ELEKTROTEHNIČKI FAKULTET

UNIVERZITET U BEOGRADU

Projektni zadatak iz Dinamike Mehaničkih Sistema, školska 2022/23

Kretanje kuglice po kružnom obruču koji rotira konstantnom brzinom

Smiljanić Pavle 2020/0057

Stojko Miloš 2021/0479

Sadržaj

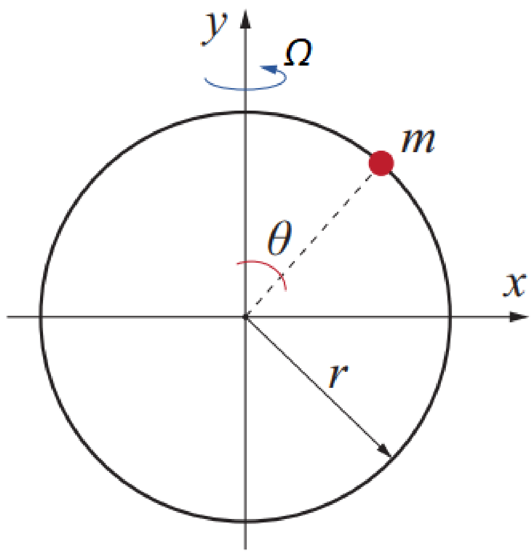
1. Izvođenje Jednačine Kretanja

1.1. Problem

Tekst problema je sledeći:

Mala kuglica mase m može da klizi bez trenja po obruču radijusa R. Obruč rotira ugaonom brzinom Ω oko vertikalne ose koja prolazi kroz njegov centar. Sistem se nalazi u konstantnom gravitacionom polju Zemlje. Smatrati da je obruč fiksiran tako da mu je jedini stepen slobode rotacija oko vertikalne ose koja prolazi kroz njegov centar.

Za početak, odredimo koordinate od kojih će zavisiti naš sistem.



Slika 1.1.: Koordinatni sistem

Ugao θ se meri od vrha, i nalazi se u opsegu $[0, 2\pi)$. Rastojanje od centra obruča do kuglice je druga koordinata, r. Iz ograničenja mi znamo da se ova koordinata neće menjati, međutim dopustimo za početak da se ona menja. Nama je od interesa zavisnost ugla otklona od vremena. Sa tim na umu, nas ne zanima treća koordinata koja bi takođe postojala, a koja bi pretstavljala ugaoni otklon kuglice u odnosu na Oxy ravan. Mi taj ugaoni otklon svakako znamo, sa obzirom da obruč rotira poznatom ugaonom brzinom. Dakle, mi ćemo pisati jednačine po 2 koordinate, θ i r. Postojaće jedna jednačina ograničenja u sistemu:

$$f(r) = r - R = 0 \tag{1.1}$$

1.2. Ojler-Lagranžove jednačine

Problem je rešen pomoću MATLAB-a, simboličkim rešavanjem. Ceo kod se može naći u dodatku A.1. Pomoću Ojler-Lagranžove jednačine po θ dolazimo do prve diferencijalne jednačine u našem sistemu. Međutim, rešavanje Ojler-Lagranžove jednačine po r nas ne dovodi do druge jednačine, zato što postoji ograničenje po koordinati r. Iz ove jednačine, pomoću metode Lagranžovih množilaca, dolazimo do jednačine koja će nam govoriti o evoluciji sile ograničenja tokom vremena.
Za početak, odredimo Lagranžijan sistema. Kinetičku energiju odredićemo prebacivanjem u Dekartov koordinatni sistem pomoću sledećih relacija:

$$\begin{aligned}
x &= r\sin(\theta)\cos(\Omega t) \\
y &= r\cos(\theta) \\
z &= -r\sin(\theta)\sin(\Omega t)
\end{aligned}$$

U Dekartovom koordinatnom sistemu, kinetičku energiju nalazimo pomoću formule

$$T = \frac{1}{2}\left(mv_x^2 + mv_y^2 + mv_z^2\right) \tag{1.2}$$

Potencijalna energija jednaka je

$$U = mgy \tag{1.3}$$

Ojler-Lagranžova jednačina po θ:

$$\frac{\partial \mathcal{L}}{\partial \theta} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) = 0 \tag{1.4}$$

Simboličkim rešavanjem iz ove jednačine dobijamo prvu diferencijalnu jednačinu koju ćemo koristiti u simulaciji:

$$\frac{mr}{2}\left(r\Omega^2\sin 2\theta + 2g\sin\theta - 2r\ddot{\theta} - 4\dot{r}\dot{\theta}\right) = 0 \tag{1.5}$$

Ova jednačina se može srediti do sledećeg oblika:

$$\Omega^2 r\cos\theta\sin\theta + g\sin\theta - r\ddot{\theta} - 2\dot{r}\dot{\theta} = 0 \tag{1.6}$$

Ojler-Lagranžovom jednačinom po r, pomoću metode Lagranžovih množilaca, možemo naći jednačinu koja opisuje vremensku evoluciju sile ograničenja:

$$\frac{\partial \mathcal{L}}{\partial r} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{r}}\right) + \frac{\partial f}{\partial r}\lambda = 0 \tag{1.7}$$

Sila ograničenja biće jednaka

$$F_{con} = \lambda \frac{\partial f}{\partial r} = \lambda \tag{1.8}$$

Simboličkim rešavanjem, nakon sređivanja, dolazimo do jednačine:

$$F_{con} = mg\cos\theta - mr\dot{\theta}^2 - mr\Omega^2\sin^2\theta + m\ddot{r} \tag{1.9}$$

Silu ograničenja računamo nakon što odredimo $\theta(t)$ i $r(t)$.

1.3. *Constraint stabilization* metoda

Mi imamo jednu diferencijalnu jednačinu. Sa obzirom da sistem ima 2 koordinate, neophodna nam je i druga diferencijalna jednačina. Jednačina koju smo dobili iz Ojler-Lagranžove jednačine po r koordinati nije korisna sa obzirom da postoji ograničenje po koordinati r. Mi možemo naći drugu difrerencijalnu jednačinu upravo iz jednačine ograničenja pomođu *constraint stabilization* metode. Polazimo od jednačine ograničenja 1.1:

$$f(r) = r - R \tag{1.10}$$

Kako bismo pronašli diferencijalnu jednačinu koja će nam garantovati ovaj rezultat, polazimo od sledeće jednačine:

$$\ddot{f} + 2a\dot{f} + a^2 f = 0 \tag{1.11}$$

Parametar a će biti takav da nam osigurava da funkcija f brzo teži 0, poput prigušene oscilacije. Ograničenje upravo i glasi da je funkcija f jednaka 0 u svim tačkama kretanja. Na osnovu veze između f i r prethodnu jednačinu možemo napisati kao:

$$\ddot{r} + 2a\dot{r} + a^2(r - R) = 0 \tag{1.12}$$

U simulaciji je korišćena vrednost $a = 100$, jer ova vrednost daje zadovoljavajuće rezultate. Konačno, dobijamo drugu diferencijalnu jednačinu:

$$\ddot{r} + 200\dot{r} + 100^2(r - R) = 0 \tag{1.13}$$

2. Rezultati simulacije

2.1. Simulacija

Za početak, vrednosti konstanti koje se koriste: $m = 0.1kg$, $g = 9.81\frac{m}{s^2}$, $R = 1m$. Vrednost Ω koju koristimo za simulaciju će zavisiti. Dve vrednosti Ω koje koristimo radi crtanja reprezentativnih grafika su $\Omega_1 = 1.56\frac{rad}{s}$ i $\Omega_2 = 4.69\frac{rad}{s}$. Ovim dvema vrednostima odgovaraju dva karakteristična slučaja: stabilna ravnoteža na dnu obruča i stabilna ravnoteža koja nije na dnu obruča. Za simulacije malih oscilacija koristi se početni ugao koji je malo pomeren od onog koji je određen za stabilnu ravnotežu numeričkom metodom. Simulacija se vrši klasičnom metodom rešavanja sistema diferencijalnih jednačina u Python-u. Na osnovu prethodno navedenih rezultata, sistem diferencijalnih jednačina je:

$$\Omega^2 r \cos\theta \sin\theta + g \sin\theta - r\ddot{\theta} - 2\dot{r}\dot{\theta} = 0 \tag{2.1}$$

$$\ddot{r} + 200\dot{r} + 100^2(r - R) = 0 \tag{2.2}$$

Celokupni kod za simulaciju se može naći u dodatku B.1.

2.2. Stabilna ravnoteža

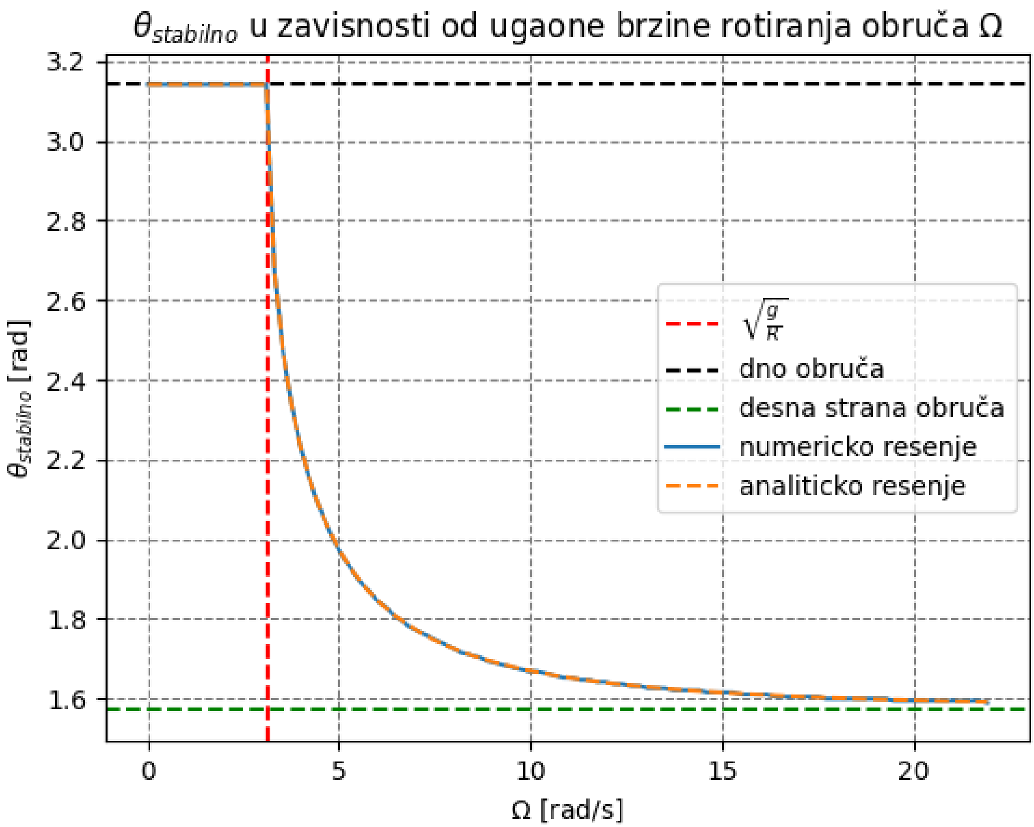
Moguće je odrediti tačku stabilne ravnoteže za male oscilacije analitički iz jednačine 1.6. Rezultat je sledeći:

$$\theta_{stabilno} = \begin{cases} \pi, & \Omega \leq \sqrt{\frac{g}{R}} \\ \arccos\left(-\frac{g}{R\Omega^2}\right), & \sqrt{\frac{g}{R}} < \Omega \end{cases}$$

Kao što vidimo, vrednost ugla stabilne ravnoteže biće π sve do neke kritične vrendnosti Ω, nakon koje on kontinualno raste, sa asimptotskom vrednošću $\frac{\pi}{2}$.

U okviru simulacije, moguće je odrediti stabilnu ravnotežu u zavisnosti od ugaone brzine na sledeći način: Za različite vrednosti Ω se za različite vrednosti početnog ugla θ vrši simulacija. Ona vrednost θ za koju su oscilacije najmanje, odnosno koja je najbliža stabilnoj ravnoteži, uzima se kao vrednost stabilne ravnoteže. Ova simulacija je veoma tehnički zahtevna zbog velikog broja tačaka (simulacija traje oko 60 minuta), ali ju je potrebno izvršiti samo jednom, jer se vrednosti čuvaju za buduće korišćenje.

Rezultat je sledeći:

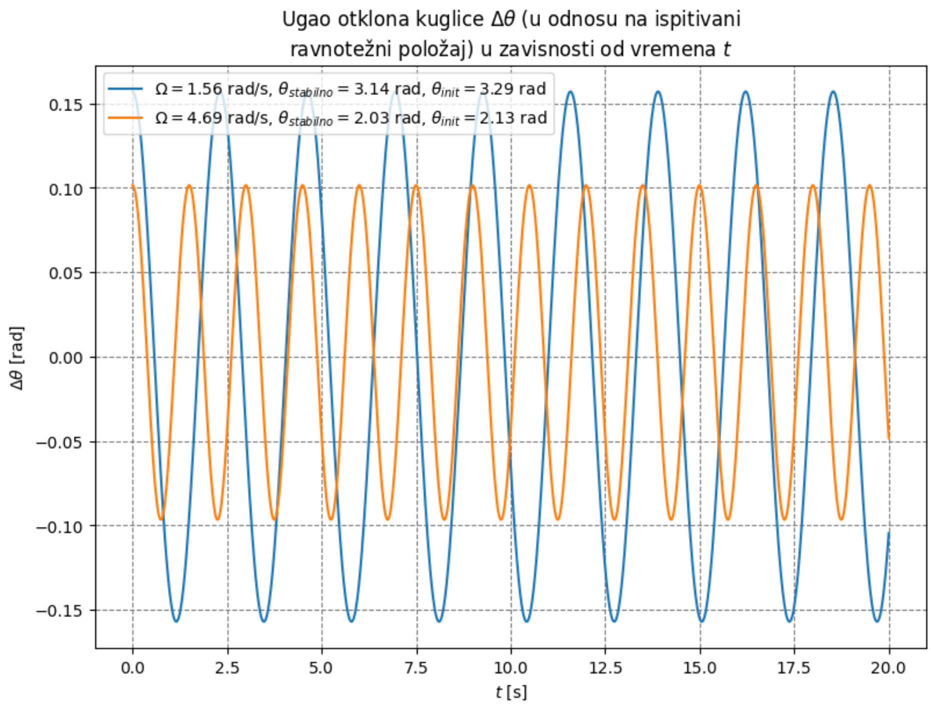


Slika 2.1.: Zavisnost ugla stabilne ravnoteže od ugaone brzine obruča

Vidimo da se analitička i numerička zavisnost dobro poklapaju. Ovo je prvi indikator da je simulacija precizna.

2.3. Male oscilacije

Slede rezultati simulacije malih oscilacija za 2 karakteristična slučaja.

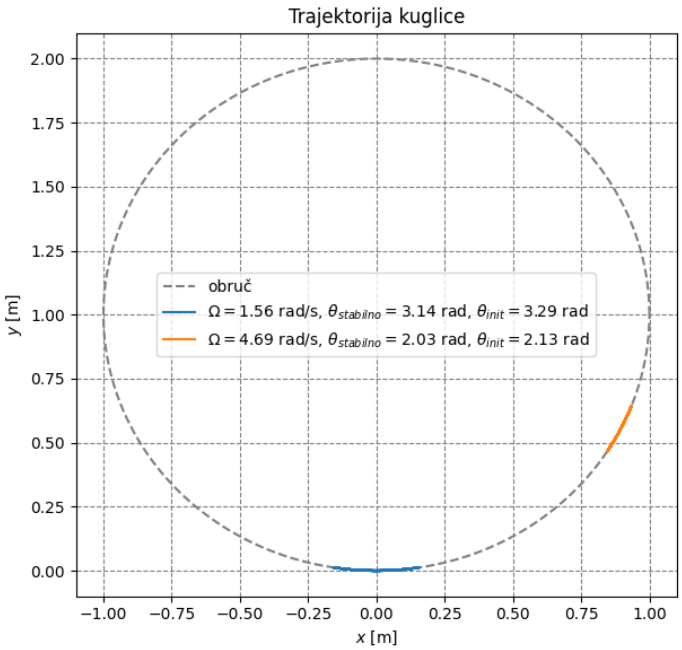


Slika 2.2.: Vremenska evolucija ugla otklona

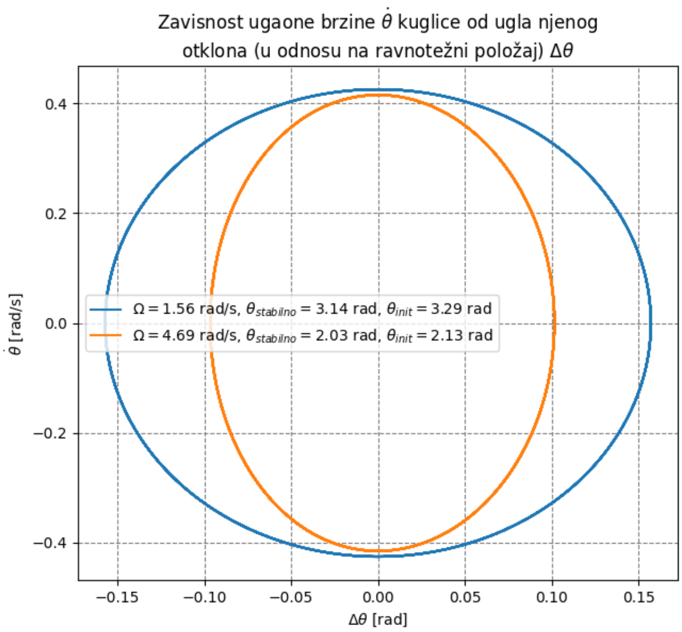
Za početak analizirajmo grafik 2.2. Za oba slučaja računat je ugao otklona u odnosu na njihov ravnotežni položaj. Uviđamo periodično oscilovanje oko ravnotežnog položaja, što se i očekivalo. Amplituda ovih oscilacija zavisi prevashodno od inicijalnog ugla otklona.

Na grafiku 2.3 isprekidana linija predstavlja obruč. Primećujemo da će ugao otklona tokom celog kretanja biti konfiniran u malu oblast oko ravnotežnog položaja. Ovo je i očekivano za male oscilacije oko stabilne ravnoteže. Na ovom grafiku y koordinata se meri od dna obruča.

Primećujemo da su fazni dijagrami na grafiku 2.4 elipse. Ovo je očekivano od sistema drugog reda, što male oscilacije oko stabilne ravnoteže i jesu.

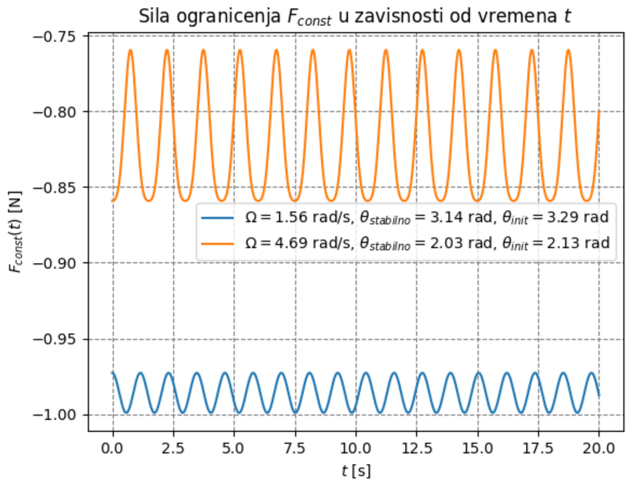


Slika 2.3.: Trajektorija kuglice



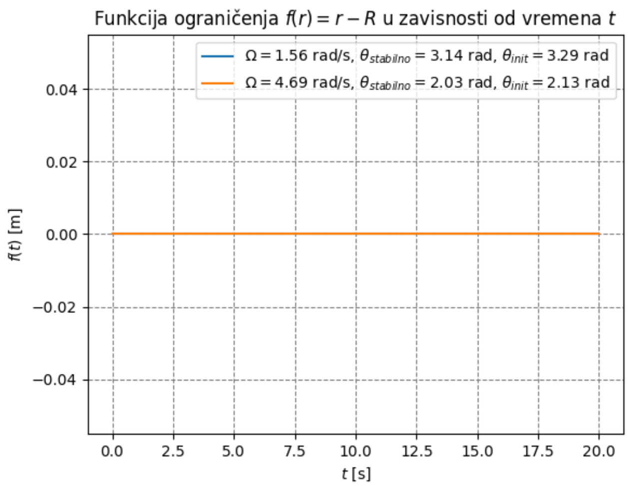
Slika 2.4.: Fazni dijagram ugla otklona

2.4. Sila ograničenja



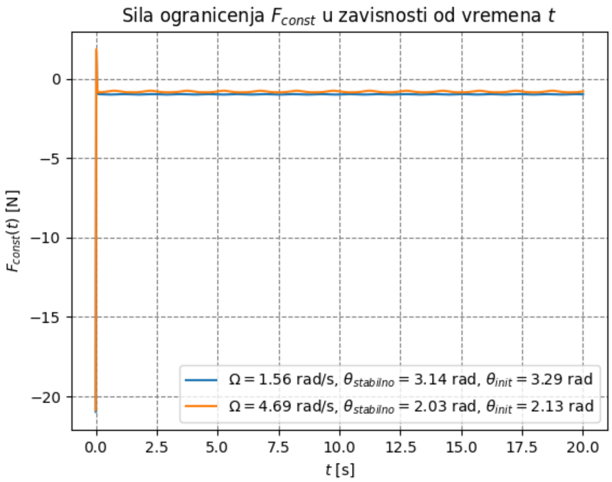
Slika 2.5.: Vremenska evolucija sile ograničenja

Prva stvar koju primećujemo je da je sila ograničenja negativna. To nam govori da se ona bori protiv porasti koordinate r. U fizičkom smislu, to znači da će centripetalna sila delovati od centra obruča, što je i očekivano. Sila ograničenja se protivi centripetalnoj sili, i garantuje da će ograničenje važiti tokom celokupnog kretanja.

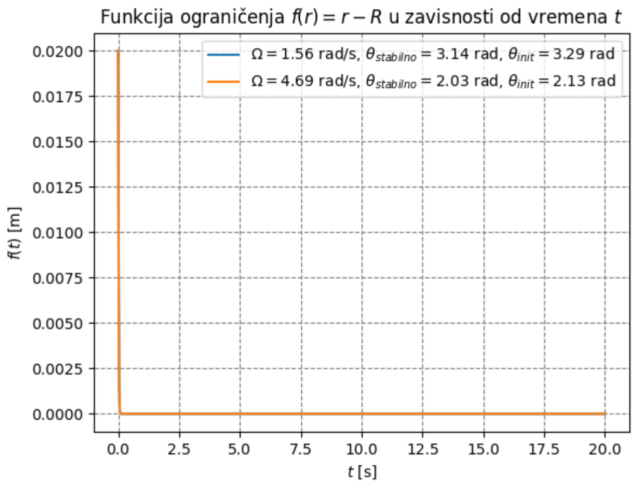


Slika 2.6.: Vremenska evolucija funkcije ograničenja

Mi smo kao početnu vrednost koordinate r uzeli upravo R, tako da je inicijalni otklon bio 0. Ovo ne znači da nam nije neophodna diferencijalna jednačina dobijena *constraint stabilization* metodom, jer će ona garantovati da otklon postane 0 u slučajevima kada on to nije. Ukoliko bi se inicijalna vrednost r postavila na neku drugu vrednost, ona bi vrlo brzo postala R. Kako bismo to demonstrirali, prilažemo prethodna 2 grafika za slučaj kada je inicijalna vrednost r jednaka $1.02 \cdot R$:

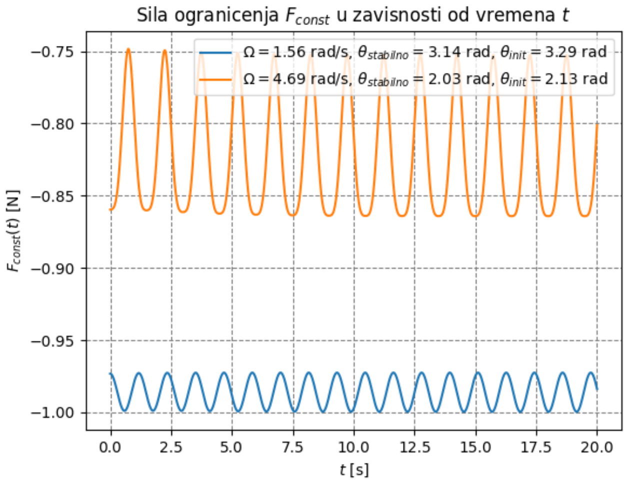


Slika 2.7.: Vremenska evolucija sile ograničenja



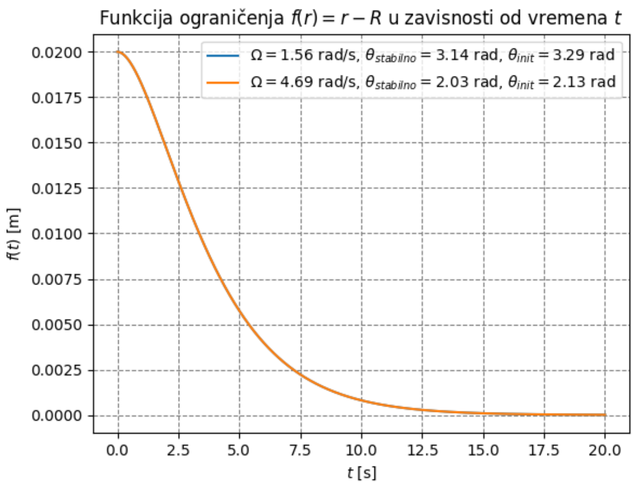
Slika 2.8.: Vremenska evolucija funkcije ograničenja

Sa prethodna 2 grafika se da videti da je sila ograničenja veoma brzo osigurala pad funkcije ograničenja na 0. Za svaku početnu vrednost r će se ispoljiti identično ponašanje. Razlog zašto je prikazan grafik za malo odstupanje od samo 2% je da bi se lepše videlo ponašanje funkcija na graficima. Ukoliko bi početna vrednost r ostala $1.02 \cdot R$, a vrednost parametra a iz 2.2 bila smanjena na $a = 0.1$, onda bi se vrednost r dosta sporije spustila na $r = R$. Prikažimo grafike sile i funkcije ograničenja za taj slučaj:



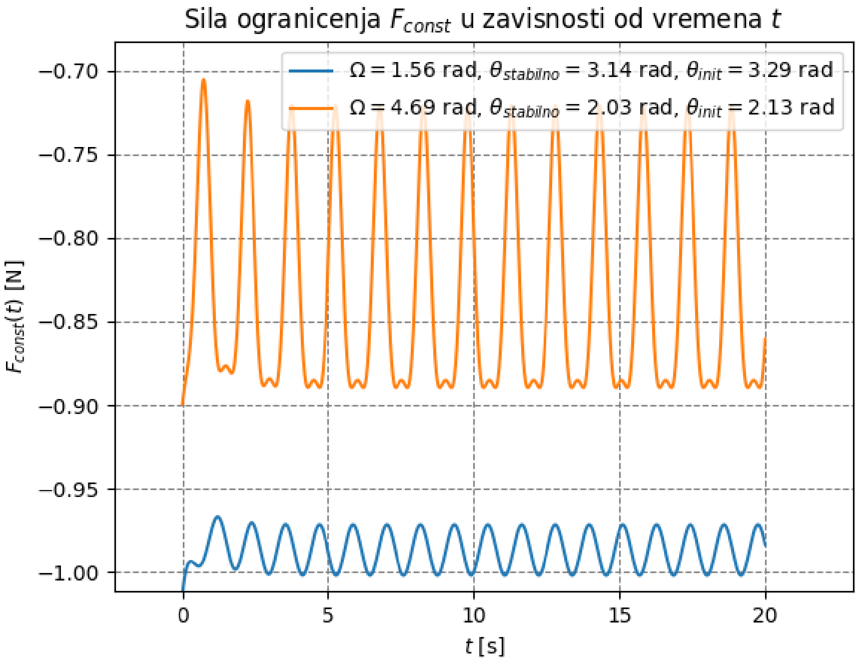
ł

Slika 2.9.: Vremenska evolucija sile ograničenja za $a = 0.1$



Slika 2.10.: Vremenska evolucija funkcije ograničenja za $a = 0.1$

Sila ograničenja se blago pomera "na dole", dok funkcija ograničenja dosta sporije opada. Sila ograničenja neće uvek imati isti oblik. Za slučaj kada je inicijalna vrednost $r = 1.1 \cdot R$, a vrednost parametra a jednaka $a = 2$ dobijamo veoma interesantan oblik sile ograničenja:

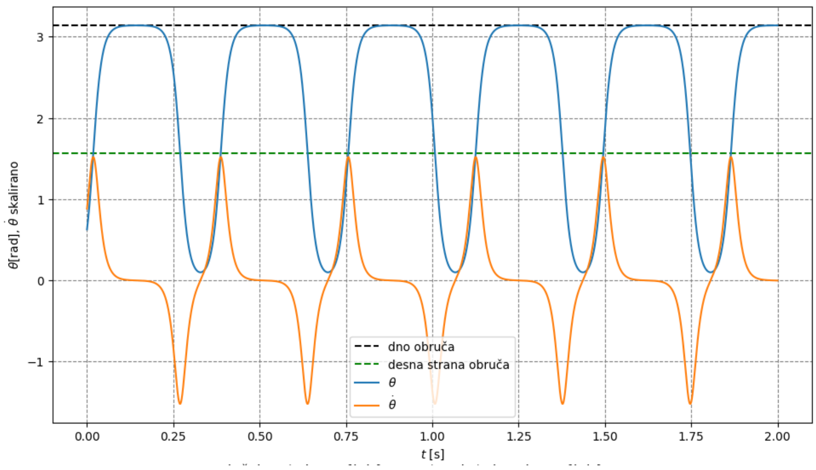


Slika 2.11.: Vremenska evolucija funkcije ograničenja za $a = 2, r = 1.1 \cdot R$

Ponovo možemo primetiti da funkcija ograničenja "migrira" na dole kako vreme prolazi (ona je svakako negativna). Za jedan od 2 slučaja primećujemo interesantno ponašanje pri donjim pikovima funkcije ograničenja.

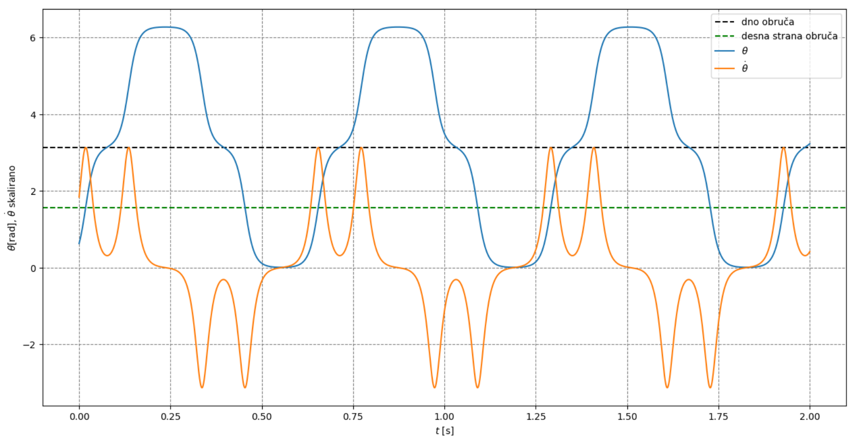
2.5. Zanimljivi slučajevi

Vratimo se sada na slučaj kada je $a = 100$, a inicijalno r jednako R. Posmatrajmo grafike vremenske zavisnosti θ i $\dot{\theta}$ za neke zanimljive granične slučajeve. Više ne posmatramo male oscilacije. Do sada smo u svim analizama inicijalnu ugaonu brzinu kuglice $\dot{\theta}$ postavljali na 0, međutim za naredne slučajeve to neće biti slučaj.



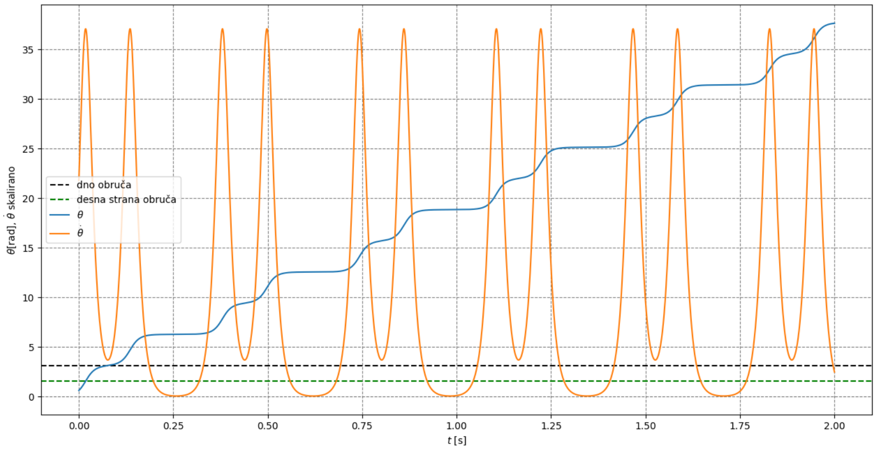
Slika 2.12.: Vremenska evolucija θ i $\dot{\theta}$

U ovom slučaju, inicijalne vrednosti su takve da omoguće kuglici da se popne skoro do vrha obruča, zatim vrati na dno, gde se zadržava neko vreme, i onda da se ponovo penje, i to istom stranom obruča. Kretanje je periodično. Povećanjem inicijalne ugaone brzine dolazimo do sledećeg slučaja:



Slika 2.13.: Vremenska evolucija θ i $\dot{\theta}$

U ovom slučaju se dešava sledeće: kuglica dođe do vrha obruča, ali ga ne pređe. Ona zatim krene da se vraća suprotnim smerom, prođe kroz dno i ponovo se popne do vrha sa druge strane obruča. Ona se zadrži neko vreme blizu vrha, ali ga ponovo ne pređe, već krene da se vraća unazad. Ona se tako periodično kreće doveka. Ukoliko bismo još više povećali početnu ugaonu brzinu kuglice, mogli bismo da osiguramo da ona prođe kroz vrh obruča:



Slika 2.14.: Vremenska evolucija θ i $\dot{\theta}$

Mi smo u ova tri slučaja koristili veoma bliske inicijalne vrednosti $\dot{\theta}$. Ovo je interesantno, sa obziom da su dobijeni rezultati u sva 3 slučaja dosta različiti.
Poslednji grafik je slučaj kada kuglica ima jedva dovoljno energije da pređe vrh obruča. Zbog toga se ona dosta dugo zadržava na vrhu. Ona će se kretati u istom smeru zauvek. Zaključujemo da ukoliko kuglica ima dovoljno energije da jednom prođe kroz vrh, ona će se zauvek kretati u istom smeru. Sa obzirom da je kretanje periodično i u ovom slučaju, energija kuglice je ograničena. Naš sistem ima beskonačni priliv izvor energije (rotacija obruča se održava konstantnom nekom spoljnom silom), moguće je da postoje početni parametri kuglice takvi da bi omogućili kuglici da beskonačno crpi energiju. Takve parametre, međutim, nismo uspeli da našetelujemo (ukoliko oni uopšte postoje).

A. MATLAB kodovi

A.1. Simboličko rešavanje

```
syms t theta(t) R Omega theta_d(t) m g theta_dd(t) r(t) r_d(t) r_dd(t)
phi = Omega * t
x = r * sin(theta) * cos(phi)
z = -r * sin(theta) * sin(phi)
y = r * cos(theta)
dot_x = diff(x,t)
dot_y = diff(y,t)
dot_z = diff(z,t)
dot_x = subs(dot_x, diff(theta(t),t),theta_d)
dot_y = subs(dot_y, diff(theta(t),t),theta_d)
dot_z = subs(dot_z, diff(theta(t),t),theta_d)
dot_x = subs(dot_x, diff(r(t),t),r_d)
dot_y = subs(dot_y, diff(r(t),t),r_d)
dot_z = subs(dot_z, diff(r(t),t),r_d)
T = 1/2 * m * (dot_x^2 + dot_y^2 + dot_z^2)
U = m*g*y
L = simplify(T-U)
Nt1 = functionalDerivative(L,theta)
Nt2 = functionalDerivative(L,theta_d)
Nt3 = diff(Nt2,t)
Nt3 = subs(Nt3, diff(theta(t),t),theta_d)
Nt3 = subs(Nt3, diff(theta_d(t),t),theta_dd)
Nt3 = subs(Nt3, diff(theta(t),t),theta_d)
Nt1 = subs(Nt1, diff(theta(t),t),theta_d)
Nt1 = subs(Nt1, diff(theta_d(t),t),theta_dd)
Nt1 = subs(Nt1, diff(theta(t),t),theta_d)
Nt3 = subs(Nt3, diff(r(t),t),r_d)
Nt3 = subs(Nt3, diff(r_d(t),t),r_dd)
Nt3 = subs(Nt3, diff(r(t),t),r_d)
Nt1 = subs(Nt1, diff(r(t),t),r_d)
Nt1 = subs(Nt1, diff(r_d(t),t),r_dd)
Nt1 = subs(Nt1, diff(r(t),t),r_d)

```

```
Nr1 = functionalDerivative(L,r)
Nr2 = functionalDerivative(L,r_d)
Nr3 = diff(Nr2,t)
Nr3 = subs(Nr3, diff(r(t),t),r_d)
Nr3 = subs(Nr3, diff(r_d(t),t),r_dd)
Nr3 = subs(Nr3, diff(r(t),t),r_d)
Nr1 = subs(Nr1, diff(r(t),t),r_d)
Nr1 = subs(Nr1, diff(r_d(t),t),r_dd)
Nr1 = subs(Nr1, diff(r(t),t),r_d)
Nr3 = subs(Nr3, diff(theta(t),t),theta_d)
Nr3 = subs(Nr3, diff(theta_d(t),t),theta_dd)
Nr3 = subs(Nr3, diff(theta(t),t),theta_d)
Nr1 = subs(Nr1, diff(theta(t),t),theta_d)
Nr1 = subs(Nr1, diff(theta_d(t),t),theta_dd)
Nr1 = subs(Nr1, diff(theta(t),t),theta_d)

simplify(Nt1-Nt3)
simplify(Nr1-Nr3)
```

B. Python kodovi

B.1. Simulacija

```python
import numpy as np
import scipy as sc
from scipy import integrate
import matplotlib.pyplot as plt
from sys import getsizeof
import tqdm
from textwrap import wrap
import os

import yaml
import pickle


def ode_stabilisation(y, t, Omega, Omega_squared, a, a_squared, R, g):
    # Uses constraint stabilisation method
    # y[0] = r, ret_params[0] = \dot r
    # y[1] = \dot r, ret_params[1] = \ddot r
    # y[2] = \theta ret_params[2] = \dot \theta
    # y[3] = \dot \theta ret_params[3] = \ddot \theta
    ret = np.empty(4)
    ret[0] = y[1]
    ret[1] = -2 * a * y[1] + a_squared * (R - y[0])

    ret[2] = y[3]
    ret[3] = (
        -2 * y[1] * y[3]
        + g * np.sin(y[2])
        + Omega_squared * y[0] * np.cos(y[2]) * np.sin(y[2])
    )
    ret[3] /= y[0]

    return ret

```

```python

def ode_simplest(y, t, Omega, Omega_squared, a, a_squared, R, g):
    # This ode was aquired by not assuming the additional degree of freedom for r, in other words r=R was taken from the get-go
    ret = np.empty(4)
    ret[0] = 0
    ret[1] = 0

    ret[2] = y[3]
    ret[3] = Omega_squared * np.sin(y[2]) * np.cos(y[2]) + g / R * np.sin(y[2])

    return ret


def get_stable_theta_from_analytic_solution(Omega):
    temp = np.sqrt(GRAVITY / RING_R)

    if isinstance(Omega, np.ndarray) is False:
        return (
            np.pi
            if Omega < temp
            else np.pi - np.arccos(GRAVITY / np.square(Omega) / RING_R)
        )

    ret = np.empty(Omega.shape[0])
    for i, Omega in enumerate(Omega):
        ret[i] = (
            np.pi
            if Omega < temp
            else np.pi - np.arccos(GRAVITY / np.square(Omega) / RING_R)
        )

    return ret


def get_sol(ode_func, Omega, theta_init, time_view, theta_dot_init=0):
    func_params = (
        Omega,
        Omega**2,
        parameters["a"],
        parameters["a"] ** 2,
        RING_R,
```

```
        GRAVITY,
    )
    inits = [
        RING_R,
        #RING_R * 1.02,
        0,
        theta_init,
        theta_dot_init,
    ]
    sol = integrate.odeint(ode_func, inits, time_view, args=func_params)

    return sol.view()


def simulate_stable():

    theta_stable_resolution = (
        parameters["theta_stable_start"] - parameters["theta_stable_end"]
    ) / parameters["nr_theta_stable_samples"]

    theta_samples = np.linspace(
        parameters["theta_stable_start"],
        parameters["theta_stable_end"],
        parameters["nr_theta_stable_samples"],
    )

    loading = tqdm.trange(len(Omega_arr), desc="Simulating for stable thetas: "
        )
    stable_arr = [[], []]
    for i, Omega_curr in enumerate(Omega_arr):

        loading.update()

        theta_stable = None
        for j, theta_curr in enumerate(theta_samples):
            sol = get_sol(parameters["ode_used"], Omega_curr, theta_curr, t.view
                ())

            spread = sol[:, 2].max() - sol[:, 2].min()
            avg = np.average(sol[:, 2])

            spread_is_small = spread < theta_stable_resolution * 1.1
```

```
            avg_is_theta = avg - theta_curr < 0.1

            if spread_is_small and avg_is_theta:
                theta_stable = theta_curr

        if theta_stable is not None:
            stable_arr[0].append(Omega_curr)
            stable_arr[1].append(theta_stable)
        else:
            print("no stable theta found for Omega=", Omega_curr)

    return stable_arr


RING_R = 1 # m
BALL_M = 0.1 # kg
GRAVITY = 9.81 # m/s^2

parameters = {}

# ------- Parameters

parameters["Omega_stable_start"] = 0
parameters["Omega_stable_end"] = np.sqrt(GRAVITY / RING_R) * 7

parameters["nr_Omega_stable"], parameters["nr_theta_stable_samples"] = 100, 300
parameters["theta_stable_start"], parameters["theta_stable_end"] = np.pi, np.pi
     * 0.499
parameters["time_start"], parameters["time_end"] = 0, 20
parameters["nr_time"] = 10**3

parameters["ode_used"] = ode_stabilisation
parameters["theta_init"] = np.pi * 0.99

parameters["a"] = 100
# 'a' controlls the 'force' at which the ball will be constrained to the bar

# -----------------

t = np.linspace(parameters["time_start"], parameters["time_end"], parameters["
    nr_time"])

```

```
Omega_arr = np.linspace(
    parameters["Omega_stable_start"],
    parameters["Omega_stable_end"],
    parameters["nr_Omega_stable"],
)

stable_arr = [[], []]

STABLE_PARAMETERS_FILE_NAME = "parameters.pickle"
STABLE_POINTS_FILE_NAME = "points.pickle"

stable_file_valid = True

if (os.path.exists(STABLE_PARAMETERS_FILE_NAME) is False) or (
    os.path.exists(STABLE_POINTS_FILE_NAME) is False
):
    stable_file_valid = False
else:
    with open(STABLE_PARAMETERS_FILE_NAME, "rb") as file:
        loaded_parameters = pickle.load(file)
    with open(STABLE_POINTS_FILE_NAME, "rb") as file:
        loaded_stable_arr = pickle.load(file)

    if loaded_parameters != parameters:
        stable_file_valid = False
    else:
        stable_arr = loaded_stable_arr

if stable_file_valid:
    print(
        "Files for saving simulations of stable points found, skipping the
            simulation"
    )
else:
    print(
        "Files for saving simulations of stable points with the same parameters
             NOT found"
    )
    stable_arr = simulate_stable()

    with open(STABLE_PARAMETERS_FILE_NAME, "wb") as file:
        pickle.dump(parameters, file)
```

```
    with open(STABLE_POINTS_FILE_NAME, "wb") as file:
        pickle.dump(stable_arr, file)

nrCalc1 = (
    parameters["nr_Omega_stable"]
    * parameters["nr_theta_stable_samples"]
    * parameters["nr_time"]
)
nrCalc2 = parameters["nr_Omega_stable"] * parameters["nr_theta_stable_samples"]
print(
    f"theta was calculated {nrCalc1:,}, times, in {nrCalc2:,} distinct
        simulations for the stable theta information"
)


# %%


figs = [None]
axs = [None]

figs[0], axs[0] = plt.subplots(1, 1)

axs[0].set_title(
    r"$\theta_{stabilno}$ u zavisnosti od ugaone brzine rotiranja obruÄŊa $\
        Omega$"
)
axs[0].axvline(
    np.sqrt(GRAVITY / RING_R),
    label=r"$\sqrt{\frac{g}{R}}$",
    color="red",
    linestyle="--",
)
axs[0].set_xlabel(r"$\Omega$ [rad/s]")
axs[0].set_ylabel(r"$\theta_{stabilno}$ [rad]")
axs[0].axhline(np.pi, label="dno obruÄŊa", color="black", linestyle="--")
axs[0].axhline(np.pi / 2, label="desna strana obruÄŊa", color="green",
    linestyle="--")

axs[0].plot(stable_arr[0], stable_arr[1], label="numericko resenje")
axs[0].plot(
    Omega_arr,
```

```
    get_stable_theta_from_analytic_solution(Omega_arr),
    label="analiticko resenje", linestyle="--"
)


# %%


two_Omegas = [
    np.sqrt(GRAVITY / RING_R) * 0.5,
    np.sqrt(GRAVITY / RING_R) * 1.5,
] # One Omega smaller than the critical Omega and one greater
two_theta_inits = [
    get_stable_theta_from_analytic_solution(Omega) * 1.05 for Omega in
        two_Omegas
]
two_labels = [
    "$\Omega="
    + str(Omega)[:4]
    + "$ rad/s, "
    + r"$\theta_{stabilno} = "
    + str(get_stable_theta_from_analytic_solution(Omega))[:4]
    + r"$ rad, $\theta_{init} = "
    + str(theta_init)[:4]
    + "$ rad"
    for theta_init, Omega in zip(two_theta_inits, two_Omegas)
]

two_sol = [
    get_sol(parameters["ode_used"], Omega, theta, t.view())
    for Omega, theta in zip(two_Omegas, two_theta_inits)
]

# Draw theta(t)
figs.append(None)
axs.append(None)
figs[-1], axs[-1] = plt.subplots(1, 1)
title = "\n".join(
    wrap(
        r"Ugao otklona kuglice $\Delta \theta$ (u odnosu na ispitivani
            ravnoteÅ¿ni poloÅ¿aj) u zavisnosti od vremena $t$",
        60,
```

```
    )
)
axs[-1].set_title(title)
axs[-1].set_xlabel(r"$t$ [s]")
axs[-1].set_ylabel(r"$\Delta \theta$ [rad]")
for i, j in enumerate([len(figs) - 1] * 2):
    axs[j].plot(
        t,
        two_sol[i][:, 2] - get_stable_theta_from_analytic_solution(two_Omegas[i
            ]),
        label=two_labels[i],
    )


# Draw the trajectory
figs.append(None)
axs.append(None)
figs[-1], axs[-1] = plt.subplots(1, 1)
title = "\n".join(
    wrap(
        r"Trajektorija kuglice",
        60,
    )
)
axs[-1].set_title(title)
axs[-1].set_xlabel(r"$x$ [m]")
axs[-1].set_ylabel(r"$y$ [m]")
angles = np.linspace(
    0, np.pi * 2, 1000
) # angle from the x axis like its usualy found in math
axs[-1].plot(
    np.cos(angles) * RING_R,
    (np.sin(angles) + 1) * RING_R,
    label="obruÄŊ",
    color="gray",
    linestyle="--",
)
for i, j in enumerate([len(figs) - 1] * 2):
    x = np.sin(two_sol[i][:, 2]) * RING_R
    y = (np.cos(two_sol[i][:, 2]) + 1) * RING_R
    axs[j].plot(x, y, label=two_labels[i])
    # axs[j].set_aspect("equal", adjustable="box")
```

```


# Draw the phase diagram
figs.append(None)
axs.append(None)
figs[-1], axs[-1] = plt.subplots(1, 1)
title = "\n".join(
    wrap(
        r"Zavisnost ugaone brzine $\dot \theta$ kuglice od ugla njenog otklona
            (u odnosu na ravnoteÅ¿ni poloÅ¿aj) $\Delta \theta$",
        60,
    )
)
axs[-1].set_title(title)
axs[-1].set_xlabel(r"$\Delta \theta$ [rad]")
axs[-1].set_ylabel(r"$\dot \theta$ [rad/s]")
for i, j in enumerate([len(figs) - 1] * 2):
    axs[j].plot(
        two_sol[i][:, 2] - get_stable_theta_from_analytic_solution(two_Omegas[i
            ]),
        # two_sol[:, 2],
        two_sol[i][:, 3],
        label=two_labels[i],
    )


# Draw the force due to the R=r "constraint"
figs.append(None)
axs.append(None)
figs[-1], axs[-1] = plt.subplots(1, 1)
title = "\n".join(
    wrap(
        r"Sila ogranicenja $F_{const}$ u zavisnosti od vremena $t$",
        60,
    )
)
axs[-1].set_title(title)
axs[-1].set_xlabel(r"$t$ [s]")
axs[-1].set_ylabel(r"$F_{const}(t)$ [N]")
for i, j in enumerate([len(figs) - 1] * 2):
    f = (
        BALL_M * GRAVITY * np.cos(two_sol[i][:, 2])
```

```
        - BALL_M * RING_R * np.square(two_sol[i][:, 3])
        - BALL_M * RING_R * np.square(np.sin(two_sol[i][:, 2])) * two_Omegas[i]
    )
    r_dot_dot = -2 * parameters["a"] * two_sol[i][:, 1] + np.square(parameters[
        "a"]) * (
        RING_R - two_sol[i][:, 0]
    ) # this is from the constraint stabilisation formula
    f = f + BALL_M * r_dot_dot
    axs[j].plot(t, f, label=two_labels[i])


# Draw the constraint function stabilizing over time
figs.append(None)
axs.append(None)
figs[-1], axs[-1] = plt.subplots(1, 1)
title = "\n".join(
    wrap(
        r"Funkcija ograniÄDenja $f(r)=r-R$ u zavisnosti od vremena $t$",
        60,
    )
)
axs[-1].set_title(title)
axs[-1].set_xlabel(r"$t$ [s]")
axs[-1].set_ylabel(r"$f(t)$ [m]")
for i, j in enumerate([len(figs) - 1] * 2):
    axs[j].plot(t, two_sol[i][:, 0] - RING_R, label=two_labels[i])


# Draw and calcualte the movement of the non stable case
Omega_unstable = np.sqrt(GRAVITY / RING_R) * 20
theta_init_unstable = np.pi * 0.2
#theta_dot_init_unstable = Omega_unstable * 0.5886 # Weird case 1 where it goes
     in circle forever
#theta_dot_init_unstable = Omega_unstable * 0.58855 # Weird case 1 where it
    goes from top to top without ever going over
#theta_dot_init_unstable = Omega_unstable * 0.5805 # 3
theta_dot_init_unstable = Omega_unstable * 0.580038
t_unstable = np.linspace(0, 2, 10**5)
sol_unstable = get_sol(
    parameters["ode_used"],
    Omega_unstable,
    theta_init_unstable,
```

```
    t_unstable.view(),
    theta_dot_init_unstable,
)
figs.append(None)
axs.append(None)
figs[-1], axs[-1] = plt.subplots(1, 1)
title = "\n".join(
    wrap(
        r"Zavisnost $\theta$ i $\dot \theta$ od vremena $t$ u nestabilnom slucaju",
        60,
    )
)
axs[-1].set_xlabel(r"$t$ [s]")
axs[-1].set_ylabel(r"$\theta$[rad], $\dot \theta$ skalirano")
axs[-1].axhline(np.pi, label="dno obruÄDa", color="black", linestyle="--")
axs[-1].axhline(np.pi / 2, label="desna strana obruÄDa", color="green", linestyle="--")
text = str(
    r"SluÄDaj sa $\theta_{init}="
    + str(theta_init_unstable)[:4]
    + r"$ rad, $\dot \theta_{init} = "
    + str(theta_dot_init_unstable)[:4]
    + r" rad/s, i $\Omega="
    + str(Omega_unstable)[:4]
    + r"$ rad/s"
)
text = wrap(text, 60)
text = "\n".join(text)
figs[-1].text(0.5, -0.02, text, wrap=True, horizontalalignment="center", fontsize=12)
axs[-1].plot(t_unstable, sol_unstable[:, 2], label=r"$\theta$")
scale = (np.max(sol_unstable[:, 2]) - np.min(sol_unstable[:, 2])) / (
    np.max(sol_unstable[:, 3]) - np.min(sol_unstable[:, 3])
)

axs[-1].plot(t_unstable, sol_unstable[:, 3] * scale, label=r"$\dot \theta$")


for axes in axs:
    axes.grid(color="gray", linestyle="--")
    axes.legend()
```

```
for ax, fig in zip(axs, figs):
    # fig.tight_layout()
    fig.show()

parameters["a"] = input()
```