

## Große TODO Liste: gezeitenreibung\_final.ipynb

gezeitenreibung\_final.ipynb möglichst nicht ändern, sondern nur die Datei gezeitenreibung.ipynb bearbeiten. Nur fertige komplette Sachen in gezeitenreibung\_final.ipynb kopieren.

TODOS:

--

Erst wenn oben alles fertig ist:

Zellen von gezeitenreibung.ipynb in gezeitenreibung\_final.ipynb kopieren.  $\backslashnewline$  Baryzentrum in barycenter umbenennen, das wird aber im Augenblick zu verstreut verwendet.

$\backslashprovidecommand{\e}[1]{\ensuremath{\cdot 10^{\#1}}}$   $\backslashprovidecommand{\fehlt}{\textcolor{red}{Fehlt!}}$   $\backslashdots$   
 $\backslashprovidecommand{\todo}{\textcolor{red}{\huge{ToDo}}}$

```
In [ ]: mErde, mOzean, RErde, mMond, rMondBahn, TMondBahn, G, TErdRotation, tau = 5.9721986*10**24, 0.0014*10**24, 6.3675*10**6, 7.345

import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import pi
import scipy
from scipy.integrate import solve_ivp
from scipy.integrate._ivp.ivp import OdeResult # : ) https://github.com/scipy/scipy/blob/main/scipy/integrate/_ivp/ivp.py
from matplotlib.animation import FuncAnimation
from IPython.display import Image #to display animations
from matplotlib import cm

# Beautiful plots
plt.rcParams['text.usetex'] = True
plt.rcParams['font.family'] = "serif"
plt.rcParams['font.serif'] = "Computer Modern Roman"

# Useful StackPosts
# https://stackoverflow.com/questions/59634279/solve-ivp-error-required-step-size-is-less-than-spacing-between-numbers

# Executing everything all the time takes too long
fastExecution = True
```

```
In [ ]: # NICHT ANFASSEN! WENN, DANN UNBEDINGT IN FINAL AUCH ÄNDERN!
def baryzentrum(m1, m2, r): return r*m2/(m1+m2)

def iv_stable_orbit_2body():
    abstand_baryzentrum_erde = baryzentrum(mErde, mMond, rMondBahn)
    return [-abstand_baryzentrum_erde, 0], [rMondBahn-abstand_baryzentrum_erde, 0], [0, -2*pi*abstand_baryzentrum_erde/TMondB

def eq_motion_2body(t, state, mass):
    x_1, y_1, x_2, y_2, vx_1, vx_2, vy_2 = state
    dist_em = ((x_2 - x_1)**2 + (y_2 - y_1)**2 )**0.5
    ax_1 = G * mass[1] / (dist_em**3) * (x_2 - x_1)
    ay_1 = G * mass[1] / (dist_em**3) * (y_2 - y_1)
    ax_2 = G * mass[0] / (dist_em**3) * (x_1 - x_2)
    ay_2 = G * mass[0] / (dist_em**3) * (y_1 - y_2)
    return np.array([vx_1, vy_1, vx_2, vy_2, ax_1, ay_1, ax_2, ay_2])

def two_body_problem(pos_body_1: list, pos_body_2: list, vel_body_1: list, vel_body_2: list, mass: list, t_max: float, steps=1
    solution = solve_ivp(eq_motion_2body, [t_start, t_max], [*pos_body_1, *pos_body_2, *vel_body_1, *vel_body_2], args=(mass,))
    x1, y1, x2, y2, v_x1, v_y1, v_x2, v_y2 = solution.y
    return [solution.t, x1, y1, x2, y2]

def tide_acceleration(x_M, y_M, x_i, y_i, phi_i, ax_E, ay_E, omega_i, m_M):
    part_a, part_b = G*m_M* ((y_M - y_i)*np.cos(phi_i) - (x_M - x_i)*np.sin(phi_i)) / ((x_M - x_i)**2 + (y_M - y_i)**2)**(3/2)
    ax_i, ay_i = -part_a*np.sin(phi_i) - part_b*np.cos(phi_i), part_a*np.cos(phi_i) - part_b*np.sin(phi_i)
    return -(ax_i - ax_E)*np.sin(phi_i) + (ay_i - ay_E)*np.cos(phi_i)/REerde

def eq_motion_4body(t, state, mass, calculate_tides=True):
    x_E, y_E, x_M, y_M, phi_1, phi_2, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2 = state
    m_E, m_M, m_1, m_2 = mass
    x_1, y_1, x_2, y_2 = x_E + REerde*np.cos(phi_1), y_E + REerde*np.sin(phi_1), x_E + REerde*np.cos(phi_2), y_E + REerde*np.sin(phi_2)
    mu_x = m_E + m_1*np.cos(phi_1)**2 + m_2*np.cos(phi_2)**2
    mu_y = m_E + m_1*np.sin(phi_1)**2 + m_2*np.sin(phi_2)**2
    a = (G*m_M*m_E)/((x_M - x_E)**2 + (y_M - y_E)**2)**(3/2)
    b = m_1*np.sin(phi_1)*np.cos(phi_1) + m_2*np.sin(phi_2)*np.cos(phi_2)
    c1 = m_1*REerde*omega_1**2 + (G*m_M*m_1*((x_M - x_1)*np.cos(phi_1) + (y_M - y_1)*np.sin(phi_1)))/((x_M - x_1)**2 + (y_M - y_1)**2)
    c2 = m_2*REerde*omega_2**2 + (G*m_M*m_2*((x_M - x_2)*np.cos(phi_2) + (y_M - y_2)*np.sin(phi_2)))/((x_M - x_2)**2 + (y_M - y_2)**2)
    ax_E = 1/(mu_x - b**2/mu_y) * (a*(x_M - x_E) - b/mu_y*(a*(y_M - y_E) + c1*np.sin(phi_1) + c2*np.sin(phi_2)) + c1*np.cos(phi_1))
    ay_E = 1/mu_y*(a*(y_M - y_E) - b*ax_E + c1*np.sin(phi_1) + c2*np.sin(phi_2))
    ax_M = G*((m_E*(x_E - x_M))/((x_E - x_M)**2 + (y_E - y_M)**2)**(3/2) + (m_1*(x_1 - x_M))/((x_1 - x_M)**2 + (y_1 - y_M)**2))
    ay_M = G*((m_E*(y_E - y_M))/((x_E - x_M)**2 + (y_E - y_M)**2)**(3/2) + (m_1*(y_1 - y_M))/((x_1 - x_M)**2 + (y_1 - y_M)**2))
    if not calculate_tides: return [vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, ax_E, ay_E, ax_M, ay_M, 0, 0] # For code reuse w
    alpha_1, alpha_2 = tide_acceleration(x_M, y_M, x_1, y_1, phi_1, ax_E, ay_E, omega_1, m_M), tide_acceleration(x_M, y_M, x_2, y_2, phi_2, ax_E, ay_E, omega_2, m_M)
    return[vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, ax_E, ay_E, ax_M, ay_M, alpha_1, alpha_2]

def center_of_mass(mass, position_x, position_y):
    return [np.sum(mass*position_x)/np.sum(mass), np.sum(mass*position_y)/np.sum(mass)]

def iv_stable_orbit_4body():
    mass = [mErde, mMond, mOzean/2, mOzean/2] # Each of the tides gets half of the ocean's mass
    abstand_baryzentrum_erde = center_of_mass(mass, np.array([0, rMondBahn, -REerde, REerde]), np.array([0, 0, 0, 0]))[0]
    x0_Eerde = [-abstand_baryzentrum_erde, 0]
    x0_Mond = [rMondBahn - abstand_baryzentrum_erde, 0]
    vy0_Eerde = 2*pi*abstand_baryzentrum_erde/TMondBahn
    vy0_Mond = 2*pi*(rMondBahn-abstand_baryzentrum_erde)/TMondBahn
    v0_Eerde = [0, -vy0_Eerde]
    v0_Mond = [0, vy0_Mond]
    phi_1, phi_2 = 0, pi
    omega_1, omega_2 = 2*pi/TMondBahn, 2*pi/TMondBahn
    return [x0_Eerde, x0_Mond, [phi_1, phi_2], v0_Eerde, v0_Mond, [omega_1, omega_2], mass]

def four_body_problem(pos_body_1: list, pos_body_2: list, phi_i: list, vel_body_1: list, vel_body_2: list, omega_i: list, mass
    solution = solve_ivp(fun=fun, t_span=[t_start, t_max], y0=[*pos_body_1, *pos_body_2, *phi_i, *vel_body_1, *vel_body_2, *om
    x_E, y_E, x_M, y_M, phi_1, phi_2, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2 = solution.y
    return [solution.t, x_E, y_E, x_M, y_M, phi_1, phi_2, omega_1, omega_2]
```

## Zweikörperproblem

```
In [ ]: # Calculate the solution of the two body problem
t, x_1, y_1, x_2, y_2 = two_body_problem(*iv_stable_orbit_2body(), 3*TMondBahn)

/home/finn/.local/lib/python3.10/site-packages/scipy/integrate/_ivp/common.py:39: UserWarning: The following arguments have no
effect for a chosen solver: `steps`.
warn("The following arguments have no effect for a chosen solver: {}."
```

- Vergleichen Sie alle Ergebnisse auch mit der exakten Lösung des Zweikörperproblems.

Vergleich mit der exakten Lösung des Zweikörperproblems:

Eine exakte analytische Lösung des Zweikörperproblems ist mit genügend Annahmen möglich. Dazu wird im Schwerpunktssystem gearbeitet. Aus den Bewegungsgleichungen der Körper die nur mit der Schwerkraft wechselwirken, kann man folgern, dass der Schwerpunkt des Systems eine gleichförmige Bewegung ausführt. Mit unseren Anfangsbedingungen bewegt dieser sich aber nicht.

Die beiden Körper können sich um ihren Schwerpunkt je nach Anfangsbedingungen auf verschiedenen Bahnen bewegen. Dies lässt sich durch die Exzentrizität beschreiben. Mögliche Bahnen sind Kreise, Ellipsen, Parabeln und Hyperbeln. Wir haben die Anfangsbedingungen so gewählt, dass die Bahn eine Kreisbahn ist.   
 Unsere Ergebnisse stimmen also mit der exakten Lösung des Zweikörperproblems überein, da wir zwei geschlossene Kreisbahnen um den gemeinsamen Schwerpunkt (mit unseren Anfangsbedingungen der Ursprung) erhalten. Diese Kreisbahnen lassen sich mit einfachen trigonometrischen Funktionen beschreiben.

$$\mathbf{r}_E(t) = \begin{pmatrix} x_E(t) \\ y_E(t) \end{pmatrix} = \begin{pmatrix} r_{E_0} \cos(\omega_E t) \\ r_{E_0} \sin(\omega_E t) \end{pmatrix} \quad (1)$$

$$\mathbf{r}_M(t) = \begin{pmatrix} x_M(t) \\ y_M(t) \end{pmatrix} = \begin{pmatrix} r_{M_0} \cos(\omega_M t) \\ r_{M_0} \sin(\omega_M t) \end{pmatrix} \quad (2)$$

Diese lassen sich ebenfalls leicht plotten.

Unser numerischer Ansatz ist natürlich in der Lage alle möglichen Bahnen zu berechnen, also auch Ellipsen und Hyperbeln, es ließe sich also auch ein Komet oder ein Sling-Shot-Manöver berechnen. Mit den geeigneten Anfangsbedingungen ließen sich diese auch analytisch berechnen.

Eine weitere Möglichkeit unsere Ergebnisse zu überprüfen ist das dritte Keplersche Gesetz, es besagt, dass die Umlaufzeit  $T$  des Zweikörpersystems beträgt:

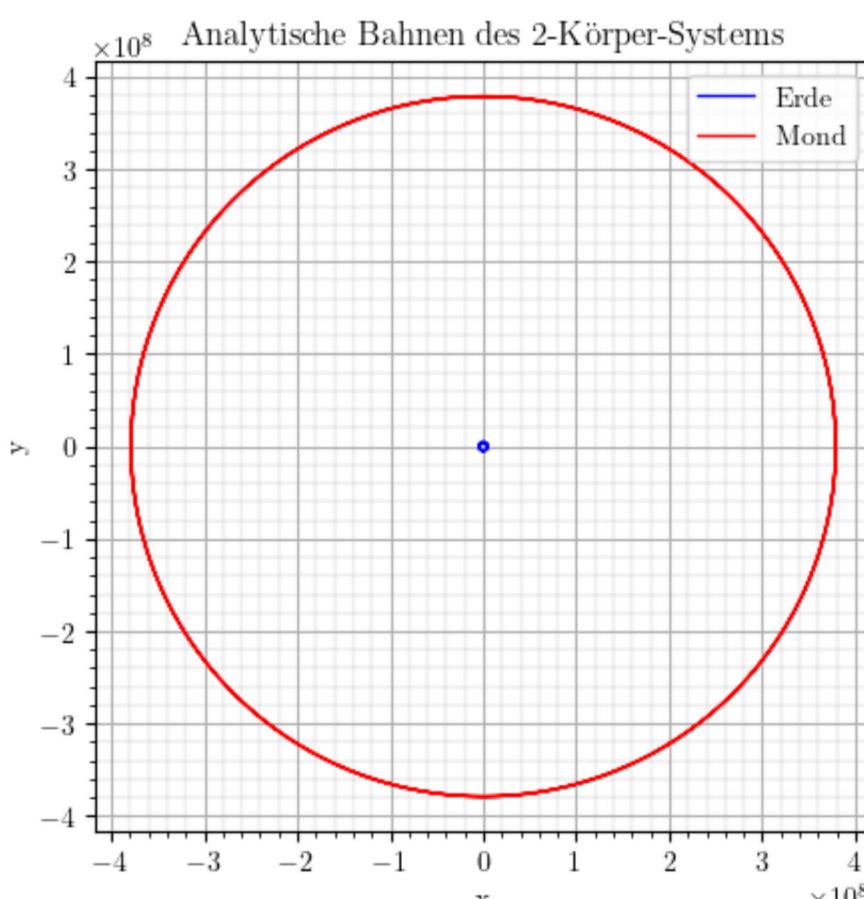
$$T^2 = \frac{4\pi^2 \cdot a^3}{G \cdot M}$$

Mit  $G$  als Gravitationskonstante,  $M = m_{Erde} + m_{Mond}$  und  $a$  als großer Halbachse welche in unserem genäherten Fall der Kreisbahn  $a = r_{MondBahn}$  entspricht.

```
In [ ]: r_E0, r_M0 = iv_stable_orbit_2body()[0][0], iv_stable_orbit_2body()[1][0]
time_points = np.linspace(0, 3*TMondBahn, 1000)
x_E, y_E = np.vectorize(lambda t: r_E0 * np.cos(2*pi/TMondBahn * t))(time_points), np.vectorize(lambda t: r_E0 * np.sin(2*pi/TMondBahn * t))(time_points)
x_M, y_M = np.vectorize(lambda t: r_M0 * np.cos(2*pi/TMondBahn * t))(time_points), np.vectorize(lambda t: r_M0 * np.sin(2*pi/TMondBahn * t))(time_points)

# Plot the analytical solution with our boundary conditions
plt.rc('axes', axisbelow=True)
plt.figure(figsize=(5, 5))
plt.title("Analytische Bahnen des 2-Körper-Systems")
plt.grid('minor', 'minor', linestyle='-', linewidth=0.2)
plt.grid('major', 'major', linestyle='-', linewidth=0.8)
plt.minorticks_on()
plt.xlabel("x")
plt.ylabel("y")
plt.plot(x_E, y_E, 'b', linewidth=1, label='Erde')
plt.plot(x_M, y_M, 'r', linewidth=1, label='Mond')
plt.legend()
plt.show()

print(f'3. Keplersches Gesetze T: {TMondBahn} = {(4*pi**2*rMondBahn**3/(G*(mErde+mMond)))**(1/2)}')
```



3. Keplersches Gesetze T: 2360591.54496 = 2350028.5085322363

## Vierkörperproblem mit Randbedingungen, intrinsischer Rotation und Reibung

Im Folgenden wollen wir das 4-Körper-Problem um die Reibung des Ozeans (der Flutberge) mit der Erde erweitern. Dies verlangsamt die intrinsische Erdrotation und sorgt für länger werdende Tage und eine größere Umlaufbahn des Mondes. Um dies zu beschreiben müssen wir mehrere zusätzliche Komponenten herleiten und mit ihnen unser Modell erweitern.

Die Reibung beschreiben wir zunächst mit der Reibungskonstante  $k = 2 \cdot 10^{-12} \frac{1}{\text{m}}$ .

### Herleitung der zusätzlichen Bewegungsgleichung und der Reibungskraft

Wir wollen als erstes die Reibungskraft zwischen der Erde und den Flutbergen bestimmen. Wir wählen dafür folgenden Ansatz:

$\vec{F}_{R,i}(\vec{v}_i) = -km_i|\vec{v}_i|\vec{v}_i$ . Hierbei ist  $\vec{v}_i$  die Geschwindigkeit des  $i$ -ten Flutbergs relativ zur rotierenden Erdoberfläche und  $k$  eine effektive Reibungskonstante.

Die Geschwindigkeit des Flutbergs bestimmen wir über seine Winkelgeschwindigkeit relativ zur Erdoberfläche ( $\omega_E - \omega_i$ ), mit der er sich um die z-Achse dreht. Hierbei ist  $\omega_E$  die Winkelgeschwindigkeit der Eigenrotation der Erde und  $\omega_i$  die Winkelgeschwindigkeit des Flutbergs.

Die Geschwindigkeit ist allgemein definiert als  $\vec{v}_i = \vec{\omega}_i \times \vec{r}_i$ :

$$\vec{v}_i = (\omega_E - \omega_i) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \times r_E \cdot \vec{e}_{r,i} = \begin{pmatrix} 0 \\ 0 \\ \omega_E - \omega_i \end{pmatrix} \times \begin{pmatrix} r_E \cdot \cos(\phi_i) \\ r_E \cdot \sin(\phi_i) \\ 0 \end{pmatrix} = \begin{pmatrix} -r_E \cdot \sin(\phi_i) \cdot (\omega_E - \omega_i) \\ r_E \cdot \cos(\phi_i) \cdot (\omega_E - \omega_i) \\ 0 \end{pmatrix}$$

Damit berechnen wir den Betrag:

$$|\vec{v}_i| = \sqrt{r_E^2 \cdot (\omega_E - \omega_i)^2 \cdot \sin^2(\phi_i) + r_E^2 \cdot (\omega_E - \omega_i)^2 \cdot \cos^2(\phi_i)} = r_E \cdot |\omega_E - \omega_i|$$

Also ist die Reibungskraft:

$$\vec{F}_{R,i}(\vec{v}_i) = -km_i|\vec{v}_i|\vec{v}_i = -km_i r_E \cdot |\omega_E - \omega_i| \cdot \begin{pmatrix} -r_E \cdot \sin(\phi_i) \cdot (\omega_E - \omega_i) \\ r_E \cdot \cos(\phi_i) \cdot (\omega_E - \omega_i) \\ 0 \end{pmatrix} = -km_i r_E^2 \cdot |\omega_E - \omega_i| (\omega_E - \omega_i) \cdot \begin{pmatrix} -\sin(\phi_i) \\ \cos(\phi_i) \\ 0 \end{pmatrix}$$

Durch teilen durch die Masse des  $i$ -ten Flutbergs erhalten wir die Beschleunigung, die wir auf die Differentialgleichung des  $i$ -ten Flutbergs aus dem 4-Körper-Problem addieren.

Als nächstes wollen wir die Differentialgleichung für die intrinsische Rotation der Erde herleiten. Es gilt die Drehimpulsrelation  $L = I \cdot \omega$ . Wir nehmen die Erde als Kugel mit homogener Massenverteilung an, damit hat sie ein Trägheitsmoment von  $I = \frac{2}{5}m_E r_E^2$ . Für Drehimpuls und Drehmoment folgt:

$$\begin{aligned} \vec{L} &= \frac{2}{5}m_E r_E^2 \vec{\omega}_E \\ \vec{M} &= \frac{d\vec{L}}{dt} = \frac{2}{5}m_E r_E^2 \vec{\dot{\omega}}_E \end{aligned}$$

Das Drehmoment welches durch die Reibungskräfte entsteht ist:

$$\vec{M}_R = \sum_{i=1}^2 \vec{r}_i \times \vec{F}_{R,i}$$

Dies berechnet sich für zwei Flutberge zu:

$$\vec{M}_R = \begin{pmatrix} 0 \\ 0 \\ -kr_E^3 (m_1 \cdot |\omega_E - \omega_1| (\omega_E - \omega_1) + m_2 \cdot |\omega_E - \omega_2| (\omega_E - \omega_2)) \end{pmatrix}$$

Setzen wir die z-Komponenten der Drehmomente gleich (nur diese sind  $\neq 0$ ), erhalten wir folgende Bewegungsgleichung für die intrinsische Rotation der Erde:

$$\begin{aligned} \frac{2}{5}m_E r_E^2 \dot{\omega}_E &= -kr_E^3 (m_1 \cdot |\omega_E - \omega_1| (\omega_E - \omega_1) + m_2 \cdot |\omega_E - \omega_2| (\omega_E - \omega_2)) \\ \dot{\omega}_E &= -\frac{5kr_E}{2m_E} (m_1 \cdot |\omega_E - \omega_1| (\omega_E - \omega_1) + m_2 \cdot |\omega_E - \omega_2| (\omega_E - \omega_2)) \end{aligned}$$

Um dies im Code umzusetzen verwenden wir, wenn es geht, die schon implementierten Funktionen. Daher wird hier der selbe Ansatz wie zuvor gewählt, nur die Beschleunigung auf die Flutberge wird wie oben erklärt erweitert, und die zusätzliche Differentialgleichung wird hinzugefügt.

```
In [ ]: def iv_stable_orbit_4body_friction():
    '''Initial conditions for the Earth-Moon-Tides system in a stable orbit'''
    x0_Eerde, x0_Mond, phi_i, v0_Eerde, v0_Mond, omega_i, mass = iv_stable_orbit_4body()
    phi0_E = 0 # The starting point of the earth's rotation is unconsequentially defined as 0
    omega0_E = 2*pi / TErdRotation # Angular velocity of the earth
    return [x0_Eerde, x0_Mond, phi_i, phi0_E, v0_Eerde, v0_Mond, omega_i, omega0_E, mass]

def tides_acceleration_friction(x_M, y_M, x_i, y_i, phi_i, ax_E, ay_E, omega_i, omega_E, m_M, k):
    '''Calculates the angular acceleration of a tide, adding the frictional force caused by the earth's rotation'''
    ax_friction = k * Rerde**2 * np.abs(omega_i - omega_E) * (omega_i - omega_E) * np.sin(phi_i)
    ay_friction = -k * Rerde**2 * np.abs(omega_i - omega_E) * (omega_i - omega_E) * np.cos(phi_i)
    part_a = G*m_M* ((y_M - y_i)*np.cos(phi_i) - (x_M - x_i)*np.sin(phi_i)) / ((x_M - x_i)**2 + (y_M - y_i)**2)**(3/2)
    part_b = (REerde*omega_i**2 - (ax_E*np.cos(phi_i) + ay_E*np.sin(phi_i)))
    ax_i = -part_a*np.sin(phi_i) - part_b*np.cos(phi_i) + ax_friction
    ay_i = part_a*np.cos(phi_i) - part_b*np.sin(phi_i) + ay_friction

    alpha_i = (-(ax_i - ax_E)*np.sin(phi_i) + (ay_i - ay_E)*np.cos(phi_i))/REerde
    return alpha_i

def eq_motion_4body_friction(t, state, mass, k):
    x_E, y_E, x_M, y_M, phi_1, phi_2, phi_E, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, omega_E = state
    x_1, y_1, x_2, y_2 = x_E + Rerde*np.cos(phi_1), y_E + Rerde*np.sin(phi_1), x_E + Rerde*np.cos(phi_2), y_E + Rerde*np.sin(phi_2)
    m_E, m_M, m_1, m_2 = mass

    # Reuse the equations of motion from the 4-body problem for the earth and the moon:
    _, _, _, _, _, ax_E, ay_E, ax_M, ay_M, _, _ = eq_motion_4body(t, np.array([x_E, y_E, x_M, y_M, phi_1, phi_2, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, omega_E]), mass)

    # Tides angular acceleration with friction:
    alpha_1 = tides_acceleration_friction(x_M, y_M, x_1, y_1, phi_1, ax_E, ay_E, omega_1, omega_E, m_M, k)
    alpha_2 = tides_acceleration_friction(x_M, y_M, x_2, y_2, phi_2, ax_E, ay_E, omega_2, omega_E, m_M, k)

    # The earth's angular acceleration:
    alpha_E = (5*k*REerde)/(2*m_E) * (m_1*np.abs(omega_1 - omega_E)*(omega_1 - omega_E) + m_2*np.abs(omega_2 - omega_E)*(omega_2 - omega_E))

    return[vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, omega_E, ax_E, ay_E, ax_M, ay_M, alpha_1, alpha_2, alpha_E]

def four_body_problem_friction(pos_body_1: list, pos_body_2: list, phi_i: list, phi_E: float, vel_body_1: list, vel_body_2: list):
    solution = solve_ivp(fun=eq_motion_4body_friction, t_span=[t_start, t_max], y0=[*pos_body_1, *pos_body_2, *phi_i, phi_E, *x_E, y_E, x_M, y_M, phi_1, phi_2, phi_E, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, omega_E])
    return [solution.t, x_E, y_E, x_M, y_M, phi_1, phi_2, phi_E, omega_1, omega_2, omega_E]
```

Graphische Darstellung und Vergleich zu ohne Reibung

```
In [ ]: if not fastExecution:
    # 4 body without friction:
    t, x_E, y_E, x_M, y_M, phi_1, phi_2, _, _ = four_body_problem(*iv_stable_orbit_4body(), 0.75*TMondBahn, rtol=1e-12, atol=1e-12)
    x_1, y_1, x_2, y_2 = x_E + Rerde*np.cos(phi_1), y_E + Rerde*np.sin(phi_1), x_E + Rerde*np.cos(phi_2), y_E + Rerde*np.sin(phi_2)
    x_E_1, y_E_1, x_M_1, y_M_1, phi_1_1, phi_2_1, omega_1_1, omega_2_1 = four_body_problem(*iv_stable_orbit_4body(), 100*TMondBahn, rtol=1e-12, atol=1e-12)
    x_1_1, y_1_1, x_2_1, y_2_1 = x_E_1 + Rerde*np.cos(phi_1_1), y_E_1 + Rerde*np.sin(phi_1_1), x_E_1 + Rerde*np.cos(phi_2_1), y_E_1 + Rerde*np.sin(phi_2_1)
    # 4 body with friction short time: 0.3s
    t_f, x_E_f, y_E_f, x_M_f, y_M_f, phi_1_f, phi_2_f, phi_E_f, omega_1_f, omega_2_f, omega_E_f = four_body_problem_friction(*x_1, y_1, x_2, y_2, phi_i, phi_E, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, omega_E, t_max=0.3)
    x_E_f1, y_E_f1, x_M_f1, y_M_f1, phi_1_f1, phi_2_f1, phi_E_f1, omega_1_f1, omega_2_f1, omega_E_f1 = four_body_problem(*x_1, y_1, x_2, y_2, phi_i, phi_E, vx_E, vy_E, vx_M, vy_M, omega_1, omega_2, omega_E, t_max=0.3)
    # 4 body with friction for 100 years: 11min
    t_f1, x_E_f1, y_E_f1, x_M_f1, y_M_f1, phi_1_f1, phi_2_f1, phi_E_f1, omega_1_f1, omega_2_f1, omega_E_f1 = four_body_problem(*x_1_f, y_1_f, x_2_f, y_2_f, phi_i, phi_E, vx_E_f, vy_E_f, vx_M_f, vy_M_f, omega_1_f, omega_2_f, omega_E_f, t_max=11*60*60)
```

```
In [ ]: if not fastExecution:
    # Plot the solution of the four body problem as a phase space diagram
    fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(12, 12)) # Create a figure and a set of subplots
    fig.suptitle("Phasenraumdiagramme des Erde-Mond-Flutberg-Systems: ohne Reibung und mit Reibung")

    def init_phase_space(ax, a, b, title, xlabel, ylabel, color):
        ax.plot(a, b, color=color)
        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_title(title)

    init_phase_space(axes[0,0], t, x_E, "x-Position Erde", "$t$ / s", "$x_1$ / m", "blue")
    init_phase_space(axes[0,1], t, x_M, "x-Position Mond", "$t$ / s", "$x_2$ / m", "red")
    init_phase_space(axes[1,0], t, y_E, "y-Position Erde", "$t$ / s", "$y_1$ / m", "blue")
    init_phase_space(axes[1,1], t, y_M, "y-Position Mond", "$t$ / s", "$y_2$ / m", "red")
    init_phase_space(axes[2,0], x_E, y_E, "Rotation Erde", "$x_E$ / m", "$y_E$ / m", "blue")
    init_phase_space(axes[2,1], x_M, y_M, "Rotation Mond", "$x_M$ / m", "$y_M$ / m", "red")
    init_phase_space(axes[3,0], t, phi_1, "Winkel Flutberg 1", "$t$ / s", "$\phi_1$ / rad", "blue")
    init_phase_space(axes[3,1], t, phi_2, "Winkel Flutberg 2", "$t$ / s", "$\phi_2$ / rad", "red")
    init_phase_space(axes[4,0], x_1, y_1, "Rotation Flutberg 1", "$x_1$ / m", "$x_2$ / m", "blue")
    init_phase_space(axes[4,1], x_2, y_2, "Rotation Flutberg 2", "$y_1$ / m", "$y_2$ / m", "red")

    init_phase_space(axes[0,2], t_f, x_E_f, "x-Position Erde mit Reibung", "$t$ / s", "$x_1$ / m", "blue")
    init_phase_space(axes[0,3], t_f, x_M_f, "x-Position Mond mit Reibung", "$t$ / s", "$x_2$ / m", "red")
    init_phase_space(axes[1,2], t_f, y_E_f, "y-Position Erde mit Reibung", "$t$ / s", "$y_1$ / m", "blue")
    init_phase_space(axes[1,3], t_f, y_M_f, "y-Position Mond mit Reibung", "$t$ / s", "$y_2$ / m", "red")
    init_phase_space(axes[2,2], x_E_f, y_E_f, "Rotation Erde mit Reibung", "$x_E$ / m", "$y_E$ / m", "blue")
    init_phase_space(axes[2,3], x_M_f, y_M_f, "Rotation Mond mit Reibung", "$x_M$ / m", "$y_M$ / m", "red")
    init_phase_space(axes[3,2], t_f, phi_1_f, "Winkel Flutberg 1 mit Reibung", "$t$ / s", "$\phi_1$ / rad", "blue")
    init_phase_space(axes[3,3], t_f, phi_2_f, "Winkel Flutberg 2 mit Reibung", "$t$ / s", "$\phi_2$ / rad", "red")
    init_phase_space(axes[4,2], x_1_f, y_1_f, "Rotation Flutberg 1 mit Reibung", "$x_1$ / m", "$x_2$ / m", "blue")
    init_phase_space(axes[4,3], x_2_f, y_2_f, "Rotation Flutberg 2 mit Reibung", "$y_1$ / m", "$y_2$ / m", "red")

    plt.tight_layout()
    plt.savefig("4body_friction_comp0.png", dpi=300)
    plt.close()

    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
    plt.rc('axes', axisbelow=True)
    plt.suptitle("Vergleich 4 Körper Problem mit Reibung / ohne Reibung")

    def plot_subplot(ax, lines, title, xlabel, ylabel):
        for x, y, label, color in lines:
            ax.plot(x, y, color, linewidth=1, label=label)
        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_title(title)

        ax.grid('minor', 'minor', linestyle='-', linewidth=0.2)
        ax.grid('major', 'major', linestyle='-', linewidth=0.8)
        ax.minorticks_on()
        ax.legend()

    # Distance from the origin for earth and moon
    r_Earth = np.sqrt(x_E_l**2 + y_E_l**2)
    r_Moon = np.sqrt(x_M_l**2 + y_M_l**2)
    r_Earth_f = np.sqrt(x_E_f1**2 + y_E_f1**2)
    r_Moon_f = np.sqrt(x_M_f1**2 + y_M_f1**2)

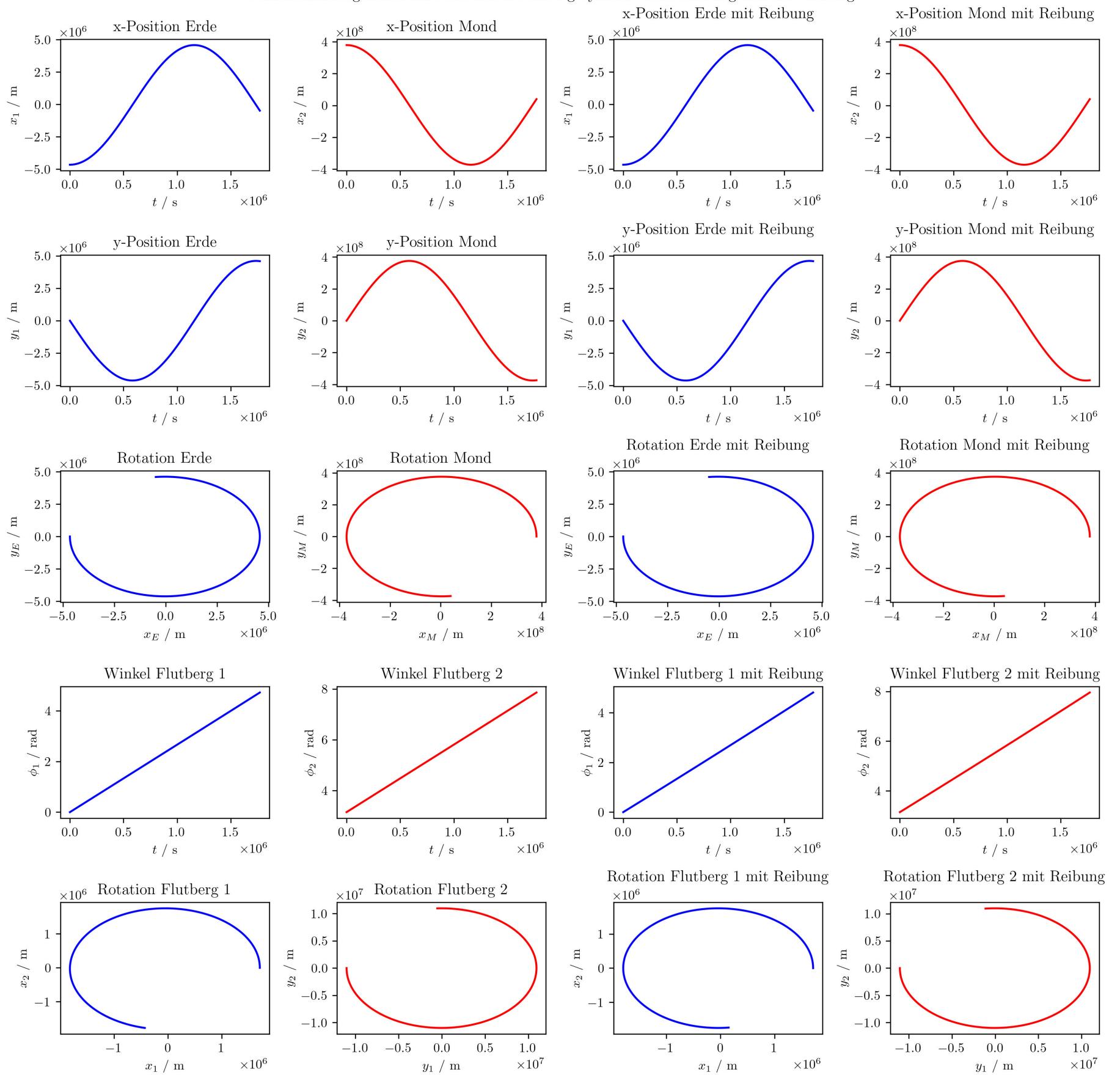
    plot_subplot(axes[0][0], [(x_E_l*10, y_E_l*10, 'Erde (*10)', 'k'), (x_M_l, y_M_l, 'Mond', 'r'), (x_1_l*10, y_1_l*10, 'Flutberg 1', 'r'), (x_2_l*10, y_2_l*10, 'Flutberg 2', 'r')], "Abstand des Mondes zum Ursprung ohne Reibung", "t / s", "r / m")
    plot_subplot(axes[0][1], [(t_l, r_Moon, 'Mond', 'r')], "Abstand der Erde zum Ursprung ohne Reibung", "t / s", "r / m")
    plot_subplot(axes[0][2], [(t_l, r_Earth, 'Erde', 'k')], "Abstand des Mondes zum Ursprung mit Reibung", "t / s", "r / m")
    plot_subplot(axes[1][0], [(x_E_f1*10, y_E_f1*10, 'Erde (*10)', 'k'), (x_M_f1, y_M_f1, 'Mond', 'r'), (x_1_f1*10, y_1_f1*10, 'Flutberg 1', 'r'), (x_2_f1*10, y_2_f1*10, 'Flutberg 2', 'r')], "Abstand des Mondes zum Ursprung ohne Reibung", "t / s", "r / m")
    plot_subplot(axes[1][1], [(t_f1, r_Moon_f, 'Mond', 'r')], "Abstand der Erde zum Ursprung mit Reibung", "t / s", "r / m")
    plot_subplot(axes[1][2], [(t_f1, r_Earth_f, 'Erde', 'k')], "Abstand der Erde zum Ursprung mit Reibung", "t / s", "r / m")
    plt.tight_layout()
    plt.savefig("4body_friction_comp1.png", dpi=300)
    plt.close()

    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 5))
    plt.rc('axes', axisbelow=True)
    plt.suptitle("Vergleich 4 Körper Problem mit Reibung / ohne Reibung")

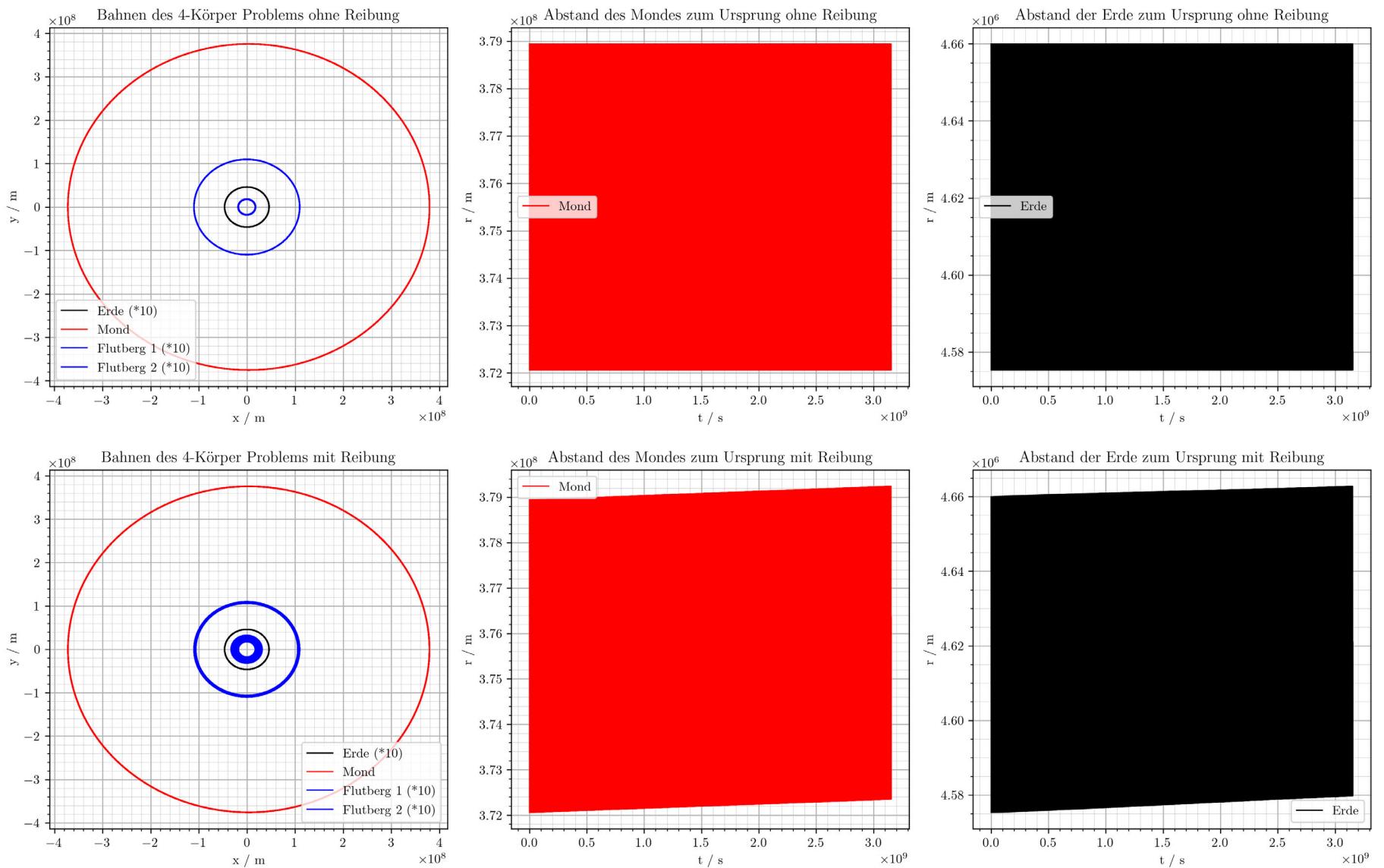
    plot_subplot(axes[0][0], [(t_l, omega_1_l, '$\omega_1$ Flutberg 1', 'k'), (t_l, omega_2_l, '$\omega_2$ Flutberg 2', 'r'), (t_f1, omega_1_f1, '$\omega_1$ Flutberg 1', 'k'), (t_f1, omega_2_f1, '$\omega_2$ Flutberg 2', 'r')], "Winkelgeschwindigkeit Flutberg 1", "t / s", "r / m")
    plot_subplot(axes[0][1], [(t_l, [2*pi/TERdRotation]*np.shape(t_l)[0], '$\omega_E$ Winkelgeschwindigkeit Erde', 'k')], "Winkelgeschwindigkeit Erde", "t / s", "r / m")
    plot_subplot(axes[1][0], [(t_f1, omega_1_f1, '$\omega_1$ Flutberg 1', 'k'), (t_f1, omega_2_f1, '$\omega_2$ Flutberg 2', 'r')], "Winkelgeschwindigkeit Flutberg 2", "t / s", "r / m")
    plot_subplot(axes[1][1], [(t_f1, omega_E_f1, '$\omega_E$ Winkelgeschwindigkeit Erde', 'k'), (t_f1, [2*pi/TERdRotation]*np.shape(t_f1)[0], '$\omega_E$ Winkelgeschwindigkeit Mond', 'r')], "Winkelgeschwindigkeit Mond", "t / s", "r / m")
    plt.tight_layout()
    plt.savefig("4body_friction_comp2.png", dpi=300)
    plt.close()

    # Display image set the size to be 500x500 pixels
    display(Image(filename='4body_friction_comp0.png', width=1000, height=2000))
    display(Image(filename='4body_friction_comp1.png', width=1000, height=500))
    display(Image(filename='4body_friction_comp2.png', width=1000, height=500))
```

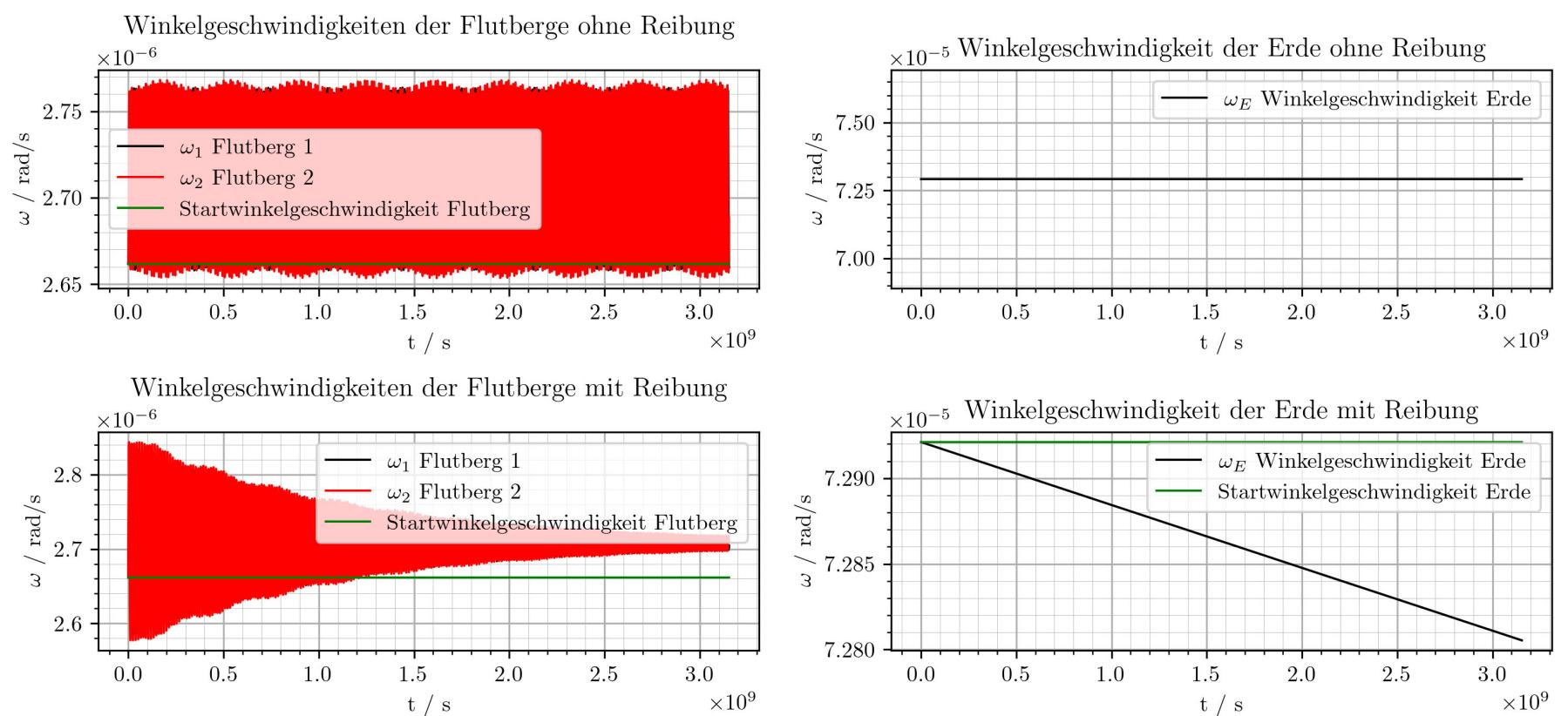
Phasenraumdiagramme des Erde-Mond-Flutberg-Systems: ohne Reibung und mit Reibung



Vergleich 4 Körper Problem mit Reibung / ohne Reibung



Vergleich 4 Körper Problem mit Reibung / ohne Reibung



Im Fall ohne Reibung ist  $\omega_E$  natürlich konstant, und im Fall mit Reibung nimmt  $\omega_E$  mit der Zeit ab, das heißt, dass die Tage auf der Erde länger werden. Hier sieht die Abnahme linear aus, es handelt sich aber um einen exponentiellen Abfall, welcher aufgrund der betrachteten Zeitskala aber linear erscheint.

Zudem ist zu sehen, dass Erde und Mond sich mit Reibung von ihrem gemeinsamen Schwerpunkt weg bewegen, was ohne Reibung nicht passiert. Das ist eine Folge davon, dass die Erdrotation schneller als die Umlaufbahn des Mondes ist. Die Flutberge werden aufgrund der Reibung vor die Verbindungslinie zwischen Erde und Mond gedrückt, was die Wirkung der Flutberge auf den Mond ändert. Der Mond erfährt eine Radialbeschleunigung, wodurch er auf eine weiter entfernte Bahn gelangt.

Vergleicht man die Winkelgeschwindigkeiten der Flutberge, so erkennt man, dass diese in beiden Fällen mit einer schwach modulierten Oszillation starten. Mit der Zeit schwächt sich dies mit Reibung ab, ohne Reibung bleibt der Effekt zeitlich gleich. Die Bewegung ist Folge davon, dass die Umlaufbahn nicht perfekt ein Kreis ist (Wie man auch in den Plots vom Abstand zum Ursprung erkennen kann), weshalb der Mond die Flutberge zu unterschiedlichen Zeitpunkten unterschiedlich stark beschleunigt. Mit Reibung wird diese Bewegung abgebremst (Energiedissipation), weshalb die Amplitude der Oszillation abnimmt. Die Amplitude ist zu Beginn jedoch größer, weil die Flutberge durch die Reibung in Richtung der Erdrotation beschleunigt werden.

Wir stellen noch dar, wie sich der Mond von der Erde entfernt:

```
In [ ]: #Show the increase of the moons orbit
if not fastExecution:
    #Moon as seen from Earth:
    x = x_M_f1 - x_E_f1
    y = y_M_f1 - y_E_f1

    fig = plt.figure(figsize=(12,6))
    fig.suptitle('Vergrößerung der Mondbahn über 100 Jahre')
    ax = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

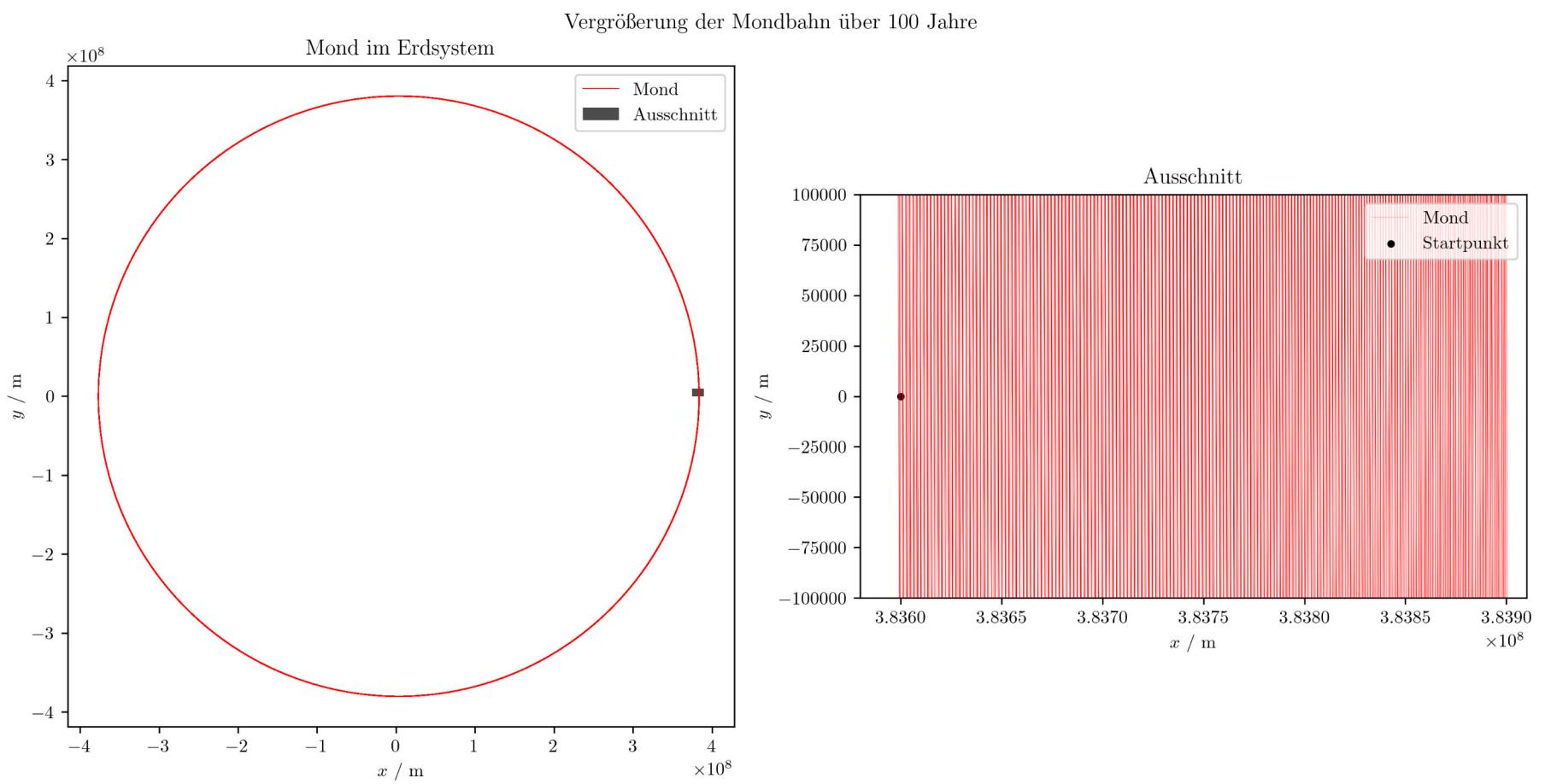
    ax.set_title('Mond im Erdsystem')
    ax.set_xlabel("$x$ / m")
    ax.set_ylabel("$y$ / m")
    ax.set_aspect('equal')
    ax.plot(x, y, color='r', linewidth=0.5, label='Mond')

    rect = plt.Rectangle((3.75e8, -1e5), 1.5e7, 1e7, facecolor="black", alpha=0.7, label='Ausschnitt')
    ax.add_patch(rect)
    ax.legend()

    ax2.set_title('Ausschnitt')
    ax2.set_xlabel("$x$ / m")
    ax2.set_ylabel("$y$ / m")
    ax2.set_aspect('equal')
    ax2.plot(x, y, 'r', linewidth=0.1, label='Mond')
    ax2.scatter(x[0], y[0], c='k', s=10, label='Startpunkt')
    ax2.set_xlim([3.8358e8, 3.8391e8])
    ax2.set_ylim([-1e5, 1e5])
    ax2.legend()

    plt.tight_layout()
    plt.savefig("4body_friction_Moon.png", dpi=300)
    plt.close()

display(Image(filename='4body_friction_Moon.png', width=1000, height=2000))
```



### Berechnung der Tageslängenänderung

Für die erstmalige Berechnung der Tageslängenänderung verwenden wir die Reibungskonstante  $k = 2 \cdot 10^{-12} \frac{1}{\text{m}}$  und Massen für Flutberge von  $m_1 = m_2 = \frac{1}{2}m_{\text{Ozean}} = \frac{1}{2} \cdot 0.0014 \cdot 10^{24}$

Die Verlangsamung der Erdrotation kann man auch in Realität beobachten. Deshalb vergleichen wir unsere Ergebnisse mit dem Literaturwert für die Tageslängenänderung über 100 Jahre  $\tau = 0.0021 \frac{s}{100a}$ .

Wie im Graphen für die Winkelgeschwindigkeiten der Flutberge zu erkennen ist braucht das System eine gewisse Einschwingzeit. Wir wählen konservativ 200 Jahre als Einschwingzeit (mit Hilfe des Graphen bestimmt). Im folgenden berechnen wir die Tageslängenänderung für 500 Jahre mit 100 Jahr Zeitschritten.

Die Simulation liefert uns die Winkelgeschwindigkeit der Erde  $\omega_E$  zu gewissen Zeitpunkten. Diese kann mit der Formel  $\omega_E = \frac{2\pi}{T} \Rightarrow T = 2\pi/\omega$  in die Tageslänge  $T$  umgerechnet werden. Die Differenz der Tageslängen ergibt die Tageslängenänderung  $\tau$ .

```
In [ ]: if not fastExecution:
    year_in_seconds = 365*24*3600
    t_f5ha, _, _, _, _, _, _, _, omega_E_f5ha = four_body_problem_friction(*iv_stable_orbit_4body_friction(), k=2e-12, t
    angular_velocities = [omega_E_f5ha[i] for i in np.argmin(np.abs(t_f5ha - j*year_in_seconds)) for j in [0, 100, 200, 300,
    tau_calc = [2*pi/angular_velocities[i+1] - 2*pi/angular_velocities[i] for i in range(5)]
    for i in range (5):
        print(f'\t für ({i}ha-{i+1}ha): {tau_calc[i]:.3f} s/ha')
else:
    print("τ für (0ha-1ha): 137.090 s/ha\nτ für (1ha-2ha): 137.086 s/ha\nτ für (2ha-3ha): 137.082 s/ha\nτ für (3ha-4ha): 137.078 s/ha
    τ für (4ha-5ha): 137.073 s/ha
```

Das berechnete Ergebnis von ca.  $137 \frac{s}{100a}$  weicht stark vom Literaturwert  $\tau$  ab.

Es ist deutlich zu erkennen, dass die Tageslängenänderung mit der Zeit abnimmt, da die Reibungskraft mit der Zeit abnimmt. Auf unseren Zeitskalen ist eine Näherung als linear aber noch vertretbar. Die Einschwingzeit scheint für die Berechnung nicht relevant zu sein, da in den ersten 100 Jahren kein deutlich größerer Unterschied des Ergebnisses zu den folgenden Zeitschritten zu erkennen ist.

Da unser berechneter Wert so stark vom Literaturwert abweicht, möchten wir die Parameter unserer Simulation anpassen um den Literaturwert zu erreichen. Dies verwenden wir für die folgenden Fits, um die Laufzeit zu verkürzen in dem wir die Berechnungen nur für die ersten 100 Jahre durchführen.

## Shooting Methode und Variation der Flutbergmasse und Reibungskonstante

Die Reibung der Flutberge auf der Erdoberfläche sorgt für eine Verlangsamung der Erdrotation und einem größeren Orbit für den Mond. Wir haben für diese Berechnung zwei freie Parameter, die wir im folgenden anpassen wollen:

- Die Masse der Flutberge  $m_1$  und  $m_2$  (die Masse des Ozeans)
- Die Reibungskonstante  $k$

### Shooting Methode

Mit Hilfe der Shooting Methode kann man Randwertprobleme numerisch auf Anfangswertprobleme zurückführen. Es wird zu Anfang ein Intervall geraten in dem die mögliche Lösung liegt. Für die Mitte des Intervalls wird das Problem numerisch gelöst und mit dem Randwert verglichen. Es wird nun mit Hilfe des Bisektionverfahrens das Intervall solange halbiert bis die Lösung ausreichend genau gefunden wurde.

### Variation der Masse der Flutberge

Wir modifizieren die Massen der Flutberge so, dass sie für die aktuellen Zunahme der Tageslänge pro 100 Jahre mit dem oben angegebenen Literaturwert  $\tau$  übereinstimmt (hier bleibt  $k$  unverändert).

In der Differentialgleichung für die Erdrotationsbeschleunigung sehen wir, dass diese linear mit der Masse der Flutberge zusammenhängt. Die berechnete Erdrotationsbeschleunigung ist zu schnell, deshalb müssen wir die Masse der Flutberge verkleinern. Wir wählen deshalb für die Shooting Methode als Suchintervall Null bis zur aktuellen Ozeanmasse. Es wäre möglich die Massen der Flutberge einzeln zu variieren, wir haben uns aber entschieden sie für die folgenden Berechnungen gleich zu setzen.

### Variation der Reibungskonstante

Die verwendete Ozeanmasse entspricht bereits den reellen Begebenheiten, deshalb ist es nicht sinnvoll diese zu fitten. Alternativ können wir die Reibungskonstante verändern. Auch sie ist aus den gleichen Gründen zu groß und wir wählen als Suchintervall Null bis zum aktuellen Wert. Die Masse der Flutberge lassen wir hier unverändert.

Zum Ausführen der Shooting-Methode müssen wir noch eine Toleranz angeben. Der Literaturwert  $\tau = 0.0021$  ist mit einer Genauigkeit von  $10^{-5}$  angegeben, deshalb wählen wir diese als Toleranz.

```
In [ ]: def rotation_perdiode_change_mOcean(mOcean: float):
    '''Calculates the change in rotation_period after 100 years, given a value for mOcean'''
    initial_values = iv_stable_orbit_4body_friction()
    # Change the mass of the two tides to each be half the value of mOcean for the simulation
    initial_values[-1][-1], initial_values[-1][-2] = mOcean/2, mOcean/2
    _, _, _, _, _, _, _, _, omega_E = four_body_problem_friction(*initial_values, k=2*10**(-12), t_max=100*365*24*3600)
    return (2*pi/omega_E[-1]) - (2*pi/omega_E[0]) # Calculate the change in rotation period as described above

def rotation_period_change_k(k: float):
    '''Calculates the change in rotation_period after 100 years, given a value for k'''
    _, _, _, _, _, _, _, _, omega_E = four_body_problem_friction(*iv_stable_orbit_4body_friction(), k=k, t_max=100*365*2
    return (2*pi/omega_E[-1]) - (2*pi/omega_E[0]) # Calculate the change in rotation period as described above

def shooting_method(func, target: float, val_min: float, val_max: float, tol: float):
    '''Find the value of x for which func(x) is in the interval [target-tol, target+tol]
    Search in the interval [val_min, val_max]'''

    # Check if the solution is in the passed interval
    if func(val_min) > target or func(val_max) < target:
        raise ValueError('The target is not in the passed interval')
    print("Target in interval")

    # Iterate until the solution is within the tolerance
    while True:
        x = (val_min + val_max)/2 # Calculate the midpoint of the interval
        func_val = func(x) # Calculate the value of the function at the midpoint
        if np.abs(func_val - target) < tol: # Solution found if the absolute distance to the solution is smaller than the tolerance
            return x
        elif func_val < target: # The lower bound can be raised
            val_min = x
        else: # The upper bound can be lowered
            val_max = x

    # Fit the value of mOceans so that the change in rotation period is tau
if not fastExecution:
    mOcean_fit = shooting_method(func=rotation_perdiode_change_mOcean, target=tau, val_min=0, val_max=mOcean, tol=0.00001)
else:
    mOcean_fit = 2.13623046875e+16

# Fit the value of k so that the change in rotation period is tau
if not fastExecution:
    k_fit = shooting_method(func=rotation_period_change_k, target=tau, val_min=0, val_max=2*10**(-12), tol=0.00001)
else:
    k_fit = 3.06640625e-17

print(f'mOcean_fit: {mOcean_fit:.3g} kg')
print(f'k_fit: {k_fit:.3g} 1/m')
```

mOcean\_fit: 2.14e+16 kg  
k\_fit: 3.07e-17 1/m

### Ergebnisse Fit durch Shooting Methode

Wir erhalten wie erwartet durch den Fit für Ozeanmasse und Reibungskonstante kleinere Werte als unsere Startwerte.

Der gefittete Wert für die Ozeanmasse liegt aber mit  $2.14 \cdot 10^{16}$  kg deutlich unter dem vorher verwendeten Literaturwert von  $1.4 \cdot 10^{21}$  kg  
Quelle: [Ocean#Weigth](#).

Die Abweichungen lassen sich durch mehrere Näherungen in der Simulation erklären:

- Korrektheit der Reibungskonstante
- Die Sonne wird nicht berücksichtigt. Sie trägt einen kleinen Teil zu den Gezeiten bei.
- Die Erde verformt sich durch die Gezeitenkräfte
- Landmassen und ihre Effekte auf die Gezeiten werden nicht berücksichtigt

Nach den obigen Betrachtungen scheint es also sinnvoller statt der Ozeanmasse die Reibungskonstante zu fitten. Diese verhält sich auch viel mehr wie ein freier Parameter, da sie viele in der Realität auftretende Effekte zusammenfasst. (z.B. Landmassen, verschiedene Meerestiefen, Strömungen)

### Vergleich der gefundenen Reibungskonstante mit der gegebenen

```
In [ ]: if not fastExecution:
    # 4 body with friction for 100 years: 11min
    t_kfit, x_E_kfit, y_E_kfit, x_M_kfit, y_M_kfit, phi_1_kfit, phi_2_kfit, phi_E_kfit, omega_1_kfit, omega_2_kfit, omega_E_kf
    x_1_kfit, y_1_kfit, x_2_kfit, y_2_kfit = x_E_kfit + RErde*np.cos(phi_1_kfit), y_E_kfit + RErde*np.sin(phi_1_kfit), x_E_kfi
```

```
In [ ]: if not fastExecution:
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
    plt.rc('axes', axisbelow=True)
    plt.suptitle("Vergleich 4 Körper Problem  $k=2 \cdot 10^{-12}$  /  $k=3 \cdot 10^{-17}$ ")

    def plot_subplot(ax, lines, title, xlabel, ylabel):
        for x, y, label, color in lines:
            ax.plot(x, y, color, linewidth=1, label=label)
        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_title(title)

        ax.grid('minor', 'minor', linestyle='-', linewidth=0.2)
        ax.grid('major', 'major', linestyle='-', linewidth=0.8)
        ax.minorticks_on()
        ax.legend()

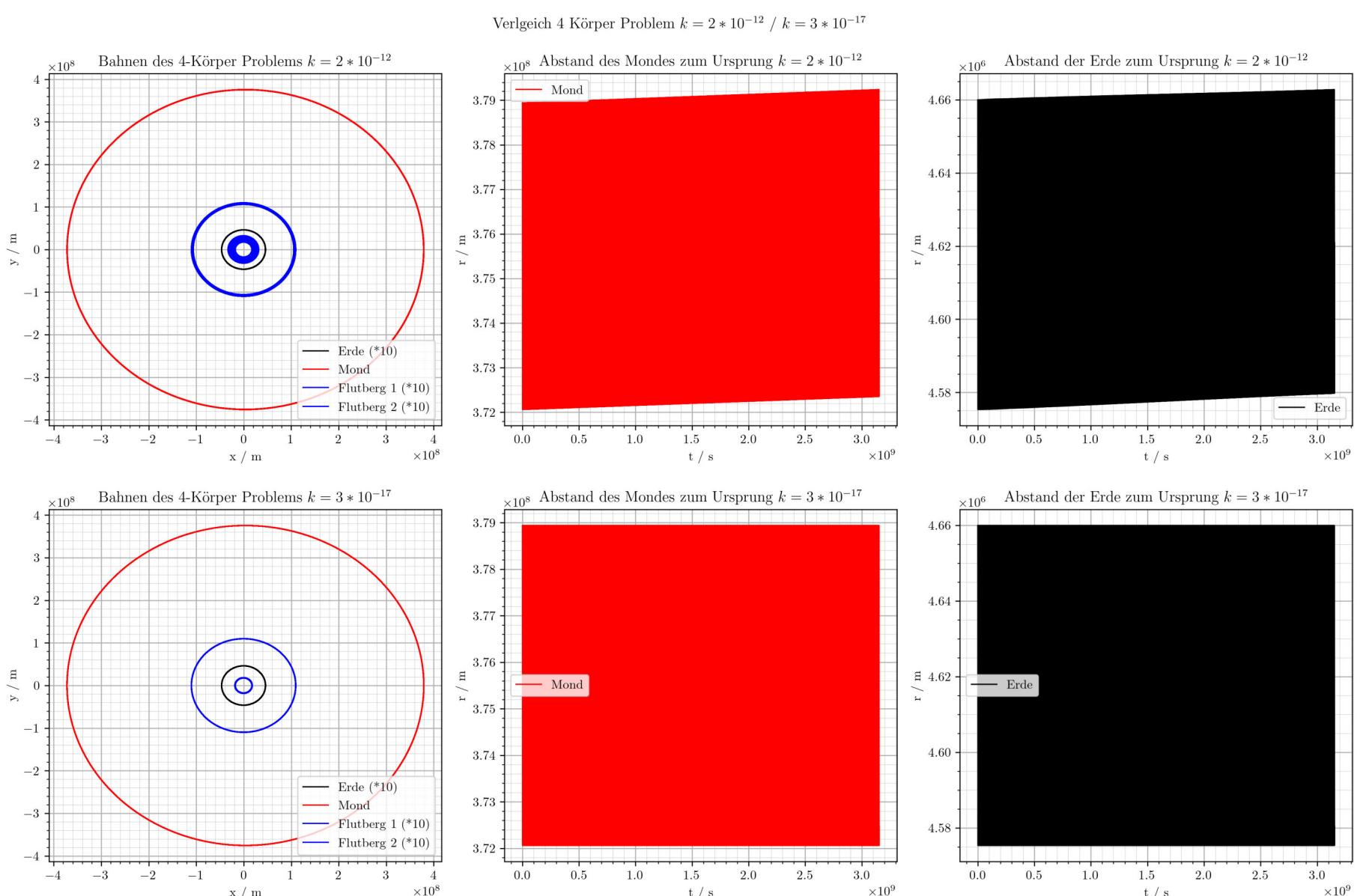
    # Distance from the origin for earth and moon
    r_Earth_fl = np.sqrt(x_E_fl**2 + y_E_fl**2)
    r_Moon_fl = np.sqrt(x_M_fl**2 + y_M_fl**2)
    r_Earth_kfit = np.sqrt(x_E_kfit**2 + y_E_kfit**2)
    r_Moon_kfit = np.sqrt(x_M_kfit**2 + y_M_kfit**2)

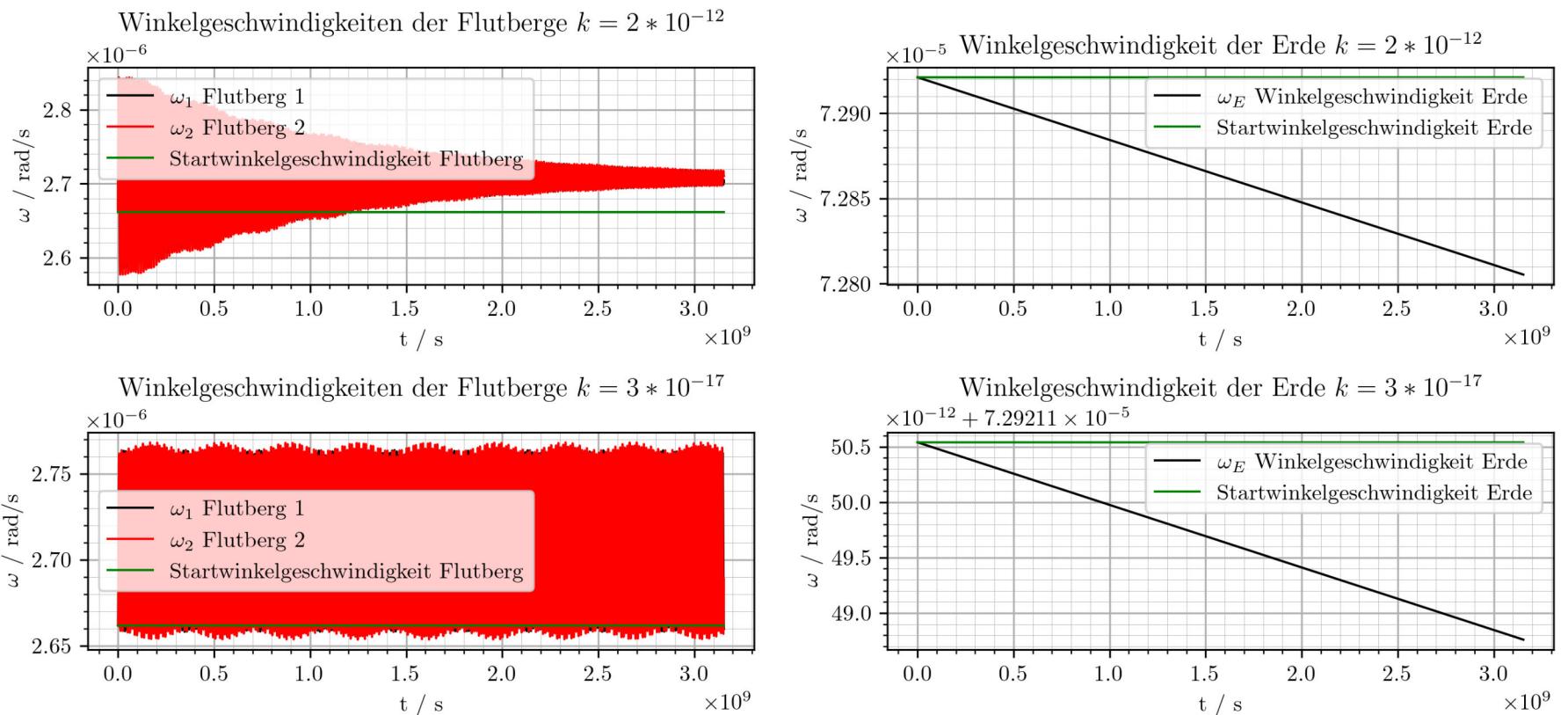
    plot_subplot(axes[0][0], [(x_E_fl*10, y_E_fl*10, 'Erde (*10)', 'k'), (x_M_fl, y_M_fl, 'Mond', 'r'), (x_1_fl*10, y_1_fl*10, 'Flutberg 1 (*10)', 'b'), (x_2_fl*10, y_2_fl*10, 'Flutberg 2 (*10)', 'b')], "Abstand des Mondes zum Ursprung  $k=2 \cdot 10^{-12}$ ", "t / s", "r / m")
    plot_subplot(axes[0][1], [(t_fl, r_Moon_fl, 'Mond', 'r')], "Abstand der Erde zum Ursprung  $k=2 \cdot 10^{-12}$ ", "t / s", "r / m")
    plot_subplot(axes[0][2], [(t_fl, r_Earth_fl, 'Erde', 'k')], "Abstand der Erde zum Ursprung  $k=2 \cdot 10^{-12}$ ", "t / s", "r / m")
    plot_subplot(axes[1][0], [(x_E_kfit*10, y_E_kfit*10, 'Erde (*10)', 'k'), (x_M_kfit, y_M_kfit, 'Mond', 'r'), (x_1_kfit*10, y_1_kfit*10, 'Flutberg 1 (*10)', 'b'), (x_2_kfit*10, y_2_kfit*10, 'Flutberg 2 (*10)', 'b')], "Abstand des Mondes zum Ursprung  $k=3 \cdot 10^{-17}$ ", "t / s", "r / m")
    plot_subplot(axes[1][1], [(t_kfit, r_Moon_kfit, 'Mond', 'r')], "Abstand der Erde zum Ursprung  $k=3 \cdot 10^{-17}$ ", "t / s", "r / m")
    plot_subplot(axes[1][2], [(t_kfit, r_Earth_kfit, 'Erde', 'k')], "Abstand der Erde zum Ursprung  $k=3 \cdot 10^{-17}$ ", "t / s", "r / m")
    plt.tight_layout()
    plt.savefig("4body_friction_comp1_k_fit.png", dpi=300)
    plt.close()

    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 5))
    plt.rc('axes', axisbelow=True)
    plt.suptitle("Vergleich 4 Körper Problem  $k=2 \cdot 10^{-12}$  /  $k=3 \cdot 10^{-17}$ ")

    plot_subplot(axes[0][0], [(t_fl, omega_1_fl, '$\omega_1$ Flutberg 1', 'k'), (t_fl, omega_2_fl, '$\omega_2$ Flutberg 2', 'r')], "Winkelgeschwindigkeit Flutberg 1 und 2", "t / s", "r / m")
    plot_subplot(axes[0][1], [(t_fl, omega_E_fl, '$\omega_E$ Winkelgeschwindigkeit Erde', 'k')], "Winkelgeschwindigkeit Erde", "t / s", "r / m")
    plot_subplot(axes[1][0], [(t_kfit, omega_1_kfit, '$\omega_1$ Flutberg 1', 'k'), (t_kfit, omega_2_kfit, '$\omega_2$ Flutberg 2', 'r')], "Winkelgeschwindigkeit Flutberg 1 und 2", "t / s", "r / m")
    plot_subplot(axes[1][1], [(t_kfit, omega_E_kfit, '$\omega_E$ Winkelgeschwindigkeit Erde', 'k')], "Winkelgeschwindigkeit Erde", "t / s", "r / m")
    plt.tight_layout()
    plt.savefig("4body_friction_comp2_k_fit.png", dpi=300)
    plt.close()

# Display image set the size to be 500x500 pixels
display(Image(filename='4body_friction_comp1_k_fit.png', width=1000, height=500))
display(Image(filename='4body_friction_comp2_k_fit.png', width=1000, height=500))
```





Da die gefüllte Reibungskonstante um fünf Größenordnungen kleiner ist, sind die zuvor diskutierten Reibungseffekte sehr viel kleiner. Die Tageslängenänderung lässt sich jedoch trotzdem erkennen. Die anderen Effekte (Entfernung des Mondes von der Erde, Einschwingen), sind jedoch so klein, dass sie in den Plots nicht zu sehen sind.

### Vielteilchenproblem mit Randbedingungen, intrinsischer Rotation und Reibung

Nun betrachten wir das Erde-Mond System mit  $N$  Massenpunkten auf der Erdoberfläche, welche die Wassermassen darstellen. Ziel ist es das Ausbilden der Flutberge zu simulieren. Für die Mond- und Erdkoordinaten verwenden wir die Differentialgleichung des Zweikörperproblems, als Näherung um den Rechenaufwand zu reduzieren. Die Näherung wird später genauer diskutiert. Die Kraft auf die Masspunkte ist gleich zur Kraft auf die zwei Flutberge mit Reibung, welche oben erläutert wurde. Die Differentialgleichung für  $\omega_E$  wird mit  $N$  Massenpunkten, analog zum Fall mit zwei Flutbergen, zu:

$$\dot{\omega}_E = \frac{5kR_E}{2m_E} \sum_{i=1}^N m_i |\omega_i - \omega_E| (\omega_i - \omega_E)$$

In [ ]: `N = 25 # Amount of tides`

Wir verwenden wenn möglich die bereits implementierten Gleichungen. Die Winkel und Winkelbeschleunigungen der Flutberge werden als Liste übergeben, um  $N$  leicht ändern zu können.

```
In [ ]: def eq_motion_Nbody(t, state, mass, k):
    '''This function calculates the derivatives of the state vector the 2 body problem with N Tides to be passed to solve_ivp.
    state: state vector has the following structure: [x_E, y_E, x_M, y_M, phi_E, vx_E, vy_E, vx_M, vy_M, omega_E, phi_1, phi_2
    mass: list of masses e.g. [m_E, m_M, m1, m2, ..., mN]'''

    # Unpack the state vector and masses:
    x_E, y_E, x_M, y_M, phi_E, vx_E, vy_E, vx_M, vy_M, omega_E, phi_tide, omega_tide = *state[0:10], [state[10+i] for i in range(N)]
    m_E, m_M, m_tide = *mass[0:2], [mass[2+i] for i in range(N)]

    # Earth and moon acceleration from the 2 body problem:
    _, _, _, _, ax_E, ay_E, ax_M, ay_M = eq_motion_2body(t, [x_E, y_E, x_M, y_M, vx_E, vy_E, vx_M, vy_M], [m_E, m_M])

    # Angular momentum of the earth:
    alpha_E = (5*k*RErde)/(2*m_E) * np.sum([m_tide[i]*abs(omega_tide[i] - omega_E)*(omega_tide[i] - omega_E) for i in range(N)])

    # The tides' cartesian coordinates:
    x_tide, y_tide = [x_E + RErde*np.cos(phi_tide[i]) for i in range(N)], [y_E + RErde*np.sin(phi_tide[i]) for i in range(N)]

    # The earth's angular acceleration:
    alpha_tide = [tides_acceleration_friction(x_M, y_M, x_tide[i], y_tide[i], phi_tide[i], ax_E, ay_E, omega_tide[i], omega_E,
                                                friction_coefficient=0.001) for i in range(N)]

    return [vx_E, vy_E, vx_M, vy_M, omega_E, ax_E, ay_E, ax_M, ay_M, alpha_E] + omega_tide + alpha_tide
```

Da für die Erd- und Mondkoordinaten die Zweikörper-Differentialgleichung verwendet wird wählen wir für diese dieselben Anfangsbedingungen. Die Massenpunkte starten gleichmäßig verteilt auf der Erdoberfläche mit der mittleren Winkelgeschwindigkeit der Flutberge mit Reibung.

```
In [ ]: if not fastExecution:
    _, _, _, _, _, _, _, omega_Niv, _, _ = four_body_problem_friction(*iv_stable_orbit_4body_friction(), k=k_fit, t_max=100)
    omega_i0 = np.mean(omega_Niv)
else:
    omega_i0 = 2.7107783823217206e-06
```

```
In [ ]: def iv_stable_orbit_Nbody():
    '''Initial conditions for the Earth-Moon-NTides system in a stable orbit'''
    # Each of the tides gets an nth of the oceans mass
    mass = [mErde, mMond] + [mOzean/N for i in range(N)]
    # Because the DEQ for the 2 body problem is used for the earth and moon movement, we can reuse the initial conditions (Tides)
    x0_Eerde, x0_Mond, v0_Eerde, v0_Mond, _, _ = iv_stable_orbit_2body()
    _, _, _, phi0_E, _, _, omega0_E, _ = iv_stable_orbit_4body_friction()

    phi0 = [(2*pi*i)/N for i in range(N)] # The tides all start spaced evenly around the earth
    #omega0 = [2*pi/TMondBahn]*N # The tides all start with the same angular velocity as the moon
    omega0 = [omega_i0]*N

    return [x0_Eerde, x0_Mond, phi0_E, v0_Eerde, v0_Mond, omega0_E, phi0, omega0, mass]

def N_body_problem(pos_body_1: list, pos_body_2: list, phi_E: float, vel_body_1: list, vel_body_2: list, omega_E: float, phi_tide: float):
    solution = solve_ivp(fun=eq_motion_Nbody, t_span=[t_start, t_max], y0=[*pos_body_1, *pos_body_2, phi_E, *vel_body_1, *vel_body_2])
    x_E, y_E, x_M, y_M, phi_E, vx_E, vy_E, vx_M, vy_M, omega_E = solution.y[0:10]
    phi_tide = [solution.y[10+i] for i in range(N)]
    omega_tide = [solution.y[10+N+i] for i in range(N)]
    return [solution.t, x_E, y_E, x_M, y_M, phi_tide, omega_E, omega_tide]
```

```
In [ ]: t, x_E, y_E, x_M, y_M, phi_tide, omega_E, omega_tide = N_body_problem(*iv_stable_orbit_Nbody(), k=k_fit, t_max=20*TMondBahn)
```

```
In [ ]: # Cartesian coordinates of the tides:
x_tide, y_tide = [x_E + RErde*np.cos(phi_tide[i]) for i in range(N)], [y_E + RErde*np.sin(phi_tide[i]) for i in range(N)]

# Plot the movement of the earth, moon and tides
plt.rc('axes', axisbelow=True)
plt.figure(figsize=(5, 5))
plt.title("Bahnen des N-Flutberg-System")
plt.grid('minor', 'minor', linestyle='-', linewidth=0.2)
plt.grid('major', 'major', linestyle='-', linewidth=0.8)
plt.minorticks_on()
plt.xlabel("x")
plt.ylabel("y")
plt.plot(x_M/10, y_M/10, 'r', linewidth=1, label='Mond (/10)')
for i in range(N):
    plt.plot(x_tide[i], y_tide[i], 'b', linewidth=0.5, alpha=0.2)
plt.plot(x_E, y_E, 'k', linewidth=1, label='Erde')
plt.legend()
plt.show()
plt.close()

# Plot the angles of the tides over time, subtracting the angle of the moon
plt.rc('axes', axisbelow=True)
plt.figure(figsize=(5, 5))
plt.title("Winkel der Flutberge über Zeit, abzüglich  $t \cdot \omega_0$ ")
plt.grid('minor', 'minor', linestyle='-', linewidth=0.2)
plt.grid('major', 'major', linestyle='-', linewidth=0.8)
plt.minorticks_on()
plt.xlabel("$t / s$")
plt.ylabel("$\phi - t \cdot \omega_0 / rad$")

for i in range(N):
    plt.plot(t, phi_tide[i] - t*2*pi/TMondBahn, 'k', linewidth=0.7, alpha=0.7)

plt.show()
plt.close()

# Plot the angular velocity of the earth over time
plt.rc('axes', axisbelow=True)
plt.figure(figsize=(5, 5))
plt.title("Winkelgeschwindigkeit der Erde über Zeit")
plt.grid('minor', 'minor', linestyle='-', linewidth=0.2)
plt.grid('major', 'major', linestyle='-', linewidth=0.8)
plt.minorticks_on()
plt.xlabel("$t / s$")
plt.ylabel("$\omega_E / rad$")
plt.plot(t, omega_E, 'k', linewidth=1, label='$\omega_E$')

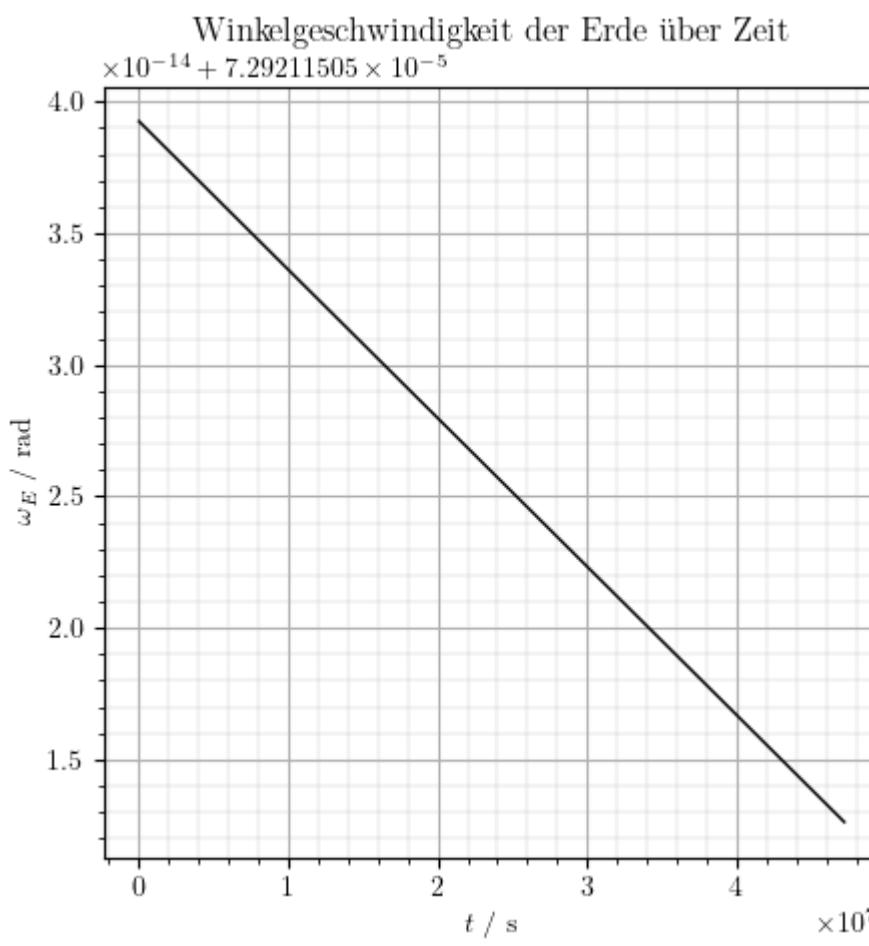
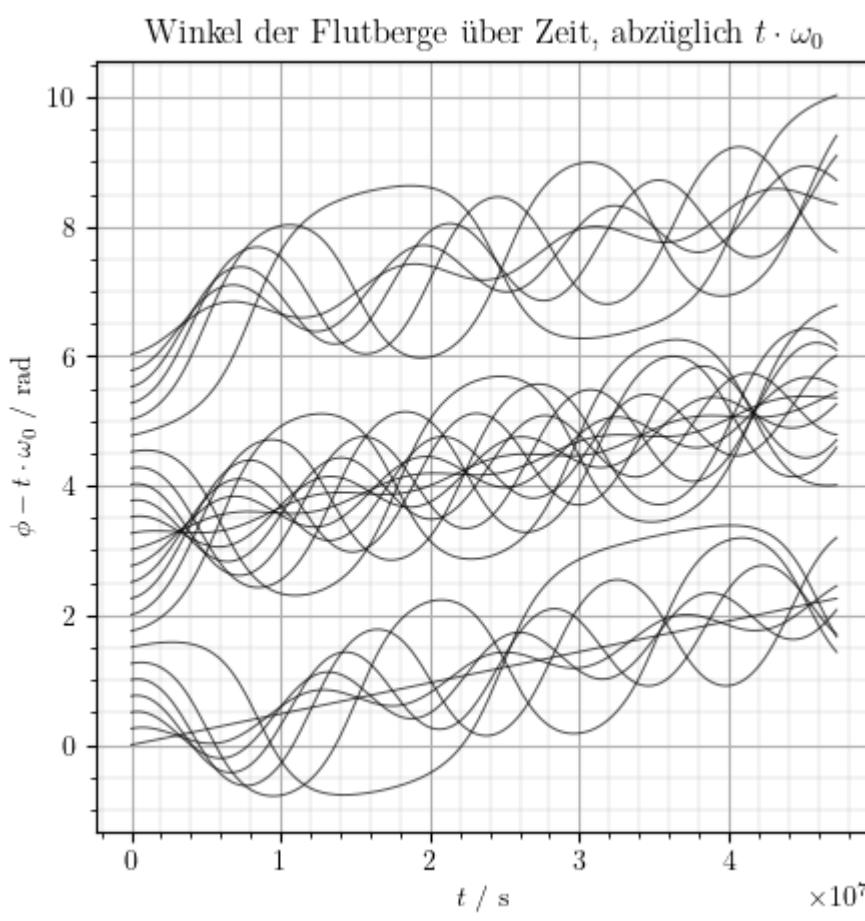
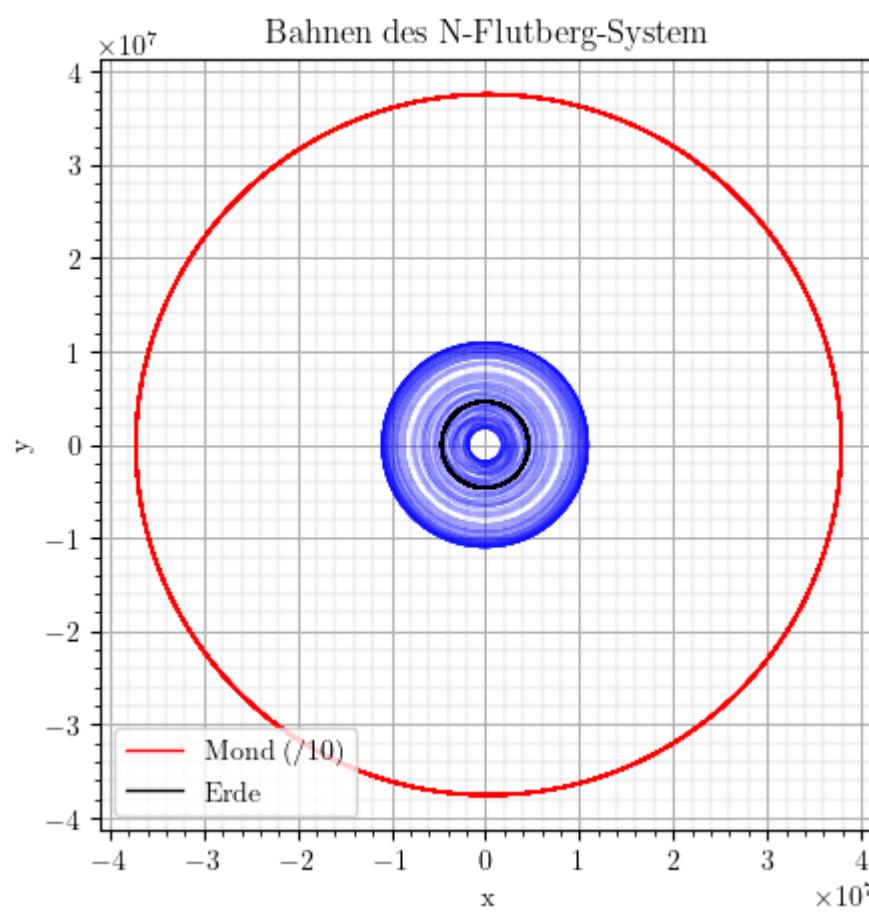
plt.show()
plt.close()

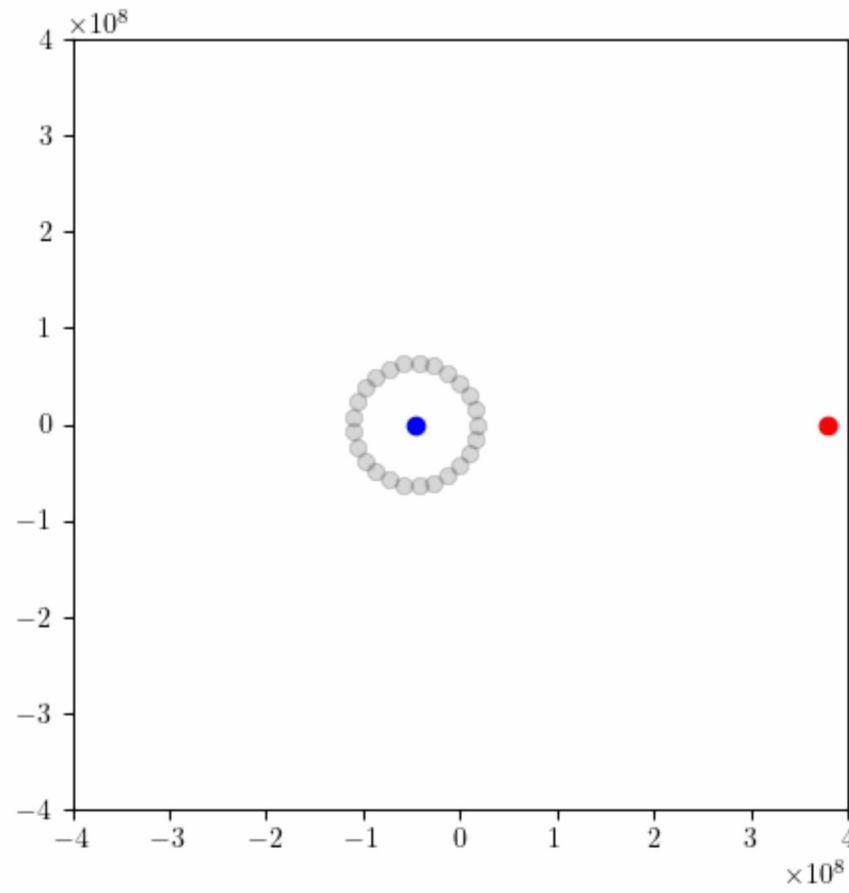
# Animation of the N body problem
if not fastExecution:
    fig = plt.figure(figsize=(5, 5))
    ax = plt.axes(xlim=(-4e8, 4e8), ylim=(-4e8, 4e8))
    ax.set_title("Animation des N-Flutberg-Systems")
    earth_line, = ax.plot([], [], marker='o', lw=0.1, color='blue', label='Erde (*10)')
    moon_line, = ax.plot([], [], marker='o', lw=0.1, color='red', label='Mond')
    tide_lines = [ax.plot([], [], marker='o', lw=0.1, color='black', label='Flutberg {i} (*10)')[0] for i in range(N)]

    def animate(i):
        earth_line.set_data([x_E[i]*10], [y_E[i]*10])
        moon_line.set_data([x_M[i]], [y_M[i]])
        for j in range(N):
            tide_lines[j].set_data([x_tide[j][i]*10], [y_tide[j][i]*10])
        return earth_line, moon_line, *tide_lines

    anim = FuncAnimation(fig, animate, init_func=None, frames=t.shape[0], interval=30, blit=True)
    anim.save('Erde_Mond_NFlutberge.gif', writer='pillow')
    plt.close()

display(Image(data=open('Erde_Mond_NFlutberge.gif', 'rb').read(), format='png'))
```





Die Flutberge bilden sich aus, sind aber nicht sehr stabil. Ein paar Massen sind immer außerhalb der Flutberge, die Mehrzahl sammelt sich jedoch dort, wo die Flutberge sein sollten. Sie oszillieren auch unterschiedlich stark um die theoretische Position der Flutberge (abhängig vom Startpunkt).

Die Näherung ist gerechtfertigt, da einerseits die Gesamtmasse der Ozeane relativ zur Erd- und Mondmasse recht klein ist. Andererseits verkleinert sich der Effekt der Flutberge auf die Erd- und Mondkoordinaten durch die Verteilung der Masse. Auf die Erde wirken die Flutberge so, dass ihre Radialbeschleunigung gegen die Erde "drückt" beziehungsweise an ihr "zieht". Da die Massen hier verteilt sind mitteln sich diese Beschleunigungen zum Teil, daher kann dieser Effekt besser vernachlässigt werden als im Fall von 2 Flutbergen. Auf den Mond wirkt die Anziehungskraft der Flutmassen, welche bewirkt, dass der Mond ein Drehmoment erfährt. Dieser Effekt ist relativ klein, wodurch der Einfluss auf die Bewegung der Flutberge ebenfalls sehr klein ist. Dadurch entfernt sich der Mond hier nicht von der Erde, was für große Zeiten ein relevanter Effekt ist.

```
In [ ]: # Two tides to compare to the previous result:
if not fastExecution:
    N = 2
    t_N, x_E_N, y_E_N, x_M_N, y_M_N, phi_tides_N, omega_E_N, omega_tides_N = N_body_problem(*iv_stable_orbit_Nbody(), k=k_fit,
    phi_1_N, phi_2_N = phi_tides_N[0], phi_tides_N[1]
    omega_1_N, omega_2_N = omega_tides_N[0], omega_tides_N[1]
    x_1_N, y_1_N, x_2_N, y_2_N = x_E_N + RErde*np.cos(phi_1_N), y_E_N + RErde*np.sin(phi_1_N), x_E_N + RErde*np.cos(phi_2_N),
```

```
In [ ]: if not fastExecution:
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
    plt.rc('axes', axisbelow=True)
    plt.suptitle("Vergleich 4 Körper Problem Exakte DGL / N=2 Näherung")

    def plot_subplot(ax, lines, title, xlabel, ylabel):
        for x, y, label, color in lines:
            ax.plot(x, y, color, linewidth=1, label=label)
        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_title(title)

        ax.grid('minor', 'minor', linestyle='-', linewidth=0.2)
        ax.grid('major', 'major', linestyle='-', linewidth=0.8)
        ax.minorticks_on()
        ax.legend()

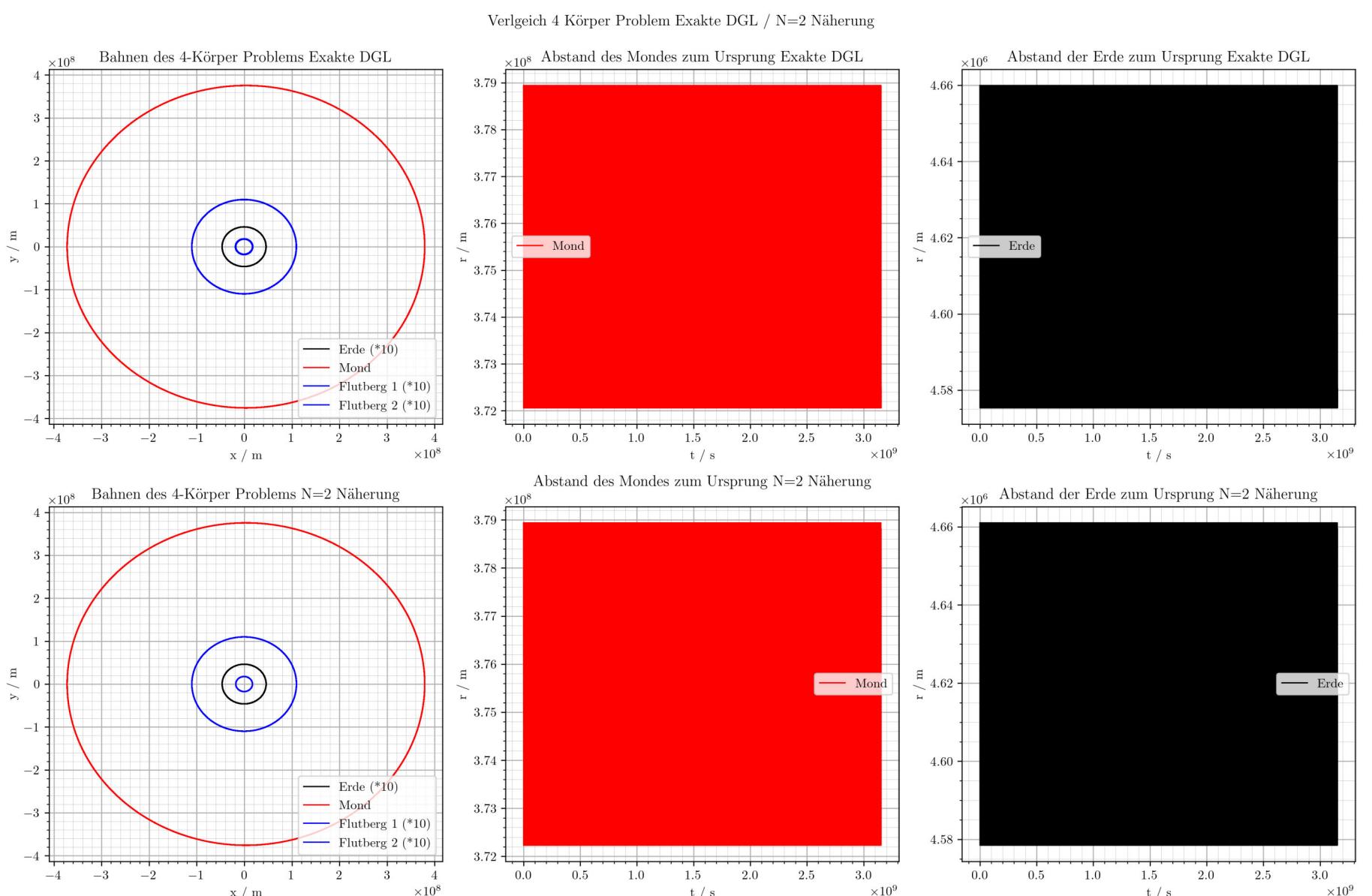
    # Distance from the origin for earth and moon
    r_Earth_kfit = np.sqrt(x_E_kfit**2 + y_E_kfit**2)
    r_Moon_kfit = np.sqrt(x_M_kfit**2 + y_M_kfit**2)
    r_Earth_N = np.sqrt(x_E_N**2 + y_E_N**2)
    r_Moon_N = np.sqrt(x_M_N**2 + y_M_N**2)

    plot_subplot(axes[0][0], [(x_E_kfit*10, y_E_kfit*10, 'Erde (*10)', 'k'), (x_M_kfit, y_M_kfit, 'Mond', 'r'), (x_1_kfit*10, y_1_kfit*10, 'Flutberg 1 (*10)', 'k'), (x_2_kfit*10, y_2_kfit*10, 'Flutberg 2 (*10)', 'k')], "Abstand des Mondes zum Ursprung Exakte DGL", "t / s", "r / m")
    plot_subplot(axes[0][1], [(t_kfit, r_Moon_kfit, 'Mond', 'r')], "Abstand der Erde zum Ursprung Exakte DGL", "t / s", "r / m")
    plot_subplot(axes[0][2], [(t_kfit, r_Earth_kfit, 'Erde', 'k')], "Abstand der Erde zum Ursprung Exakte DGL", "t / s", "r / m")
    plot_subplot(axes[1][0], [(x_E_N*10, y_E_N*10, 'Erde (*10)', 'k'), (x_M_N, y_M_N, 'Mond', 'r'), (x_1_N*10, y_1_N*10, 'Flutberg 1 (*10)', 'k'), (x_2_N*10, y_2_N*10, 'Flutberg 2 (*10)', 'k')], "Abstand des Mondes zum Ursprung N=2 Näherung", "t / s", "r / m")
    plot_subplot(axes[1][1], [(t_N, r_Moon_N, 'Mond', 'r')], "Abstand des Mondes zum Ursprung N=2 Näherung", "t / s", "r / m")
    plot_subplot(axes[1][2], [(t_N, r_Earth_N, 'Erde', 'k')], "Abstand der Erde zum Ursprung N=2 Näherung", "t / s", "r / m")
    plt.tight_layout()
    plt.savefig("4body_friction_comp1_N=2.png", dpi=300)
    plt.close()

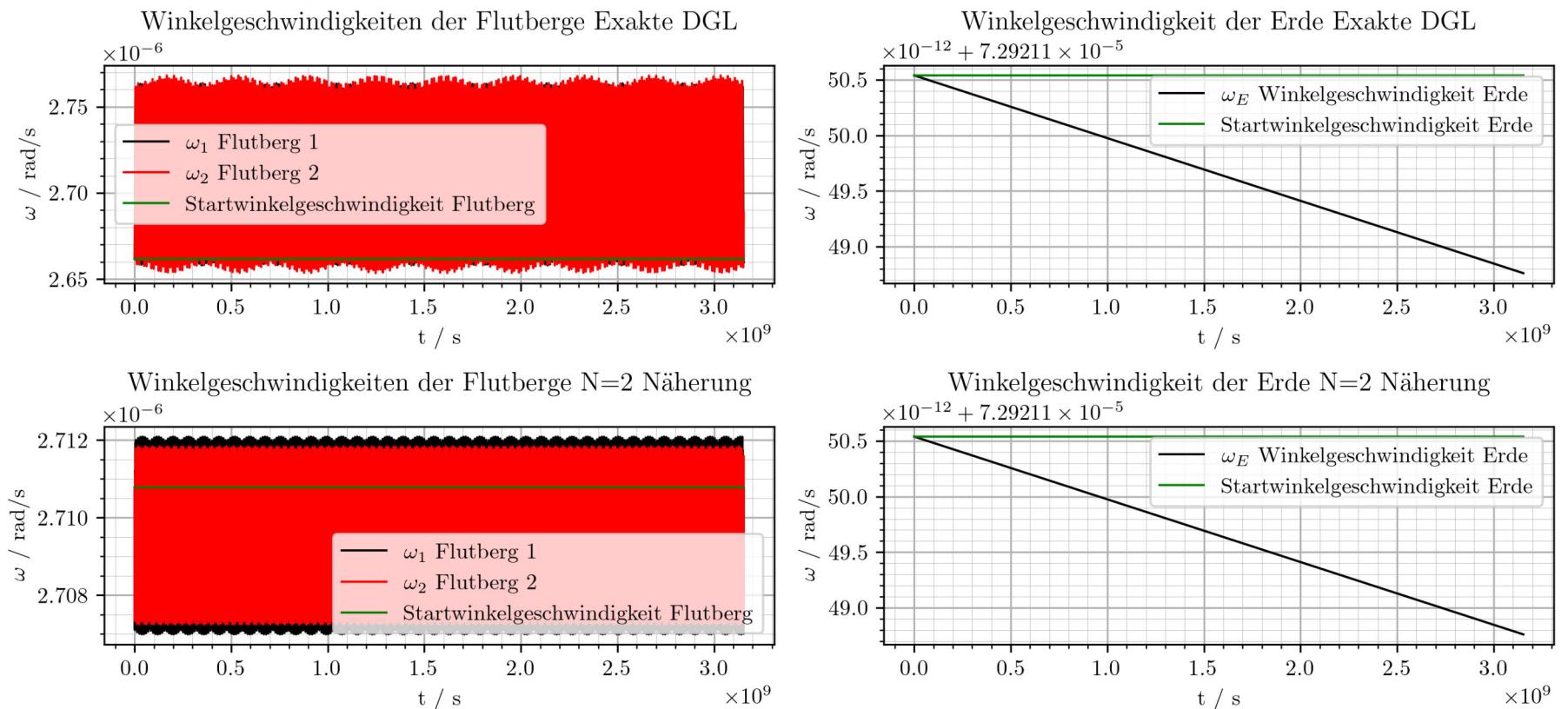
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 5))
    plt.rc('axes', axisbelow=True)
    plt.suptitle("Vergleich 4 Körper Problem Exakte DGL / N=2 Näherung")

    plot_subplot(axes[0][0], [(t_kfit, omega_1_kfit, '$\omega_1$ Flutberg 1', 'k'), (t_kfit, omega_2_kfit, '$\omega_2$ Flutberg 2', 'k')], "Winkelgeschwindigkeit Flutberg 1 und 2", "t / s", "r / m")
    plot_subplot(axes[0][1], [(t_kfit, omega_E_kfit, '$\omega_E$ Winkelgeschwindigkeit Erde', 'k')], "Winkelgeschwindigkeit Erde", "t / s", "r / m")
    plot_subplot(axes[1][0], [(t_N, omega_1_N, '$\omega_1$ Flutberg 1', 'k'), (t_N, omega_2_N, '$\omega_2$ Flutberg 2', 'r')], "Winkelgeschwindigkeit Flutberg 1 und 2", "t / s", "r / m")
    plot_subplot(axes[1][1], [(t_N, omega_E_N, '$\omega_E$ Winkelgeschwindigkeit Erde', 'k')], "Winkelgeschwindigkeit Erde", "t / s", "r / m")
    plt.tight_layout()
    plt.savefig("4body_friction_comp2_N=2.png", dpi=300)
    plt.close()

# Display image set the size to be 500x500 pixels
display(Image(filename='4body_friction_comp1_N=2.png', width=1000, height=500))
display(Image(filename='4body_friction_comp2_N=2.png', width=1000, height=500))
```



### Vergleich 4 Körper Problem Exakte DGL / N=2 Näherung



```
In [ ]: #Show the increase of the moons orbit
if not fastExecution:
    #Moon as seen from Earth:
    x = x_M_N - x_E_N
    y = y_M_N - y_E_N

    fig = plt.figure(figsize=(12,6))
    fig.suptitle('Veränderung der Mondbahn über 100 Jahre')
    ax = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

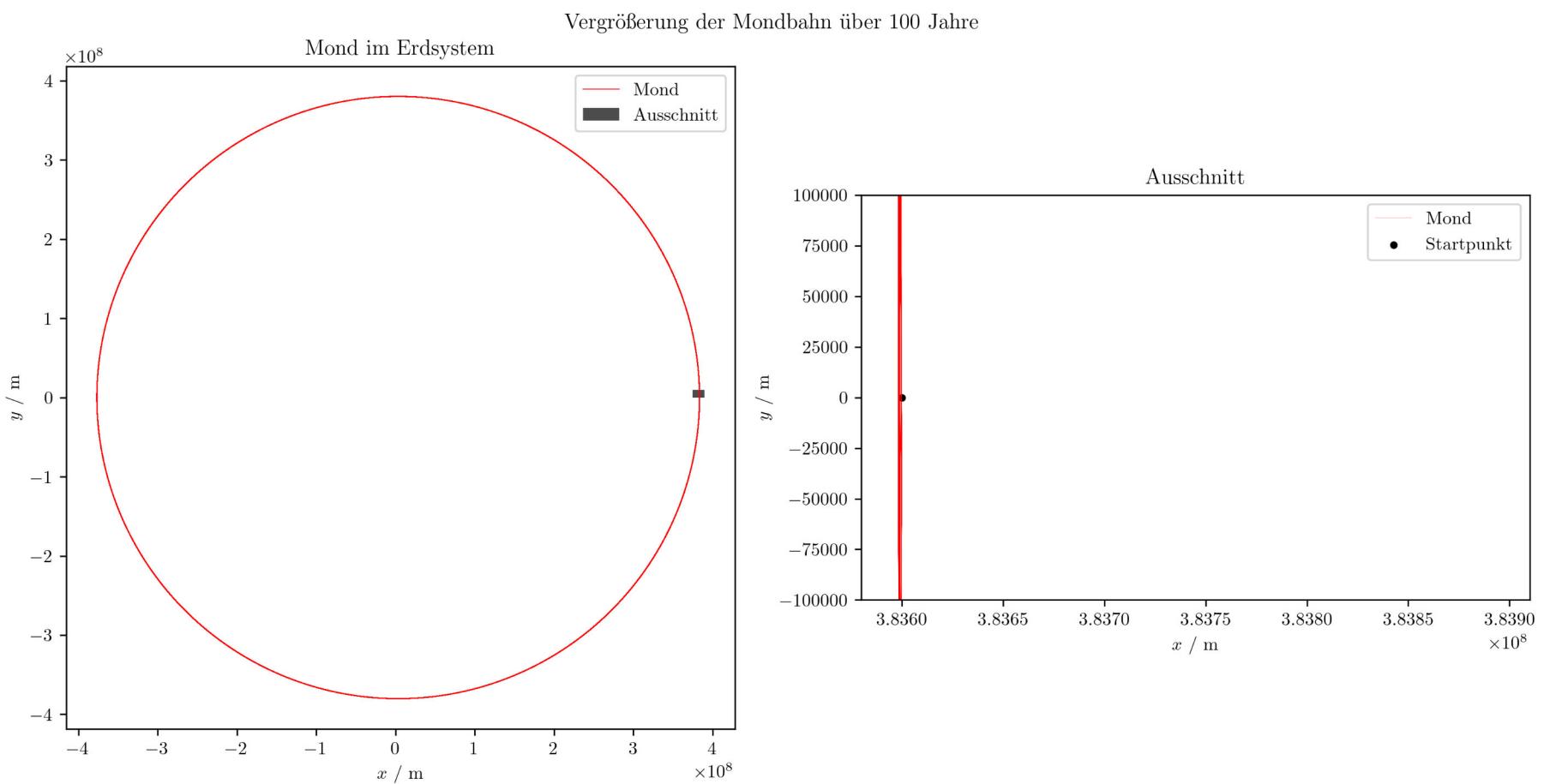
    ax.set_title('Mond im Erdsystem')
    ax.set_xlabel("$x$ / m")
    ax.set_ylabel("$y$ / m")
    ax.set_aspect('equal')
    ax.plot(x, y, color='r', linewidth=0.5, label='Mond')

    rect = plt.Rectangle((3.75e8, -1e5), 1.5e7, 1e7, facecolor="black", alpha=0.7, label='Ausschnitt')
    ax.add_patch(rect)
    ax.legend()

    ax2.set_title('Ausschnitt')
    ax2.set_xlabel("$x$ / m")
    ax2.set_ylabel("$y$ / m")
    ax2.set_aspect('equal')
    ax2.plot(x, y, 'r', linewidth=0.1, label='Mond')
    ax2.scatter(x[0], y[0], c='k', s=10, label='Startpunkt')
    ax2.set_xlim([3.8358e8, 3.8391e8])
    ax2.set_ylim([-1e5, 1e5])
    ax2.legend()

    plt.tight_layout()
    plt.savefig("4body_friction_Moon_N=2.png", dpi=300)
    plt.close()

display(Image(filename='4body_friction_Moon_N=2.png', width=1000, height=2000))
```



```
In [ ]: if not fastExecution:
    N = 2
    t, x_E, y_E, x_M, y_M, phi_tide, omega_E, omega_tide = N_body_problem(*iv_stable_orbit_Nbody(), k=k_fit, t_max=5*TMondbahn)

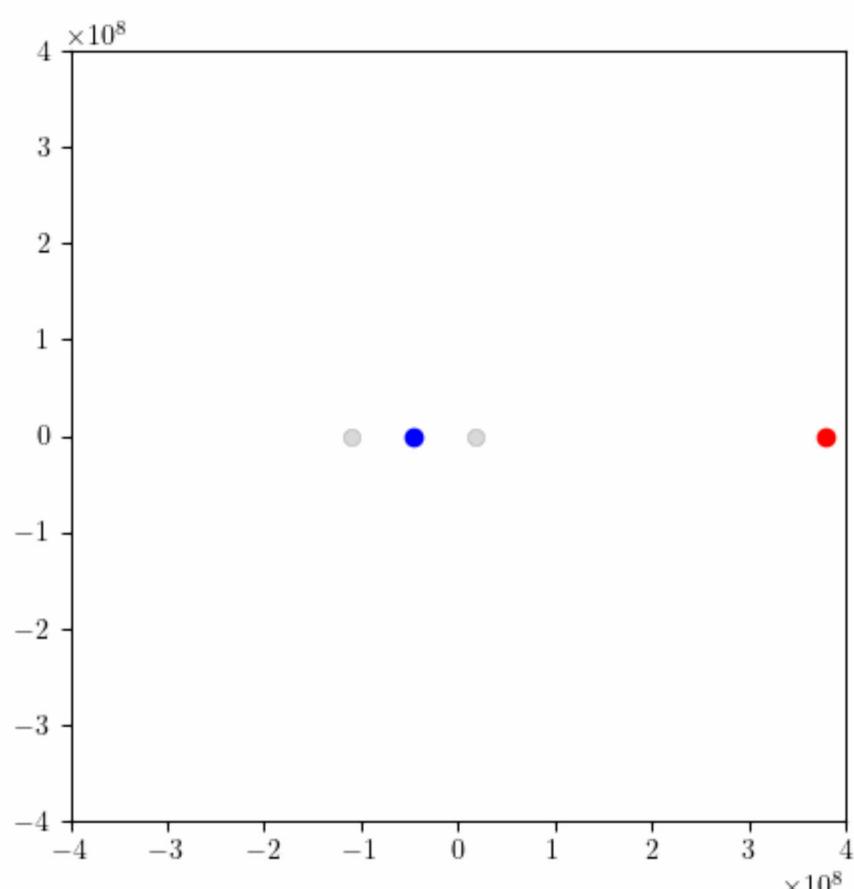
In [ ]: # Cartesian coordinates of the tides:
x_tide, y_tide = [x_E + RErde*np.cos(phi_tide[i]) for i in range(N)], [y_E + RErde*np.sin(phi_tide[i]) for i in range(N)]

# Animation of the N body problem
if not fastExecution: # Don't render if not necessary
    fig = plt.figure(figsize=(5, 5)) # Square figure, so the circles are actually circles
    ax = plt.axes(xlim=(-4e8, 4e8), ylim=(-4e8, 4e8))
    earth_line, = ax.plot([], [], marker='o', lw=0.1, color='blue', label='Erde (*10)')
    moon_line, = ax.plot([], [], marker='o', lw=0.1, color='red', label='Mond')
    tide_lines = [ax.plot([], [], marker='o', lw=0.1, color='black', label='Flutberg {i} (*10)[0] for i in range
    # ax.legend()

    def animate(i):
        earth_line.set_data([x_E[i]*10], [y_E[i]*10])
        moon_line.set_data([x_M[i]], [y_M[i]])
        for j in range(N):
            tide_lines[j].set_data([x_tide[j][i]*10], [y_tide[j][i]*10])
        return earth_line, moon_line, *tide_lines

    anim = FuncAnimation(fig, animate, init_func=None, frames=t.shape[0], interval=30, blit=True)
    anim.save('Erde_Mond_N=2Flutberge.gif', writer='pillow')
    plt.close()

display(Image(data=open('Erde_Mond_N=2Flutberge.gif','rb').read(), format='png'))
```



```
In [ ]: if not fastExecution:
    #30min
    _, _, _, _, _, omega_E, _ = N_body_problem(*iv_stable_orbit_Nbody(), k=k_fit, t_max=100*365*24*3600)

    # The change in the angular velocity
    angular_velocity_change = (omega_E[0] - omega_E[-1])

    #  $\omega = 2\pi/T \rightarrow T = 2\pi/\omega$ 
    rotation_period_at_start = (2*pi/omega_E[0])
    rotation_period_at_end = (2*pi/omega_E[-1])
    print(f'rotation_period_at_start: {rotation_period_at_start}, rotation_period_at_end: {rotation_period_at_end}')
    print(f'rotation_period_at_start [h]: {rotation_period_at_start/3600}, rotation_period_at_end [h]: {rotation_period_at_end/3600}')

    print(f'tau_lit: {tau}, tau_calc: {rotation_period_at_end - rotation_period_at_start}')
else:
    print(f'tau_lit: {tau}, tau_calc: {0.0021018376137362793}')


tau_lit: 0.0021, tau_calc: 0.0021018376137362793
```

Die größten Unterschiede zur exakten Differentialgleichung sind:

Der Mond entfernt sich nicht von der Erde, da die Gravitation der Flutberge auf den Mond vernachlässigt wird. Auch die Schwerpunktsbewegung unterscheidet sich, was aufgrund der entsprechend gewählten Anfangsbedingungen nicht zu sehen ist.

Die Bewegung der Flutberge bleibt sehr ähnlich, um diese zu betrachten ist die Näherung also gut. Daher bleibt auch die Verlangsamung der Erdrotation fast gleich. Die Winkelgeschwindigkeiten unterscheiden sich nur so stark, weil wir unterschiedliche Anfangswerte für sie gewählt haben. Insgesamt sehen sich die Lösungen sehr ähnlich, und solange man nicht an der Änderung der Mondbahn interessiert ist, ist die Näherung auch gerechtfertigt. Für sehr lange Zeiten ist sie dementsprechend nicht ausreichend.