Assignment

Oszillation of a Fluidsphere

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: February 15, 2025

Contents

1 Theoretical derivation

1.1 Lane-Emden equation

Lets assume the following:

$$\frac{d\theta}{d\xi} = \frac{\phi}{\xi^2} \tag{1}$$

If we plug this in the Lane-Emden equation we get:

$$\frac{d\phi}{d\xi} = -\theta^n \cdot \xi^2 \tag{2}$$

We can now use this results to calculate the fluidsphere oscillation of our star.

1.2 Rewriting the master equation

We need to rewrite the following equation into am Matrix-eigenvalue problem so that we can solve it easily:

$$\frac{d^2\xi_r}{dr^2} + \left[\frac{2}{r} + \frac{1}{\Gamma_1 P}\frac{dP}{dr}\right]\frac{d\xi_r}{dr} + \left[\frac{\omega^2\rho}{\Gamma_1 P} - \frac{4}{r^2}\right]\xi_r = 0.$$

Via central differences we can rewrite the derivatives the following:

$$\frac{d^2\xi_r}{dr^2} = \frac{\xi_{i+1} - 2\xi_i + \xi_{i-1}}{h^2}$$

$$\frac{d\xi_r}{dr} = \frac{\xi_{i+1} - \xi_{i-1}}{2h}$$

$$\frac{dP}{dr} = \frac{P_{i+1} - P_{i-1}}{2h}$$

If we plug this in our master equation and separate the ω^2 terms we get:

$$\begin{aligned}
&\xi_{i+1}\cdot\left[\frac{1}{h^2} + \frac{2}{r_i} + \frac{1}{\Gamma_{1,i}P_i}\frac{P_{i+1}-P_{i-1}}{2h}\right] \\
&+\xi_{i-1}\cdot\left[\frac{1}{h^2} - \frac{2}{r_i} - \frac{1}{\Gamma_{1,i}P_i}\frac{P_{i+1}-P_{i-1}}{2h}\right] \\
&-\xi_i\left[\frac{2}{h^2} + \frac{4}{r_i^2}\right] = -\frac{\omega^2\rho_i}{\Gamma_{1,i}P_i}\xi_i
\end{aligned}$$

If we do this for every i. We get the following matrix equation:

$$A\xi = \omega^2 B\xi \tag{3}$$

which we can solve with our given methods for ω^2.

2 Code for Lame-Emden

To get the solution of the Lane-Emden equation we use the following code with Runke-Kutta 4th order:

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

# Lane-Emden equation
def Lane_Emden_system(y, xi, n):
    theta, dtheta_dxi = y
    if xi == 0:
        return np.array([dtheta_dxi, 0])  # If we divide by
                                            zero
    return np.array([dtheta_dxi, - (2 / xi) * dtheta_dxi -
                                    theta ** n])

# Runge-Kutta 4 (RK4)
def rk4(System, y0, xi_max, h, n):
    xi_values = np.arange(1E-4, xi_max, h)
    y = np.zeros((len(xi_values), len(y0)))
    y[0] = y0

    for i in range(1, len(xi_values)):
        xi = xi_values[i - 1]
        k1 = System(y[i - 1], xi, n)
        k2 = System(y[i - 1] + h * k1 / 2, xi + h / 2, n)
        k3 = System(y[i - 1] + h * k2 / 2, xi + h / 2, n)
        k4 = System(y[i - 1] + h * k3, xi + h, n)
        y[i] = y[i - 1] + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6

    return xi_values, y[:, 0], y[:, 1]

# BC
y0 = np.array([1, 0])
xi_max = 10
h = 0.001

for n in [0, 1, 2, 3, 4]:
    xi_vals, theta_vals, dtheta_vals = rk4(Lane_Emden_system,
                                        y0, xi_max, h, n)
```

```
    # Plot
    plt.figure(figsize=(6, 4))
    plt.plot(xi_vals, theta_vals, label=f'Lane-Emden Solution
                                        (n={n})')
    plt.xlabel("Radius")
    plt.ylabel("Density")
    plt.legend()
    plt.title(f"Lane-Emden equation with RK4 (n={n})")
    plt.grid()


    plt.savefig(f"Lane_Emden_n{n}.png", dpi=300)
    plt.close()
```

3 Solutions for different polytropic indexes

We solved the equation for different indexes. As a result we get:

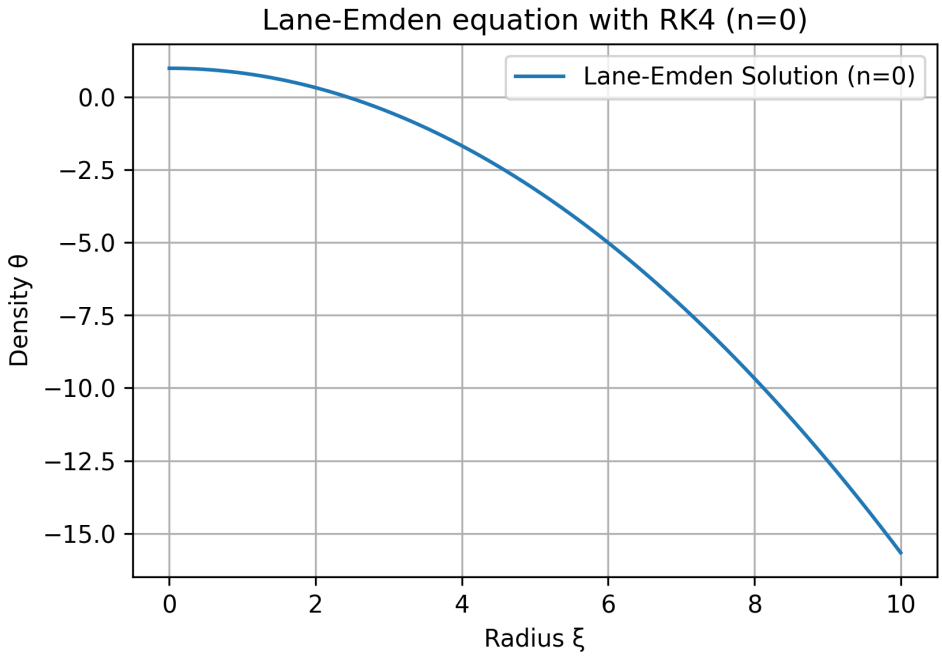

Figure 1: $n = 0$

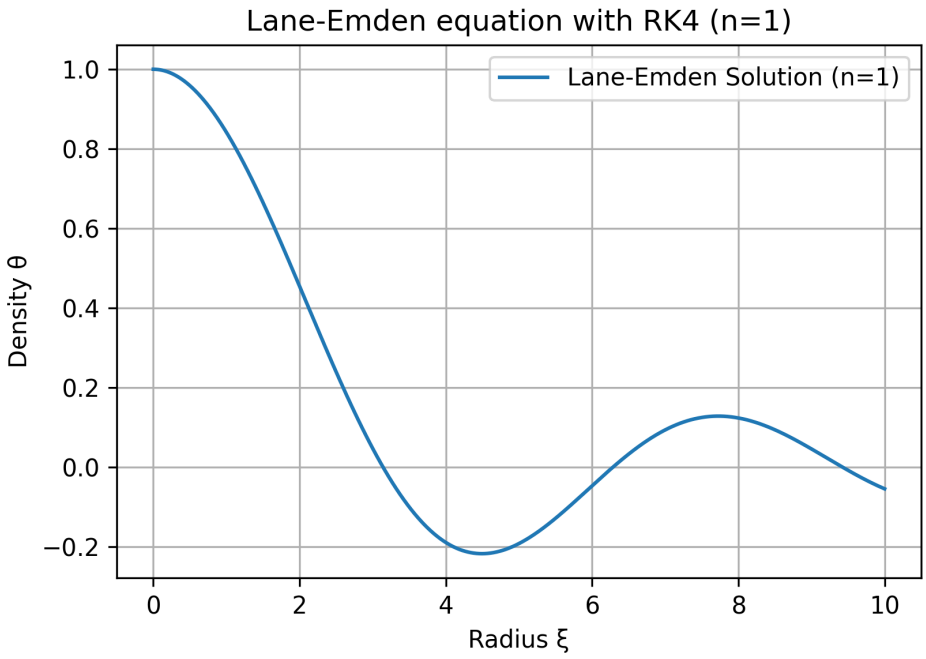
Lane-Emden equation with RK4 (n=1)
Lane-Emden Solution (n=1)
Density θ
Radius ξ
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8
10

Figure 2: $n = 1$

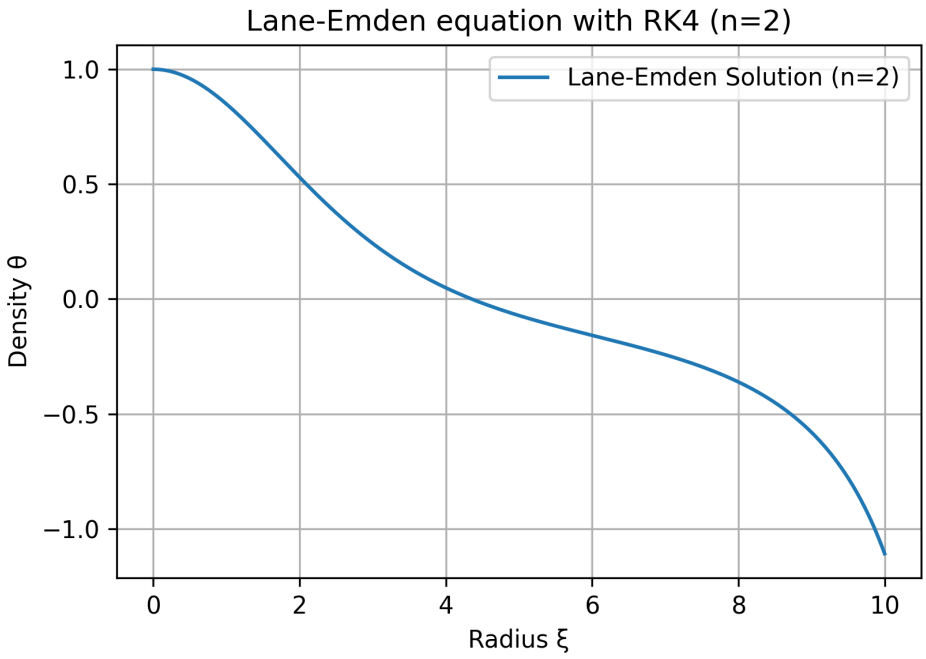
Lane-Emden equation with RK4 (n=2)
Lane-Emden Solution (n=2)
Density θ
Radius ξ
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10

Figure 3: $n = 2$

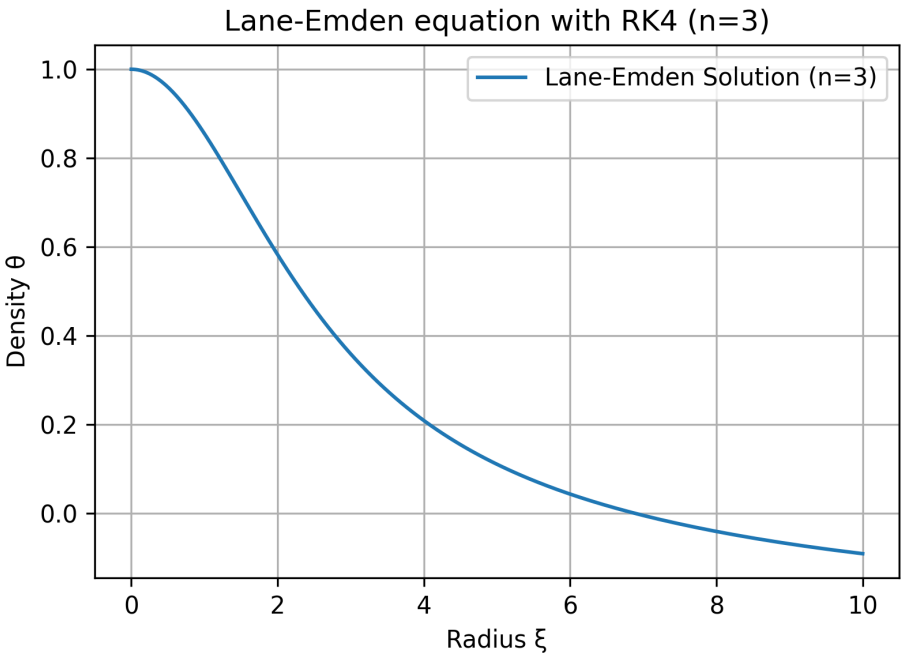
Lane-Emden equation with RK4 (n=3)
Lane-Emden Solution (n=3)
Density θ
Radius ξ
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

Figure 4: $n = 3$

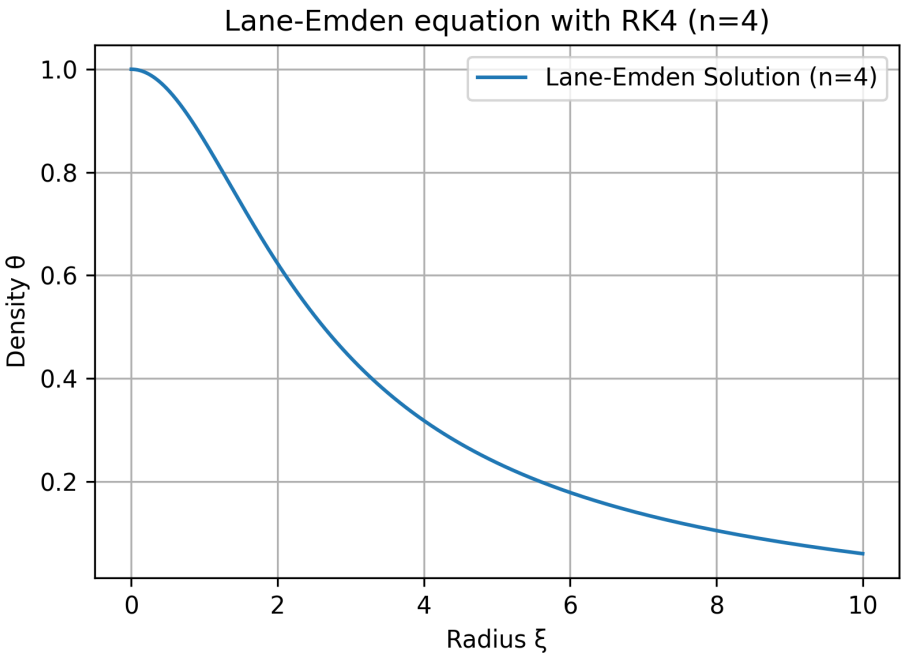
Lane-Emden equation with RK4 (n=4)
Lane-Emden Solution (n=4)
Density θ
Radius ξ
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8
10

Figure 5: $n = 4$

4 Eigenvalues of the fluidsphere

Based on the equation we get in the theortical chapter we can use the following code to calculate the eigenvalues of our system:

```
import numpy as np
import scipy.linalg as linalg

# Lane-Emden equation
def Lane_Emden_system(y, xi, n):
    theta, dtheta_dxi = y
    if xi == 0:
        return np.array([dtheta_dxi, 0])  # Avoid division by
                                            zero at xi=0
    return np.array([dtheta_dxi, - (2 / xi) * dtheta_dxi - np
                                    .maximum(theta,0) ** n])

# Runge-Kutta 4th order based on the code we used in the
                                  Lorentz attractor part
def rk4(System, y0,xi_min, xi_max, h, n):
    xi_values = np.arange(xi_min, xi_max, h)
    y = np.zeros((len(xi_values), len(y0)))
    y[0] = y0

    for i in range(1, len(xi_values)):
        xi = xi_values[i - 1]
        k1 = System(y[i - 1], xi, n)
        k2 = System(y[i - 1] + h * k1 / 2, xi + h / 2, n)
        k3 = System(y[i - 1] + h * k2 / 2, xi + h / 2, n)
        k4 = System(y[i - 1] + h * k3, xi + h, n)
        y[i] = y[i - 1] + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6

    return xi_values, y[:, 0], y[:, 1]

# Initial conditions
y0 = np.array([1, 0])
xi_max = 10
xi_min = 1e-4
h = 1
n = (xi_max-xi_min)/h

xi_vals, theta_vals, dtheta_vals = rk4(Lane_Emden_system, y0,
                                  xi_min, xi_max, h, n)
```

```
# Densities and pressures calculation
def get_values_dens_press(theta_vals, n, dens_c=1, K=1):
    dens_vals = dens_c * np.maximum(theta_vals, 0)**n
    press_vals = K * dens_vals**(1 + 1/n)
    return dens_vals, press_vals

dens_vals, press_vals = get_values_dens_press(theta_vals, n)

# Ensure density values are positive for log computation
dens_vals = np.where(dens_vals > 0, dens_vals, 1e-10)
press_vals = np.where(press_vals > 0, press_vals, 1e-10)

# Calculation of k(r)
def k_values(r, Gamma_1, P, h):
    k = np.zeros(len(r)-1)  # Initialize array with correct
                                size
    for i in range(1, len(r)-1):
        k[i] = 2/r[i] + 1/(Gamma_1[i] * P[i]) * (P[i+1] - P[i
                                -1]) / (2*h)
    return k

def gamma_values(r, P, dens_val):
    gamma = np.zeros(len(r))

    for i in range(1, len(r)-1):
        delta_dens = np.log(np.maximum(dens_val[i+1], 1e-10))
                                 - np.log(np.maximum(
                                dens_val[i-1], 1e-10))
        delta_P = np.log(np.maximum(P[i+1], 1e-10)) - np.log(
                                np.maximum(P[i-1], 1e-
                                10))
        gamma[i] = delta_P / delta_dens if delta_dens != 0
                                else 1  # Avoid
                                dividing by small
                                values

    return gamma

# Calculation of matrices A and B for the oscillatory
                            equation
def compute_AB(xi_vals, press_vals, dens_vals, h):
    N = len(xi_vals)
    A = np.zeros((N-2, N-2))
    B = np.zeros((N-2, N-2))
```

```
    # Calculate inner points
    gamma1 = gamma_values(xi_vals,press_vals,dens_vals)
    k = k_values(xi_vals, gamma1, press_vals, h)


    for i in range(1, N-2):
        r = xi_vals[i]

        if i-2 >= 0:
            A[i-1, i-2] = 1/h**2 - k[i] / (2*h)

        A[i-1, i-1] = -2/h**2 - 4/r**2

        if i < N-2-1:
            A[i-1, i] = 1/h**2 + k[i] / (2*h)

        B[i-1, i-1] = dens_vals[i] / (gamma1[i] * press_vals[
                                        i])



    A[0, 0] = 1  # Regularit t am Zentrum:  r(0) = 0
    A[-1, -2] = -1/h
    A[-1, -1] = A[-1, -1] = 1/h + dens_vals[-1] / (press_vals
                                    [-1] + 1e-10)
    B[-1, -1] = 1e-10

    return A, B


A, B = compute_AB(xi_vals, press_vals, dens_vals, h)

try:
    w,v = linalg.eig(A, B)
except linalg.LinAlgError:
    print("Error in eigenvalue computation")

if len(w) == 0:
    print("No eigenvalues found")
    exit()
else:
    for i in range(len(w)):
        print(f"Eigenvalue {i+1}: {np.real(w[i])}")
```

As a result we get for ω^2:

$$\begin{aligned}
&\text{Eigenvalue } 1 = 1.5 \times 10^{10}\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 2 = -1.2027408507864075\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 3 = -1.2027408507864075\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 4 = -3.9075948763623356\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 5 = -3.1775881891692763\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 6 = -2.187346186522438\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 7 = -1.201958281860447\,\mathrm{s}^{-2}\\
&\text{Eigenvalue } 8 = -0.44283234130692356\,\mathrm{s}^{-2}
\end{aligned}$$

5 Sources

Wikipedia Lame-Emden equation