.0,          0.0,
↪ 0.0,          0.0]
135         ,vec![12.0 / 13.0,   1932.0 / 2197.0, -7200.0 / 2197.0,   7296.0 /
↪ 2197.0,    0.0,          0.0]
136         ,vec![1.0,          439.0 / 216.0,    -8.0,          3680.0 /
↪ 513.0,    -845.0 / 4104.0,    0.0]
137         ,vec![1.0 / 2.0,    -8.0 / 27.0,          2.0,          -3544.0 /
↪ 2565.0,    1859.0 / 4104.0, -11.0/40.0]
138         ,vec![25.0 / 216.0, 0.0,          1408.0 / 2565.0,   2197.0 /
↪ 4104.0,    -1.0/5.0,          0.0]
139         ,vec![16.0 / 135.0, 0.0,          6656.0 / 12825.0,  28561.0 /
↪ 56430.0, -9.0/50.0,          2.0/55.0]];
140
141
142     let path = Path::new("out.csv");
143     let mut file = File::create(&path).unwrap();
144
145     let rs = [0.5_f64, 1.15, 1.3456, 23.5, 29.0];
146     let sigma = 10.0;
147     let b = 8.0_f64 / 3.0;
148
149     for r in rs.iter() {
150         let T = 100.0;
151         let mut dt = 0.001;
152         let mut t = 0.0;
153
154         let a0 = (b * (r - 1.0)).sqrt();
155
156         let mut xn = vector![0.0, 0.0, 0.0];
157         if *r > 1.0 {
158             xn += vector![a0, a0, r - 1.0];
159         }
160
161         xn += vector![0.01, 0.01, 0.01];

```

```

162
163     let mut last_out = 0.0;
164
165     while t < T {
166         let (xn_n, dt_n) = runge_kutta_extended(t, &xn, dt, &|t, x| {
167             lorentz(t, x, sigma, b, *r)
168         }, &rk45, 1e-12);
169
170         xn = xn_n;
171         dt = dt_n;
172
173         t += dt;
174
175         if (t - last_out) > 0.001 {
176             writeln!(file, "{}", t, xn[0], xn[1], xn[2]).unwrap();
177             last_out = t;
178         }
179     }
180
181     writeln!(file, "").unwrap();
182     writeln!(file, "").unwrap();
183 }
184
185 let r = 26.0;
186 let T = 1000.0;
187 let mut dt = 0.001;
188 let mut t = 0.0;
189
190 let a0 = (b * (r - 1.0)).sqrt();
191
192 let mut xn = vector![0.0, 0.0, 0.0];
193
194 if r > 1.0 {
195     xn += vector![a0, a0, r - 1.0];
196 }
197
198 xn += vector![0.01, 0.01, 0.01];
199
200 let mut last_z = vector![0.0, 0.0, 0.0];
201 let mut last_min = 0.0;
202 let mut first = true;
203
204 while t < T {
205     let (xn_n, dt_n) = runge_kutta_extended(t, &xn, dt, &|t, x| {
206         lorentz(t, x, sigma, b, r)
207     }, &rk45, 1e-12);
208
209     xn = xn_n;
210     dt = dt_n;
211

```

```
212     last_z[0] = last_z[1];
213     last_z[1] = last_z[2];
214     last_z[2] = xn[2];
215
216     t += dt;
217
218     if last_z[0] > last_z[1] && last_z[1] < last_z[2] {
219         if !first {
220             writeln!(file, "{}", last_min, last_z[1]).unwrap();
221         } else {
222             first = false;
223         }
224
225         last_min = last_z[1];
226     }
227 }
228 }
```

Listing 1: rust implementation of the Runge-Kutta-Fehlberg integrator and application to the lorentz attractor