Assignment

Exercise sheet 6 - Equations of States

Name TN 1: Tim Peinkofer

tim.peinkofer@student.uni-tuebingen.de

Name TN 2: Fabian Kostow

fabian.kostow@student.uni-tuebingen.de

Tutor: Jose Carlos Olvera Meneses

Date: 2. Dezember 2024

Inhaltsverzeichnis

1 Note **2**

2 Part I **3**

3 Part II **4**

4 Part III **5**

4.1 III a 5
4.2 III b 8
4.3 Part III c 10
4.4 III d 13

1 Note

This document contains only code snippets to explain them, therefore they don't work independently. The complete code is on GitHub

2 Part I

In this part, we want to discuss the code for the Lagrange-Polynomial interpolation.

```
def LagrangeInterpolation (x,xs,ys):
    interpolated_value = 0
    for n in range(len(xs)):
      interpolated_value = interpolated_value + ys[n] *
                                             LagrangeCoeffiecient(n,
                                             x, xs,ys)
    return interpolated_value

def LagrangeCoeffiecient(n, x, xs,ys):
    denominator = 1
    numerator = 1

    for i in range(len(xs)):
        if i != n:
            ys = xs[n] - xs[i]
            if ys != 0:
                denominator = denominator * ys
                numerator = numerator * (x - xs[i])

    return numerator / denominator
```

Here we have two functions:

- in the "LagrangeInterpolation"function we simply multiply the known values to the corresponding Lagrange Coefficients and add these values up.
- In the "LagrangeCoefficient"function we calculate the the needed coefficient, which is the known algorithm. But we have to check that we don't divide by zero.

3 Part II

Here we want to approximate a Heaviside function in the following form

$$H(x) = \frac{1}{2} \cdot (1 + \tanh(\alpha x))$$

To approximate a hybrid-barotropic EoS:

$$P(\rho) = H(\rho_T - \rho)\kappa_1 \rho^{\Gamma_1} + H(\rho - \rho_T)\kappa_2 \rho^{\Gamma_2}$$

With the constants $\alpha = 5$,$\rho_T = 5$,$\kappa_1 = 20$,$\kappa_2 = 1$, $\Gamma_1 = \frac{4}{3}$, $\Gamma_2 = \frac{5}{3}$

The associated code to define these functions is

```
def H(x):
    return 0.5*(1 + np.tanh(5*x))

def P(rho):
    return H(5-rho) *20*rho**(4/3) + H(rho-5) * 1 * rho**(5/3
                                  )
```

4 Part III

```
def Plot (rho_0,rho_1, n,m,):
    #known values
    xs = np.linspace(rho_0,rho_1,n)
    ys = [P(x) for x in xs]
    #interpolated values
    xk= np.linspace(rho_0,rho_1,m)
    yk=[ LagrangeInterpolation(x,xs,ys) for x in xk]
    return xk,yk,xs,ys
```

Here we can choose with "rho_0"and "rho_1" the length of the interval we want to discuss. xs, ys are the coordinates of the known values, and xk, yk the coordinates of the interpolated values. n is the number of known points, and m is the number of interpolated points.

4.1 III a

In this section we want to test our function around the transition point on the interval [4.5,5.5] for different numbers of knwon points therefore we plot the true function, the known points, and the interpolated points.

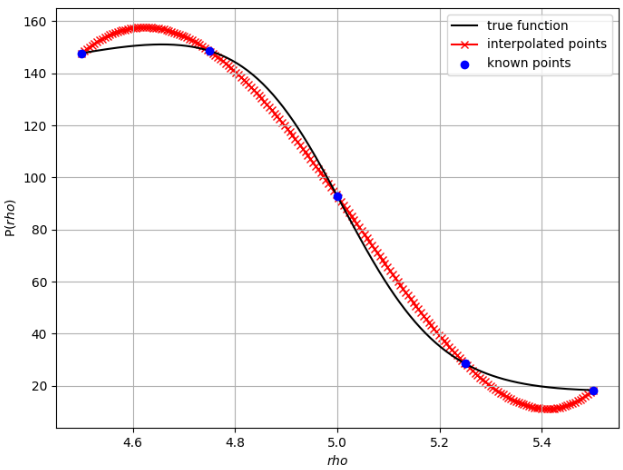

Abbildung 1: Interpolation with 5 known points

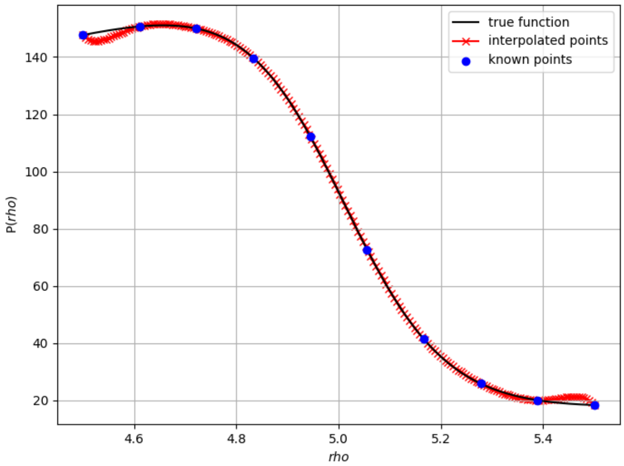

Abbildung 2: Interpolation with 10 known points

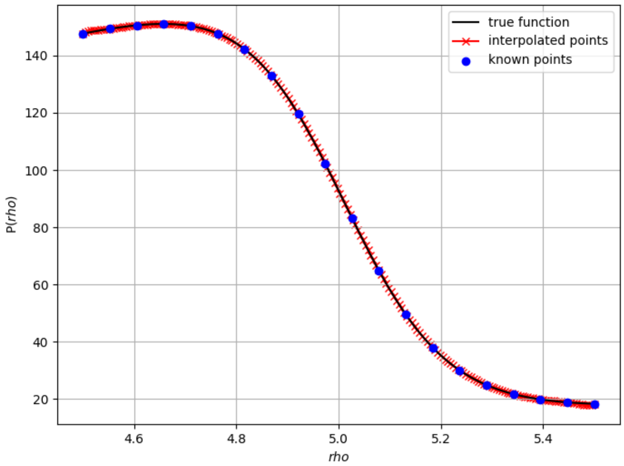

Abbildung 3: Interpolation with 20 known points

- for n=5 we see that the approximation overall isn't good, because we don't have enough known points. At the transition point, the interpolated function is close to linear but falls too early; therefore, the slope is less than the original function.
- for n=10 is the approximation at the transition point good but at the edges are errors
- for n=20 is the approximation even better, at the edges is just a slight difference to the original function.

4.2 III b

Now we want to do the same as in a but on the interval [0,10]

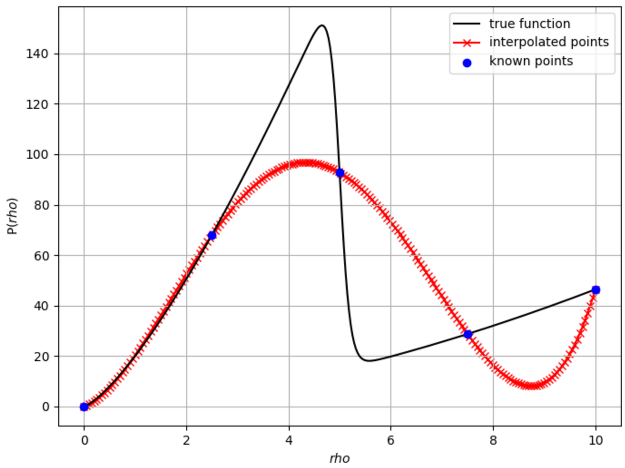

Abbildung 4: Interpolation with 5 known points

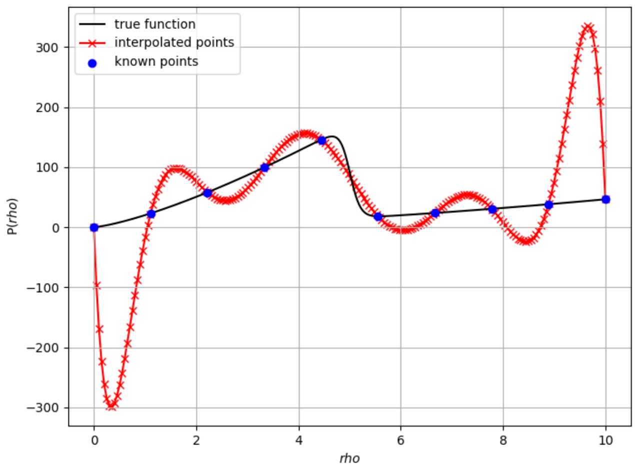

Abbildung 5: Interpolation with 10 known points

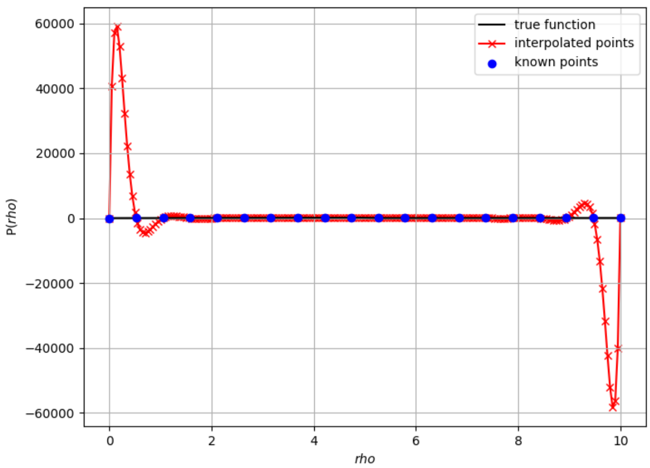

Abbildung 6: Interpolation with 20 known points

- for n=5 we see that the approximation is even worse because we have even fewer points around the transition point
- for n=10 is the approximation around the transition point close to the error for n=5 is (a). The error gets higher if we go to the edges.
- for n=20 we see that the errors at the edges get higher for more points we discuss. therefore the approximation only works close to the transition point but this is what we want.
- the true function graphs look different, this is due to the different scaling of the plot

4.3 Part III c

Now we want to calculate the chi-square error

$$E = \sqrt{\frac{1}{m} \cdot \sum_{k=1}^{m-1} (p(x_k) - y_k)^2}$$

the following code calculates the error for different numbers of known points

```
def ChiErrorPlot(rho_0, rho_1, r, n_values):
    error_log = []
    for n in n_values:
        xs = np.linspace(rho_0, rho_1, n)
        ys = [P(x) for x in xs]
        yk = [LagrangeInterpolation(x, xs, ys) for x in r]
        error = ChiError(r, P(r), yk)
        error_log.append(np.log(error))
    return error_log

def ChiError(r, true_values, interpolated_values):
    m = len(r)
    error = np.sqrt((1 / m) * np.sum((true_values -
                                    interpolated_values)**2))
    return error
```

now we want to plot the error for known points between 3 and 40, for different Intervalls.
we want to start with the intervall [4.5,5.5]

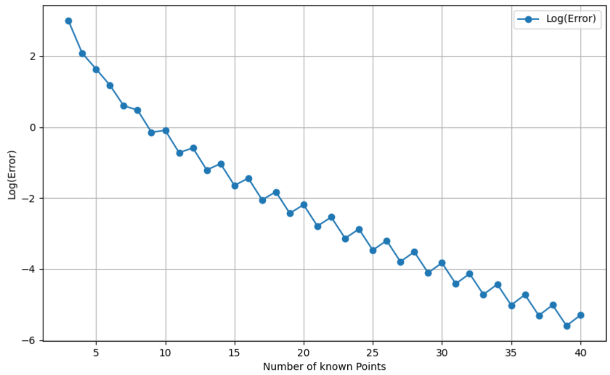

Abbildung 7: log(error) for the interval [4.5,5.5]

We see that the Log(Error) converges.
Now we want to look at the intervall [0,10]

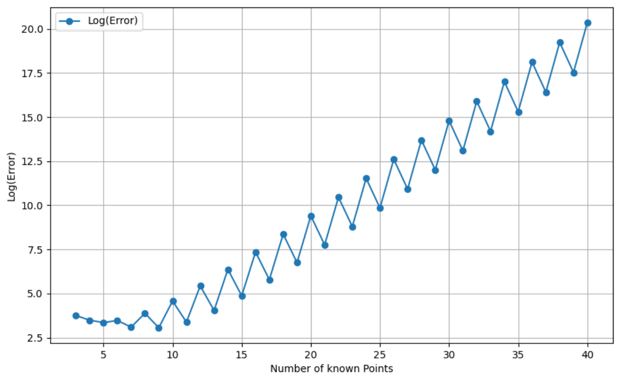

Abbildung 8: log(error) for the interval [0,10]

Here the log(Error) diverges
And at last we want to look at the error on the intervall [0,30]

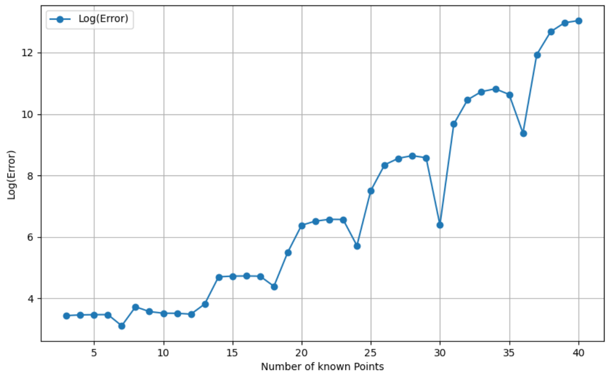

Abbildung 9: log(error) for the interval [0,30]

The log(error) also diverges, therefore we should use intervalls close to the transition point what we also see if we compare the graphics in (a) and (b).

4.4 III d

For the last part we want to change the parameters of our function $P(\rho)$, on a fixed interval, we learned from the tasks above that the intervals close to the transition point are the best so we want to choose the interval [4.5,5.5]. The most interesting parameter we can change is α which represents how well we approximate the Heaviside function.
First we set $\alpha = 7$

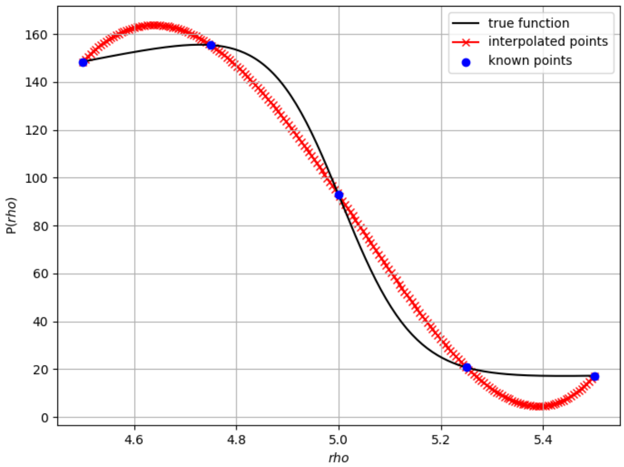

Abbildung 10: Interpolation with 5 known point and $\alpha = 7$

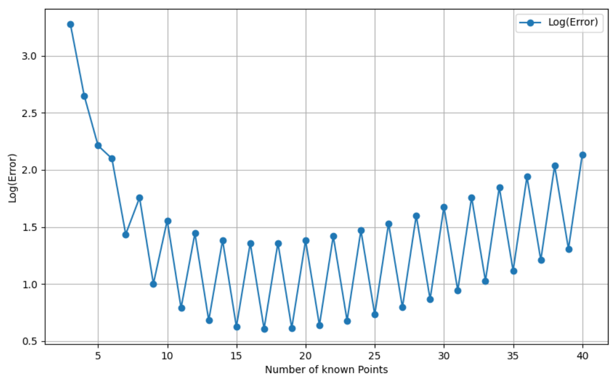

Abbildung 11: log(error) for the interval [4.5,5.5] and $\alpha = 7$

For the Interpolation plot, we don't see a big difference to $\alpha = 5$ but we see that for this α the error starts diverging, therefore for better Heaviside function approximations we need smaller and smaller intervals. If we choose the interval [4.8,5.2] we see

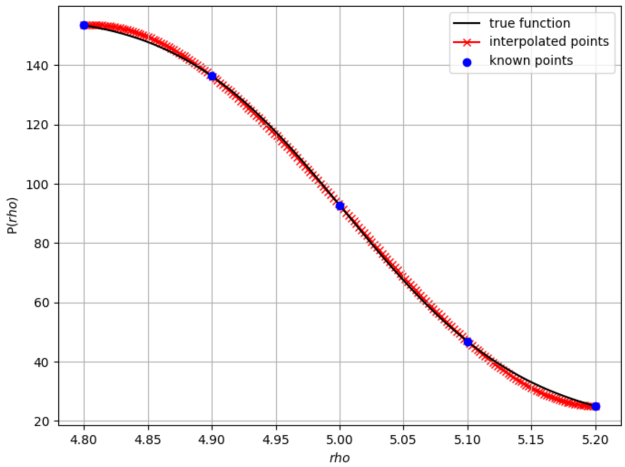

Abbildung 12: Interpolation with 5 known point and $\alpha = 7$ on smaller interval

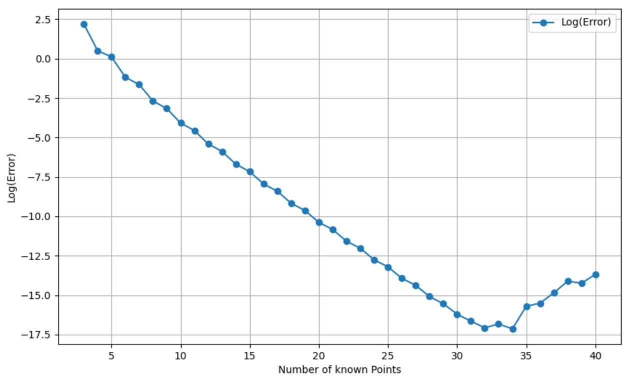

Abbildung 13: log(error) for the interval [4.8,5.2] and $\alpha = 7$

The number of known points until the error diverges gets bigger.

To see the divergence of bigger alpha we want to make a really good approximation of the Heaviside function with $alpha = 30$

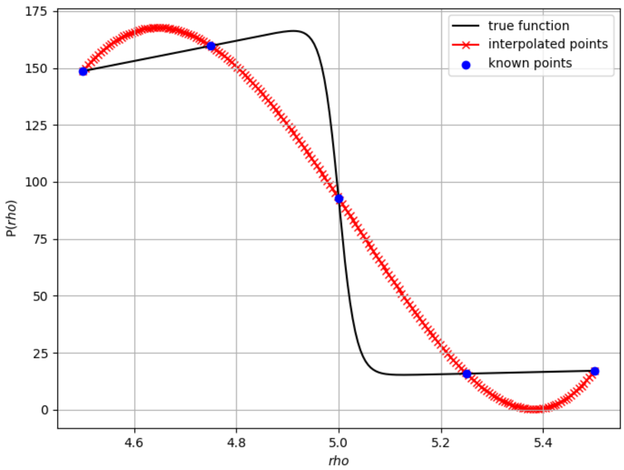

Abbildung 14: Interpolation with 5 known point and $\alpha = 30$ on smaller interval

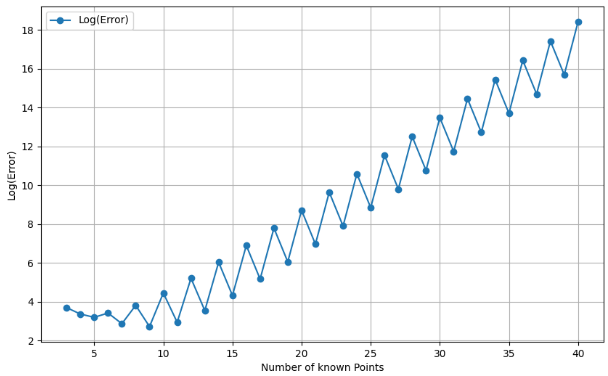

Abbildung 15: log(error) for the interval [4.5,5.5] and $\alpha = 30$

Here we can see the fast divergence of the error, and even at the plot for 5 known points, we can see a big error. Furthermore, the error will always diverge if we use a certain number of points, because the function at the transition will fit better but the errors ad the edges rise rapidly.