Exercise sheet 9

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

July 21, 2018

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 $r > 1$.

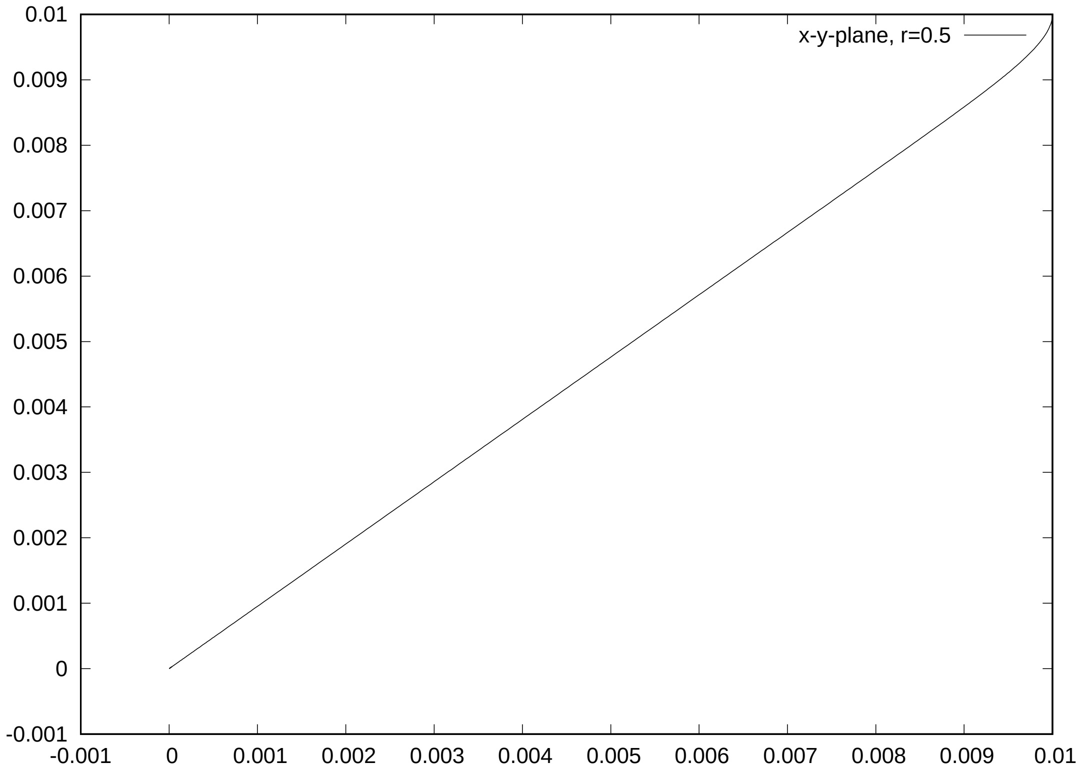

Figure 1: Plot of the projection on the x-y-plane for $r = 0.5$

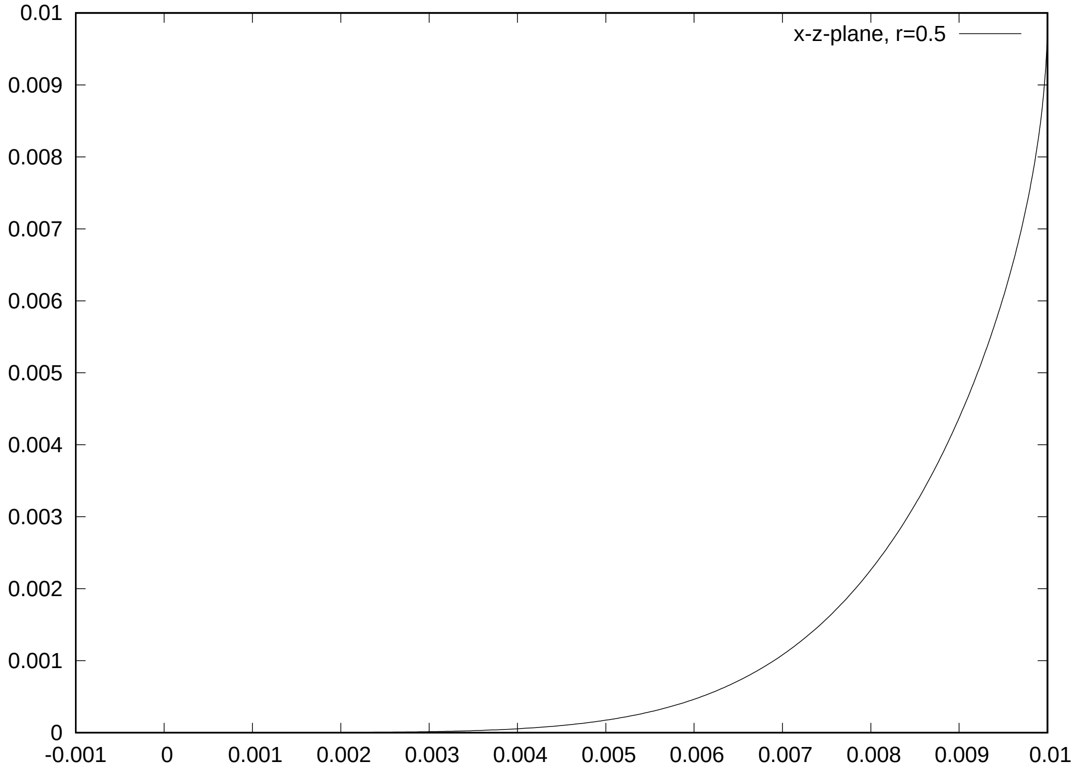

Figure 2: Plot of the projection on the x-z-plane for $r = 0.5$

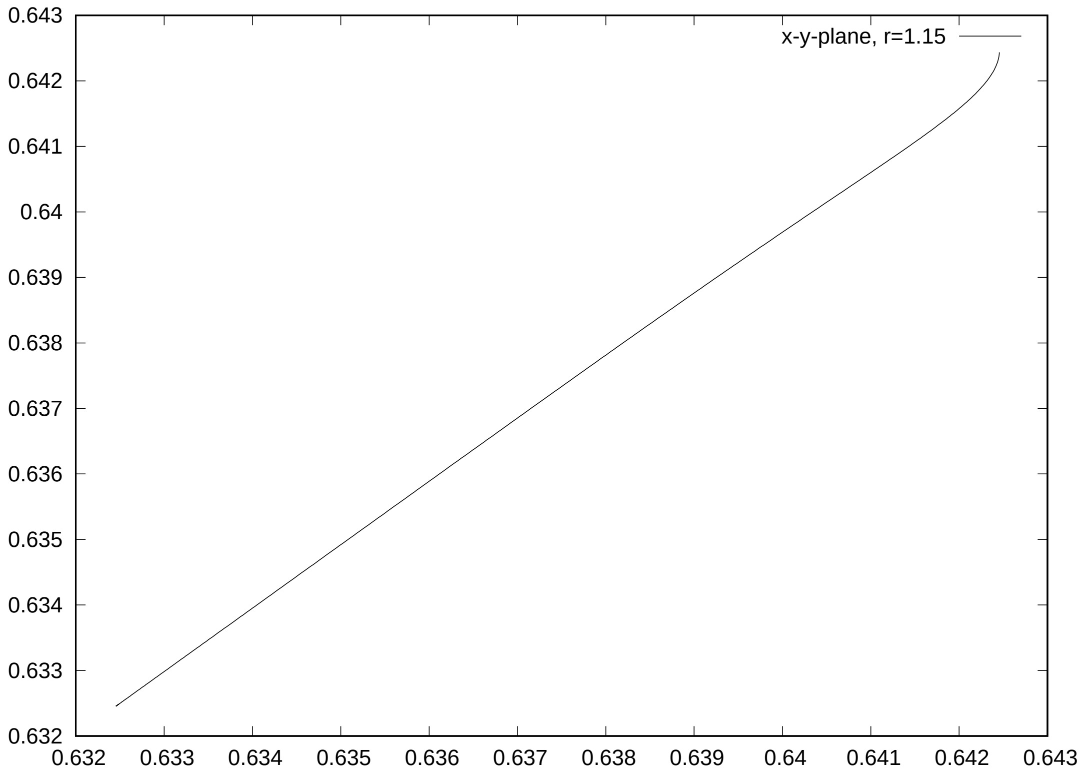

Figure 3: Plot of the projection on the x-y-plane for $r = 1.15$

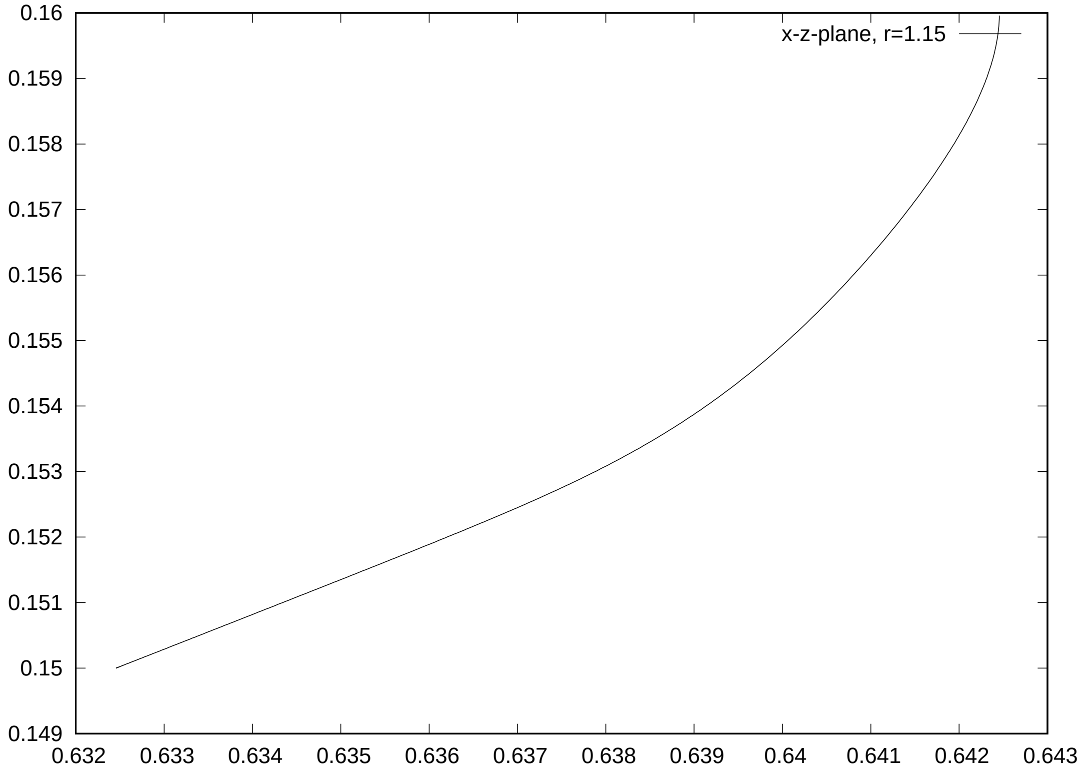

Figure 4: Plot of the projection on the x-z-plane for $r = 1.15$

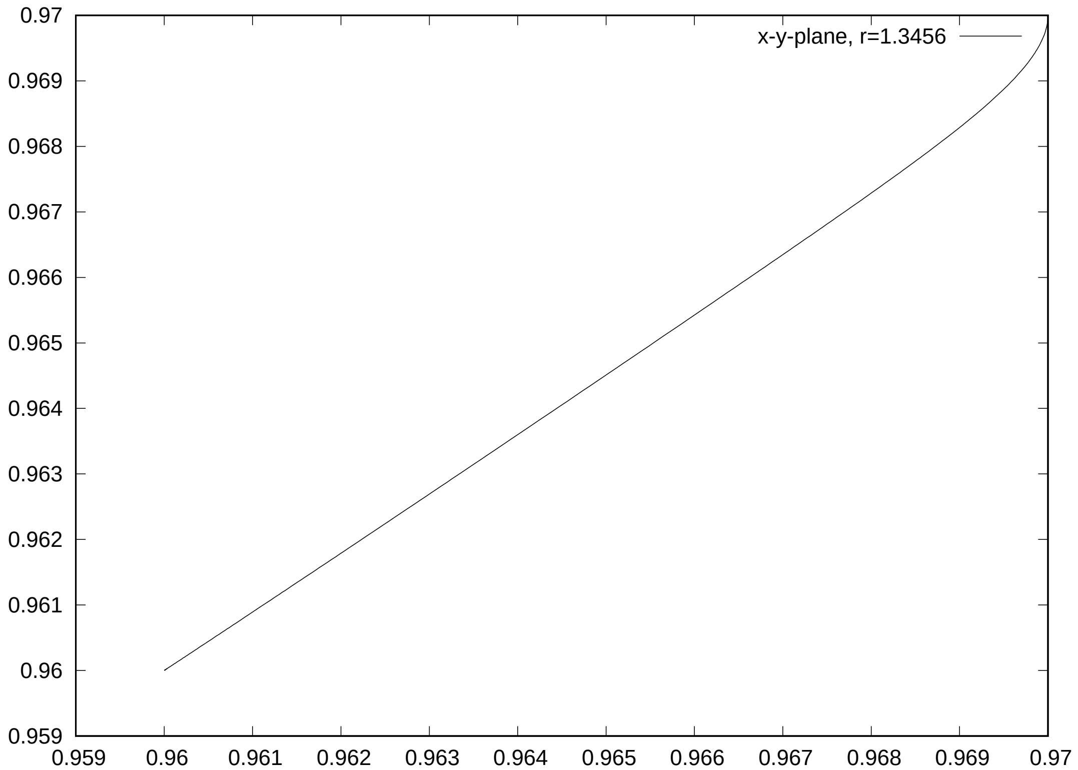

Figure 5: Plot of the projection on the x-y-plane for $r = 1.3456$

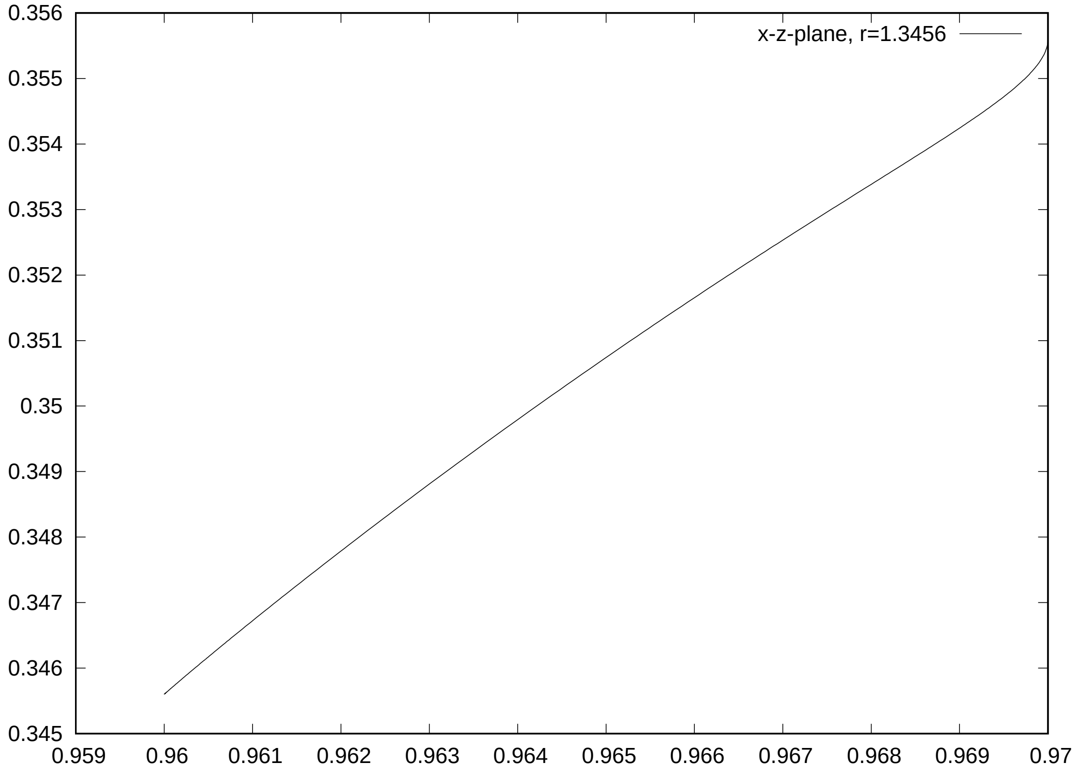

Figure 6: Plot of the projection on the x-z-plane for $r = 1.3456$

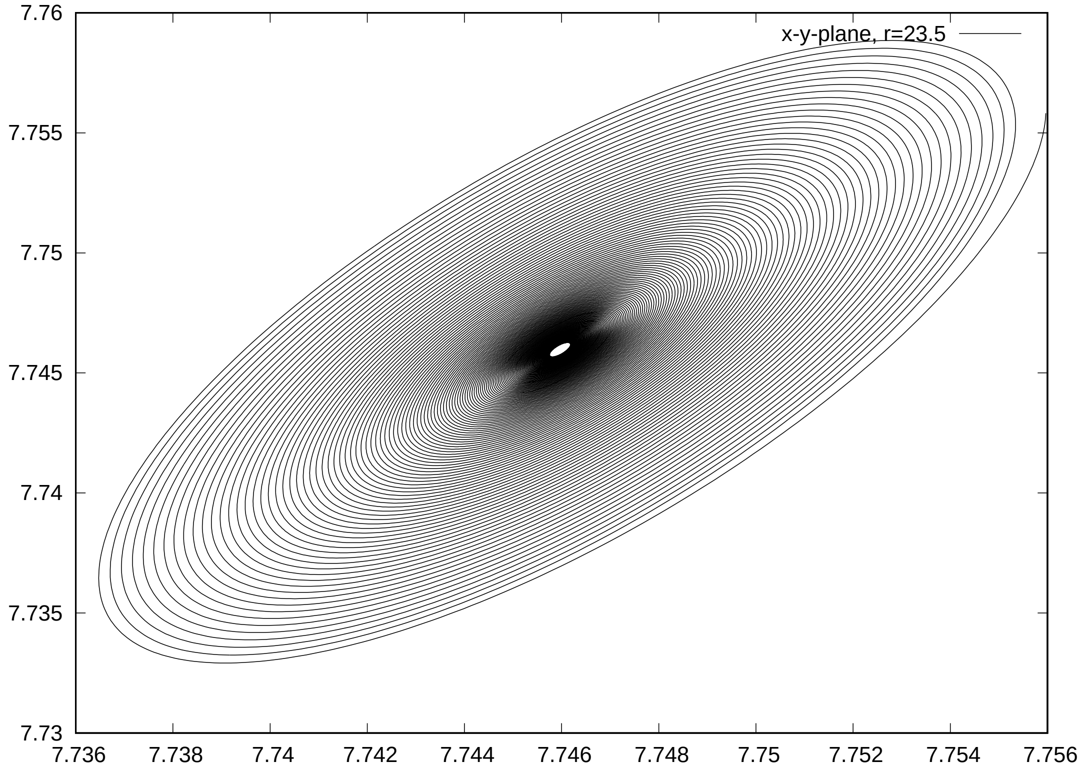

Figure 7: Plot of the projection on the x-y-plane for $r = 23.5$

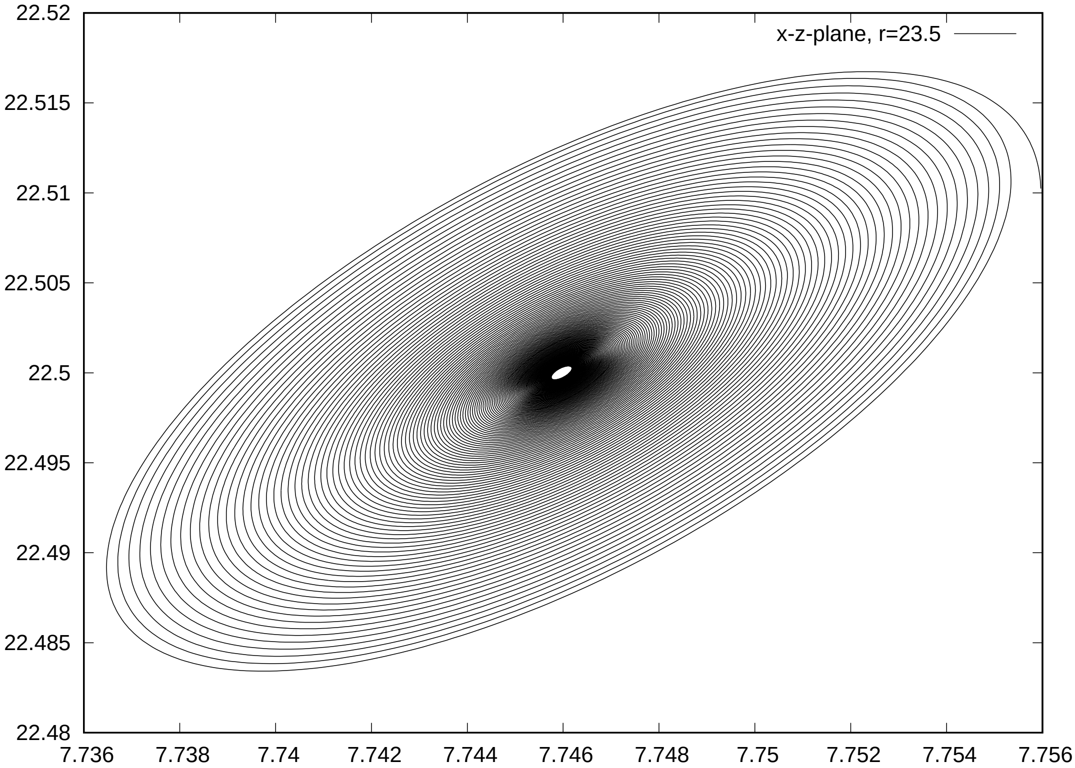

Figure 8: Plot of the projection on the x-z-plane for $r = 23.5$

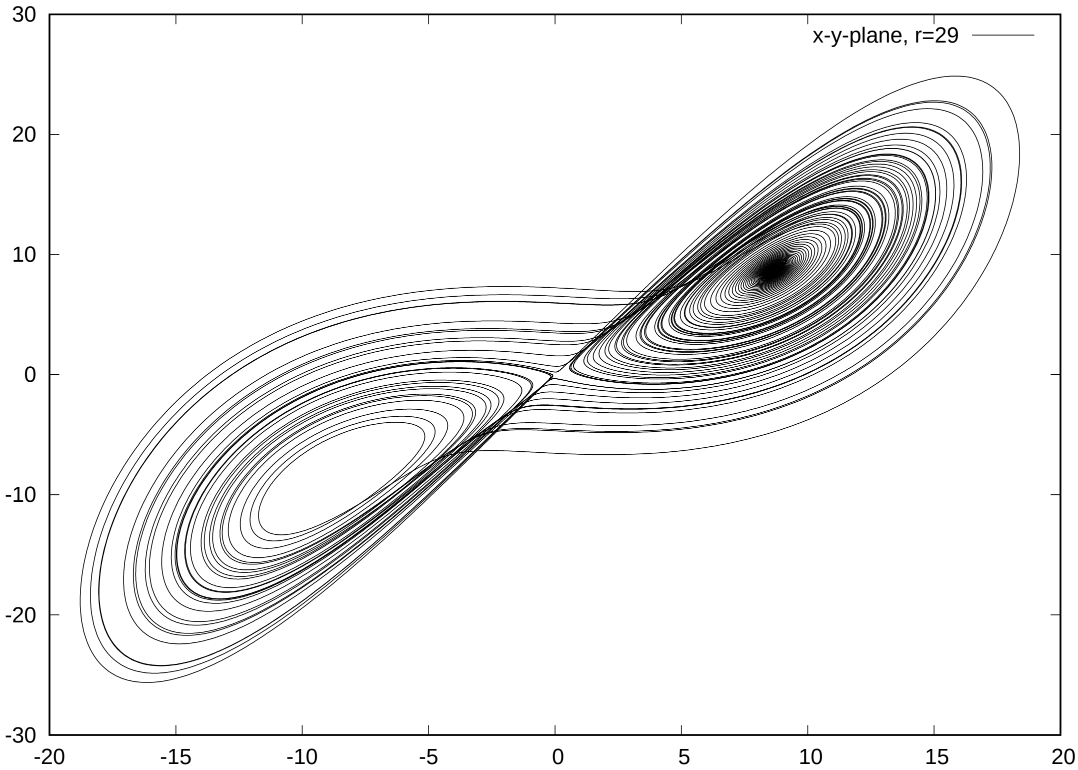

Figure 9: Plot of the projection on the x-z-plane for $r = 29$

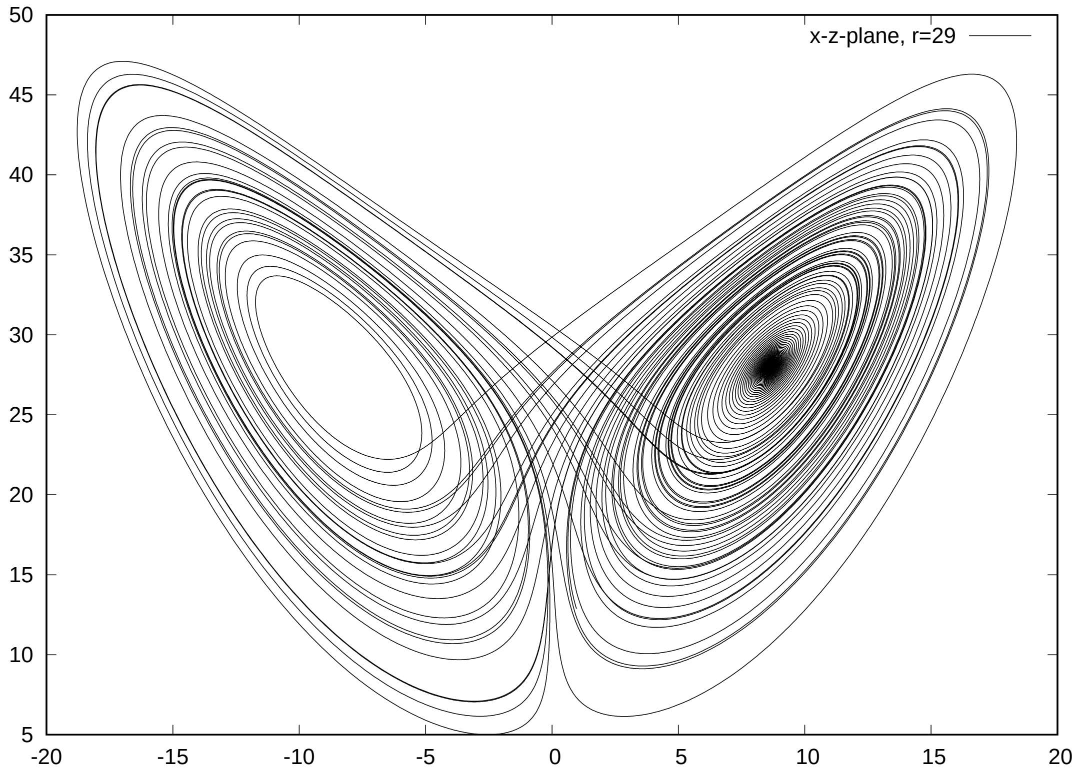

Figure 10: Plot of the projection on the x-z-plane for $r = 29$

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 approches 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 give 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 $\vec{y}'(t) = J\vec{y}(j)$ is given by $\vec{y}(t) = \exp(Jt)$. 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 negativ. 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 a 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 as oscillation with increasing radius around the stationary point. This is exactly what happens in the calculated evolution, at first there is a 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 analze $r = 26$. This time we determine the local minima of the z component. This 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 kth local minima. We then plot $f(z_k) = z_{k+1}$.

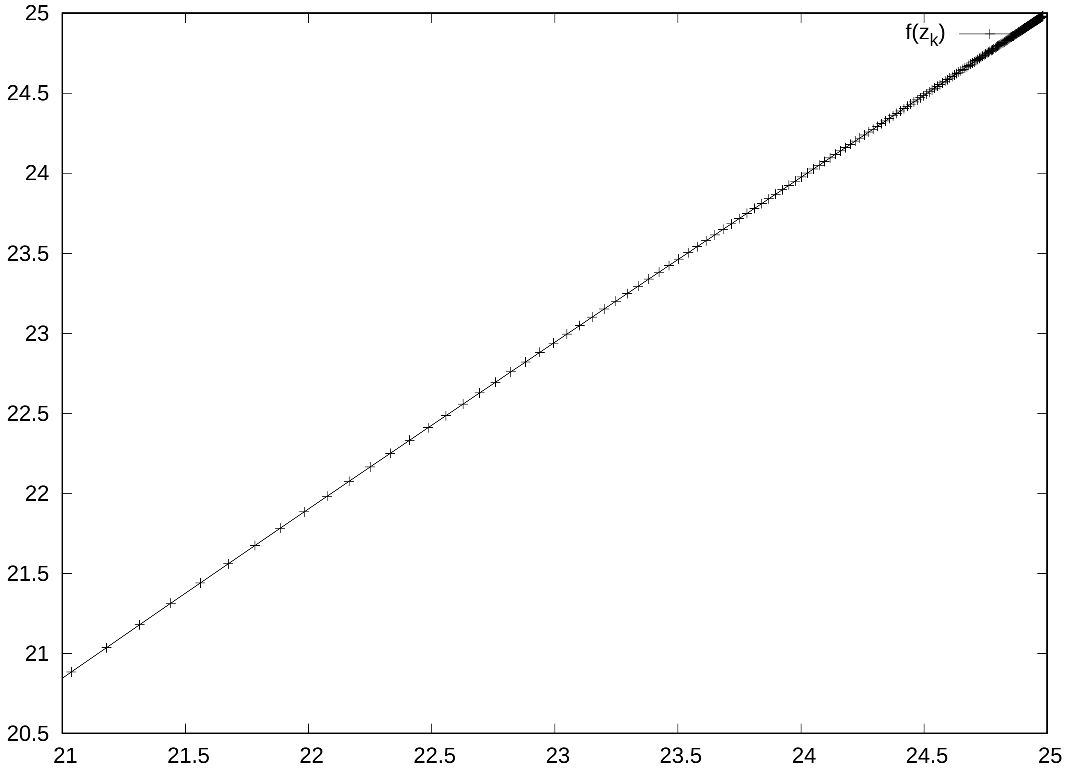

Figure 11: Plot of $f(z_k) = z_{k+1}$

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 ommited for clarity).

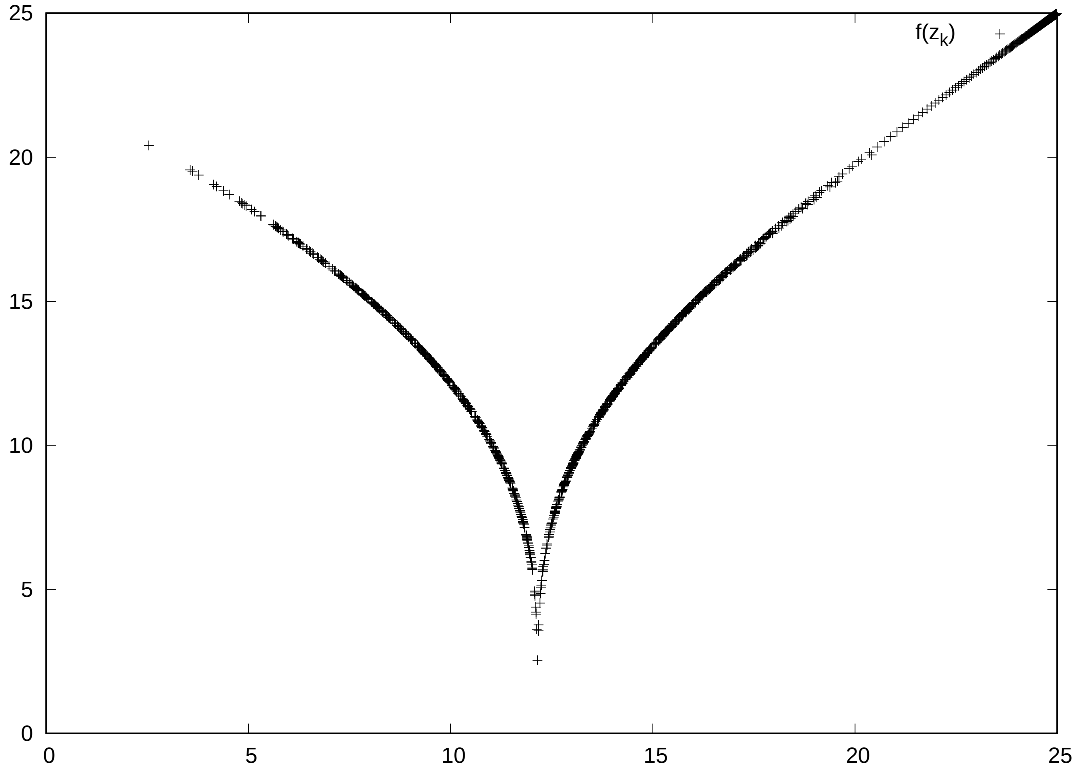

Figure 12: Plot of $f(z_k) = z_{k+1}$

2 Rust implementation

```
// cargo-deps: rulinalg, num-traits
#![feature(core_float)]
#[macro_use]
extern crate rulinalg;
extern crate core;
extern crate num_traits;

use rulinalg::vector::Vector;
use std::ops::{Mul, Div};
use num_traits::pow::Pow;
use std::fs::File;
use std::path::Path;
use std::io::Write;

pub trait MulEx<RHS = Self> {
    type Output;
    fn mul(self, rhs: RHS) -> Self::Output;
}

pub trait AddEx<RHS = Self> {
    type Output;
    fn add(self, rhs: RHS) -> Self::Output;
```

```
}

impl MulEx<f64> for f64 {
    type Output = f64;
    fn mul(self, f: f64) -> f64 {
    self * f
    }
}

impl AddEx<f64> for f64 {
    type Output = f64;
    fn add(self, f: f64) -> f64 {
    self + f
    }
}

impl<T: Clone, TV: Clone + MulEx<T, Output = TV>> MulEx<T> for Vector<TV> {
    type Output = Vector<TV>;
    fn mul(mut self, f: T) -> Vector<TV> {

    for val in self.mut_data().iter_mut() {
        *val = val.clone().mul(f.clone());
    }

    self
    }
}

impl<TV: Clone + AddEx<TV, Output = TV>> AddEx<Vector<TV>> for Vector<TV>
{
    type Output = Vector<TV>;
    fn add(mut self, f: Vector<TV>) -> Vector<TV> {
    let mut i = 0;
    for val in self.mut_data().iter_mut() {
        *val = val.clone().add(f[i].clone());
        i += 1;
    }

    self
    }
}

fn runge_kutta_extended<T: Clone>(t: f64, x0: &Vector<T>, h: f64, f: &Fn(f64,
 ↪  &Vector<T>) -> Vector<T>, rk_tab: &Vec<Vec<f64>>, tol: T) -> (Vector<T>, f64)
where Vector<T> : MulEx<f64, Output=Vector<T>>
    , Vector<T> : AddEx<Vector<T>, Output=Vector<T>>
    , T : core::num::Float
    , T : Copy
    , T : std::cmp::PartialOrd
    , T : Div<T, Output=T>
```

```
    , T : Div<f64, Output=T>
    , T : Pow<f64, Output=T>
    , T : Mul<f64, Output=f64>
{
    let order = rk_tab.len() - 2;
    let mut k = vec![x0.clone(); order];

    for i in 0..order {
      let mut x = x0.clone();
      let rk_row = &rk_tab[i];

      for j in 0..(order - 1) {
        x = x.add(k[j].clone().mul(h * rk_row[j + 1]));
    }


    k[i] = f(t + h * rk_row[0], &x);
    }

    let mut xn = x0.clone();

    let mut delta = x0.clone();
    delta = delta.clone().add(delta.clone().mul(-1.0));

    for i in 0..order {
    delta = delta.add(k[i].clone().mul(rk_tab[order][i] - rk_tab[order + 1][i]));
    }

    delta = delta.mul(h);
    for i in 0..delta.size() {
    delta[i] = delta[i].abs();
    }

    let delta_max = delta.argmax().1;

    if delta_max > tol {
    return runge_kutta_extended(t, x0, (tol / delta_max).abs().pow(1.0 / 5.0) * h
 ↪  * 0.5, f, rk_tab, tol);
    }

    if delta_max < tol / 100.0_f64 {
    return runge_kutta_extended(t, x0, (tol / delta_max).abs().pow(1.0 / 5.0) * h
 ↪  * 0.5, f, rk_tab, tol);
    }

    for i in 0..order {
    xn = xn.add(k[i].clone().mul(h * rk_tab[order + 1][i]));
    }

    (xn, h)
```

```
}

fn lorentz(_t: f64, x: &Vector<f64>, sigma: f64, b: f64, r: f64) -> Vector<f64> {
    let mut dx = x.clone();
    dx[0] = - sigma * (x[0] - x[1]);
    dx[1] = r * x[0] - x[1] - x[0] * x[2];
    dx[2] = x[0] * x[1] - b * x[2];

    return dx;
}

fn main() {
    let rk45 = vec![vec![0.0,          0.0,              0.0,              0.0,              0.0,              0.0]
           ,vec![1.0 / 4.0,    1.0 / 4.0,        0.0,              0.0,              0.0,              0.0]
           ,vec![3.0 / 8.0,    3.0 / 32.0,       9.0 / 32.0,       0.0,              0.0,              0.0]
           ,vec![12.0 / 13.0,  1932.0 / 2197.0, -7200.0 / 2197.0,  7296.0 / 2197.0,  0.0,              0.0]
           ,vec![1.0,          439.0 / 216.0,   -8.0,              3680.0 / 513.0,   -845.0 / 4104.0,  0.0]
           ,vec![1.0 / 2.0,   -8.0 / 27.0,       2.0,             -3544.0 / 2565.0,  1859.0 / 4104.0, -11.0/40.0]
           ,vec![25.0 / 216.0, 0.0,              1408.0 / 2565.0,  2197.0 / 4104.0,  -1.0/5.0,         0.0]
           ,vec![16.0 / 135.0, 0.0,              6656.0 / 12825.0, 28561.0 / 56430.0, -9.0/50.0,       2.0/55.0]];


    let path = Path::new("out.csv");
    let mut file = File::create(&path).unwrap();

    let rs = [0.5_f64, 1.15, 1.3456, 23.5, 29.0];
    let sigma = 10.0;
    let b = 8.0_f64 / 3.0;

    for r in rs.iter() {
    let T = 100.0;
    let mut dt = 0.001;
    let mut t = 0.0;

    let a0 = (b * (r - 1.0)).sqrt();

    let mut xn = vector![0.0, 0.0, 0.0];
    if *r > 1.0 {
        xn += vector![a0, a0, r - 1.0];
    }

    xn += vector![0.01, 0.01, 0.01];
```

```

    let mut last_out = 0.0;

    while t < T {
        let (xn_n, dt_n) = runge_kutta_extended(t, &xn, dt, &|t, x| {
                lorentz(t, x, sigma, b, *r)
             }, &rk45, 1e-12);

        xn = xn_n;
        dt = dt_n;

        t += dt;

        if (t - last_out) > 0.001 {
        writeln!(file, "{}, {}, {}, {}", t, xn[0], xn[1], xn[2]).unwrap();
        last_out = t;
        }
    }

    writeln!(file, "").unwrap();
    writeln!(file, "").unwrap();
    }

    let r = 26.0;
    let T = 1000.0;
    let mut dt = 0.001;
    let mut t = 0.0;

    let a0 = (b * (r - 1.0)).sqrt();

    let mut xn = vector![0.0, 0.0, 0.0];

    if r > 1.0 {
    xn += vector![a0, a0, r - 1.0];
    }

    xn += vector![0.01, 0.01, 0.01];

    let mut last_z = vector![0.0, 0.0, 0.0];
    let mut last_min = 0.0;
    let mut first = true;

    while t < T {
    let (xn_n, dt_n) = runge_kutta_extended(t, &xn, dt, &|t, x| {
                lorentz(t, x, sigma, b, r)
             }, &rk45, 1e-12);

    xn = xn_n;
    dt = dt_n;

```

```
    last_z[0] = last_z[1];
    last_z[1] = last_z[2];
    last_z[2] = xn[2];

    t += dt;

    if last_z[0] > last_z[1] && last_z[1] < last_z[2] {
            if !first {
        writeln!(file, "{}, {}", last_min, last_z[1]).unwrap();
            } else {
                first = false;
        }

        last_min = last_z[1];
    }
    }
}
```

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