Određivanje dinamike fizikalnog sustava pomoću dubokih neuralnih mreža

Sesartić, Luka

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Luka Sesartić

ODREĐIVANJE DINAMIKE FIZIKALNOG SUSTAVA POMOĆU DUBOKIH NEURALNIH MREŽA

Diplomski rad

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

SMJER: NASTAVNIK FIZIKE I INFORMATIKE

Luka Sesartić

Diplomski rad

Određivanje dinamike fizikalnog sustava pomoću dubokih neuralnih mreža

Voditelj diplomskog rada: prof. dr. sc. Davor Horvatić

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2023.

ZAHVALJUJEM:
Majci Helgi, ocu Borisu i bratu Marku na bezuvjetnoj podršci i ljubavi koju su mi pružili tijekom cijelog školovanja. Bez vas, ovo ne bi bilo moguće.

Karli na podršci, ohrabrivanju i iskrenoj empatiji koju mi je pružila kad mi je bilo najteže.

Naposljetku, prof. dr. sc. Davoru Horvatiću na stručnosti, mentorstvu i razumijevanju koje mi je pružio tijekom pisanja ovog rada i kroz godine prije.

Sažetak

Brz napredak dubokih neuronskih mreža otvorio je nove načine razumijevanja kompleksnih fizičkih sustava. Ovaj diplomski rad istražuje primjenu dubokih neuronskih mreža u određivanju dinamike fizičkih sustava. Ciljevi istraživanja obuhvaćaju razvoj novih metodologija za modeliranje i predviđanje ponašanja dinamičkih sustava, koristeći snagu umjetne inteligencije. Kako bismo postigli ove ciljeve, istražili smo principe rada dubokog učenja i neuronskih mreža. Duboke neuronske mreže učene su i prilagođene korištenjem stvarnih podataka iz fizičkih sustava, omogućavajući relativno precizno karakteriziranje dinamike sustava. Rezultati su pokazali učinkovitost ovog pristupa u modeliranju jednostavnih fizičkih sustava i potencijal za primjenu na složenim sustavima. Ključni nalazi uključuju sposobnost bilježenja jednostavnijih i složenijih obrazaca u ponašanju sustava, parirajući tradicionalnim tehnikama modeliranja u točnosti i prilagodljivosti. Implikacije ovog istraživanja proširuju se na različite domene, uključujući fiziku, strojarstvo i odlučivanje temeljeno na podacima. Zaključno, ovaj diplomski rad pokazuje potencijal dubokih neuronskih mreža kao alata za određivanje dinamike fizičkih sustava. Primjena umjetne inteligencije na takve probleme obećava poboljšanje našeg razumijevanja kompleksnih sustava i olakšavanje stvaranje rješenja temeljenih na podacima.

Ključne Riječi: duboke neuronske mreže, fizički sustavi, dinamika, strojno učenje

Determining the Dynamics of a Physical System Using Deep Neural Networks

Abstract

The rapid advancement of deep neural networks has opened new avenues for understanding complex physical systems. This master's thesis explores the application of deep neural networks in determining the dynamics of physical systems. The research objectives include the development of new methodologies for modelling and predicting the behaviour of dynamic systems, and harnessing the power of artificial intelligence. To achieve these objectives, we investigated the principles of deep learning and neural networks. Deep neural networks were trained and adapted using real data from physical systems, enabling a relatively accurate characterization of system dynamics. The results demonstrated the effectiveness of this approach in modelling simple physical systems and its potential for application on more complex systems. Key findings include the ability to capture both simple and more complex patterns in system behaviour, approaching the level of precision and adaptability characteristic of conventional modelling techniques. The implications of this research extend to various domains, including physics, engineering, and data-driven decision-making. In conclusion, this master's thesis showcases the potential of deep neural networks as tools for determining the dynamics of physical systems. Applying artificial intelligence to such problems promises to enhance our understanding of complex systems and facilitate data-driven solutions.

Keywords: deep neural networks, physical systems, dynamics, machine learning

Sadržaj

1 Uvod **1**

2 Osnove klasične mehanike **2**
2.1 Newtonovi zakoni 2
2.2 Lagrangeova formulacija mehanike 2
2.3 Dokaz ekvivalentnosti između Lagrangeove mehanike i Newtonove mehanike 4
2.4 Harmoničko gibanje 4
2.5 Dinamika mase na opruzi 7
2.6 Dinamika matematičkog njihala 9

3 Pregled osnova strojnog učenja **13**
3.1 Uvod u strojno učenje 13
3.2 Osnovni koncepti strojnog učenja 16
3.2.1 Funkcija gubitka 16
3.2.2 Optimizacija u strojnom učenju 20
3.3 Algoritmi strojnog učenja 21
3.4 Neuronske mreže 22

4 PDE-FIND algoritam **24**
4.1 Teorijska pozadina PDE-FIND algoritma 24
4.2 Numerička obrada podataka 25
4.3 Metode za pronalazak koeficijenata 26
4.4 Numerička derivacija diskretiziranih podataka 28

5 Mjerenja i rezultati **30**
5.1 Laboratorijska mjerenja 31
5.1.1 Masa na opruzi 32
5.1.2 Matematičko njihalo 33
5.2 Određivanje dinamike sustava putem PDE-FIND algoritma 35
5.2.1 Ekstrakcija pozicija centra mase matematičkog njihala i mase na opruzi 35
5.2.2 Izračun kuta otklona 38
5.2.3 Provedba PDE-FIND algoritma 41
5.2.4 Rezultat za matematičko njihalo 42
5.2.5 Rezultat za masu na opruzi 42
5.2.6 Izvori pogreške 43

6 Zaključak **44**

Dodaci **45**

A Python kod **45**

Literatura **54**

1 Uvod

Mehanika igra ključnu ulogu u razumijevanju i tumačenju fizičkih procesa koji nas okružuju. U svijetu znanosti postoje više pristupa za analizu i opisivanje dinamičkih sustava. U ovom radu probleme ćemo sagledati kroz dva pristupa - Newtonova mehanika, koja se temelji na zakonima gibanja koje je otkrio Isaac Newtona, i Lagrangeova mehanika, koja koristi varijacijske principe za izračun dinamike sustava. Ovaj rad započinje razmatranjem osnovnih principa klasične mehanike, uključujući Newtonove zakone gibanja, Lagrangeovu formulaciju i dokaz ekvivalentnosti između ova dva formalizma. Dalje, istražujemo pojam harmoničkog gibanja, što je fundamentalni koncept u mehanici, te se bavimo dinamikom mase na opruzi i dinamikom matematičkog njihala. Međutim, moderna znanost ide korak dalje i koristi koncepte strojnog učenja za dublje razumijevanje i analizu složenih dinamičkih sustava. Stoga ćemo u drugom dijelu rada proučiti osnove strojnog učenja i ključne koncepte koji su potrebni kako bismo bolje razumjeli algoritam koji ćemo kasnije koristiti. Ovaj dio uključuje teme kao što su funkcija gubitka, optimizacija u strojnom učenju, algoritmi strojnog učenja i neuronske mreže. Središnji dio rada posvećen je PDE-FIND algoritmu, moćnom alatu koji ćemo koristiti za pronalaženje parcijalnih diferencijalnih jednadžbi koje opisuju dinamiku nekog sustava. Istaknut ćemo teorijsku pozadinu ovog algoritma, numeričku obradu podataka, metode za pronalaženje koeficijenata i numeričku derivaciju diskretiziranih podataka. U posljednjem dijelu rada analizirat ćemo rezultate mjerenja u laboratorijskom okruženju. Provest ćemo mjerenja za masu na opruzi i matematičko njihalo na klasičan način, a zatim ćemo primijeniti PDE-FIND algoritam kako bismo odredili dinamiku oba sustava novim pristupom. Ovaj rad predstavlja analizu klasične mehanike uz primjenu modernih alata strojnog učenja te pruža doprinos u razvijanju alata za razumijevanje dinamike kompleksnijih sustava u fizici.

2 Osnove klasične mehanike

2.1 Newtonovi zakoni

Newtonovi zakoni mehanike predstavljaju temelj klasične mehanike i ključni su koncept u proučavanju dinamike fizikalnih sustava. Formulirani su od strane slavnog engleskog fizičara Sir Isaaca Newtona u njegovom djelu "Mathematical Principles of Natural Philosophy", objavljenom 1687. godine. Ovi zakoni predstavljaju osnovu za razumijevanje gibanja tijela pod djelovanjem sila te omogućavaju matematičko modeliranje i predviđanje kretanja tijela u raznim fizičkim situacijama.

U ovom potpoglavlju, detaljno ćemo proučiti tri Newtonova zakona mehanike, koji se odnose na inerciju tijela, relaciju između sile i ubrzanja te akciju i reakciju između dvaju tijela. Njihova primjena omogućava razumijevanje i analizu dinamike različitih sustava, od jednostavnih čestica do složenih tijela u stvarnom svijetu.

Prvi Newtonov zakon, poznat i kao zakon inercije, govori da svako tijelo ostaje u stanju mirovanja ili jednolikog gibanja po pravcu, sve dok vanjske sile ne uzrokuju promjenu tog stanja.

Drugi zakon povezuje ukupnu silu na tijelo s promjenom količine gibanja. Uz pretpostavku stalne mase možemo ga matematički formulirati:

$$\vec{F} \;=\; \frac{d\vec{p}}{dt} = \frac{d(m\vec{v})}{dt} = m\frac{d\vec{v}}{dt} = m\vec{a} \tag{2.1}$$

U gornjoj jednadžbi m označava masu, a $\vec{v}$ brzinu promatranog tijela, $\vec{F}$ označava silu, a $\vec{p}$ količinu gibanja.

Treći zakon, poznat i kao princip akcije i reakcije, govori o međusobnom djelovanju dvaju tijela, gdje je sila kojim jedno tijelo djeluje na drugo jednaka po iznosu i smjeru ali suprotna po orijentaciji sili kojom drugo tijelo djeluje na prvo.

$$\vec{F}_{12} \;=\; -\vec{F}_{21} \tag{2.2}$$

U gornjoj jednadžbi F_{12} označava silu prvog tijela na drugo, a F_{21} silu drugog tijela na prvo.

2.2 Lagrangeova formulacija mehanike

Lagrangeova mehanika i Newtonova mehanika dvije su ekvivalentne formulacije klasične mehanike koje opisuju fizikalne sustave. One su ekvivalentne u smislu da daju iste jednadžbe gibanja i mogu se koristiti naizmjenično za analizu i rješavanje fizikalnih problema.

Newtonova mehanika, temeljena na gore navedenim Newtonovim zakonima gibanja, opisuje kretanje tijela u sustavu povezujući sile koje djeluju na tijelo s promjenom količine gibanja tijela. Newtonovi zakoni su intuitivni i jednostavni za primjenu u mnogim slučajevima, posebno za sustave s malim brojem čestica.

S druge strane, Lagrangeova mehanika, formulirana od strane Josepha-Louisa Lagrangea, koristi drugačiji pristup. Umjesto fokusiranja na sile, koristi se količina koja se naziva lagranžijan, označena s $\mathcal{L}$, koja je definirana kao razlika između kinetičke energije T i potencijalne energije V sustava:

$$\mathcal{L} = T - V \tag{2.3}$$

Lagranžijan je funkcija generaliziranih koordinata (q) i njihovih derivacija prema vremenu (dq/dt). Osnovno načelo Lagrangeove mehanike je princip najmanje akcije, poznatiji i kao Hamiltonov princip. On kaže da je put kojim sustav prolazi između dvije točke u vremenu onaj za koji je akcija minimalna. Akcija, označena sa S, definirana je kao integral lagranžijana po vremenu:

$$S = \int \mathcal{L} dt \tag{2.4}$$

Primjenom principa najmanje akcije na Lagranžijan, mogu se izvesti Euler-Lagrangeove jednadžbe, odnosno jednadžbe gibanja za sustav.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = 0 \tag{2.5}$$

U gornjoj jednadžbi, q predstavlja generaliziranu koordinatu sustava koja se matematički može prikazati kao:

$$q = [q_1, q_2, \ldots, q_N] \tag{2.6}$$

Analogno ovome, $\dot{q}$ predstavlja vremensku derivaciju generalizirane koordinate q i matematički se može prikazati kao:

$$\dot{q} = [\dot{q_1}, \dot{q_2}, \ldots, \dot{q_N}] \tag{2.7}$$

Ove generalizirane koordinate i njihove derivacije u odnosu na vrijeme od ključne su važnosti u Lagrangeovoj mehanici jer omogućuju sveobuhvatniju i sustavniju analizu dinamike kompleksnih sustava. Upotrebom generaliziranih koordinata, Lagrangeove jednadžbe drugog reda mogu opisati kretanje sustava u terminima njegovih nezavisnih koordinata i odgovarajućih derivacija u odnosu na vrijeme, čime postaju moćno sredstvo za analizu različitih mehaničkih sustava. Euler-Lagrangeove jednadžbe opisuju razvoj generaliziranih koordinata tijekom vremena i ekvivalentne su Newtonovim jednadžbama gibanja. Ekvivalentnost između Lagrangeove mehanike i Newtonove mehanike vidljiva je u činjenici da Euler-Lagrangeove jednadžbe, izvedene iz principa najmanje akcije, dovode do istih jednadžbi gibanja kao Newtonovi zakoni. Obe formulacije opisuju istu fizičku stvarnost i pružaju ekvivalentne načine razumijevanja i rješavanja mehaničkih problema. Odabir formulacije često ovisi o složenosti i prirodi problema, s obzirom da svaka ima svoje prednosti i matematičku praktičnost u različitim situacijama.

2.3 Dokaz ekvivalentnosti između Lagrangeove mehanike i Newtonove mehanike

Fizikalni sustav može biti prilično složena kolekcija čestica ili tijela, no za potrebe našeg dokaza koristiti ćemo Lagrangeovu formulaciju za jedno tijelo mase m koje se giba u jednoj dimenziji pod utjecajem gravitacije. Neka x označava smjer kretanja (koji je vertikalan) tada su nam izrazi za kinetičku i potencijalnu energiju sljedeći:

$$U(x) = mgx \tag{2.8}$$
$$T(x) = \frac{1}{2}mv_x^2 \tag{2.9}$$

U gornjoj jednadžbi za potencijalnu energiju g označava gravitacijsku konstantu. Iz ovih jednadžbi dalje slijedi da je Lagranžijan sustava:

$$\mathcal{L}(x, v_x) = \frac{1}{2}mv_x^2 - mgx \tag{2.10}$$

Nultu razinu potencijalne energije izabiremo kada je $x = 0$. Nadalje, trebamo napisati Euler-Lagrangeovu jednadžbu za dani sustav pa su nam potrebne sljedeće derivacije:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{2}mv_x^2 - mgx\right) = -mg \tag{2.11}$$
$$\frac{\partial L}{\partial v_x} = \frac{\partial}{\partial v_x}\left(\frac{1}{2}mv_x^2 - mgx\right) = mv_x \tag{2.12}$$
$$\frac{d}{dt}\left(\frac{\partial L}{\partial v_x}\right) = \frac{d}{dt}\left(\frac{1}{2}mv_x^2 - mgx\right) = ma_x \tag{2.13}$$

Pišemo Euler-Lagrangeovu jednadžbu i u nju uvrštavamo gore izračunate derivacije te riješimo za a_x:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial v_x}\right) - \frac{\partial \mathcal{L}}{\partial x} = 0 \tag{2.14}$$
$$(ma_x) - (-mg) = 0 \tag{2.15}$$
$$ma_x = -mg \tag{2.16}$$
$$a_x = -g \tag{2.17}$$

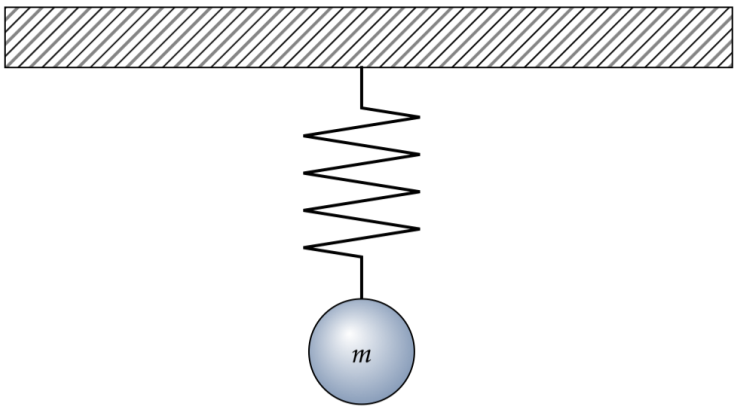
Ovo je identičan rezultat koji bismo dobili da smo problem rješavali preko Newtonovih zakona, dapače jednadžba (2.16) jest drugi Newtonov zakon.

2.4 Harmoničko gibanje

Jednostavno harmoničko gibanje je fundamentalni koncept u klasičnoj mehanici koji se koristi za opisivanje periodičnih oscilacija raznih fizičkih sustava. Ovo gibanje

može se primijeniti na različite fenomene u prirodi, uključujući gibanje masa na oprugama, titranje atoma u kristalima, elektromagnetske valove i još mnogo toga. U ovom poglavlju, pružit ćemo pregled osnova jednostavnog harmoničkog gibanja i matematičkih modela koji ga opisuju. Jednostavno harmoničko gibanje je vrsta periodičnog gibanja u kojem se tijelo ponaša tako da oscilira oko svog ravnotežnog položaja. U idealnim uvjetima, nema unutarnjih gubitaka energije, što znači da sustav nema prigušenja. Jednostavno harmoničko gibanje karakterizira osciliranje oko ravnotežnog položaja. Ravnotežni položaj je položaj u kojoj se tijelo nalazi kada nema vanjskih sila koje djeluju na njega. Ako se tijelo pomakne od ravnotežnog položaja javlja se povratna sila koja ga vraća prema ravnotežnom položaju. Glavne karakteristike jednostavnog harmoničkog gibanja su postojanje povratne sile koja je usmjerena ka ravnotežnom položaju i sinusni oblik oscilacija.

Kako bi uveli matematički model harmoničkog gibanja potreban nam je primjer tog gibanja. Stoga uzimamo masu obješenu na bezmasenu oprugu sa stropa. Strop je krut i neće imati utjecaja na dinamiku sustava. Povratna sila u našem sustavu biti



Slika 2.1: Masa m na opruzi obješena sa krutog stropa

će elastična sila opruge koja je opisana Hookeovim zakonom:

$$\vec{F}_{el} = -k\vec{x} \tag{2.18}$$

U gornjoj jednadžbi, k predstavlja konstantu opruge, $\vec{x}$ predstavlja pomak iz ravnotežnog položaja, a minus nam govori da je smjer sile suprotan smjeru pomaka. Matematički model jednostavnog harmoničkog gibanja može se izraziti kao diferencijalna jednadžba drugog reda:

$$m\frac{d^2x}{dt^2} + kx = 0 \tag{2.19}$$

gdje je m masa tijela koje oscilira, x je pomak od ravnotežnog položaja, t je vrijeme, a k je konstanta opruge. Rješenje ove jednadžbe daje jednadžbu koja opisuje jednostavno harmoničko gibanje:

$$x(t) = A\cos(\omega t + \phi) \tag{2.20}$$

gdje je A amplituda oscilacija, ϕ predstavlja pomak u fazi, a ω predstavlja kružnu frekvenciju koja se može izraziti kao:

$$\omega = \sqrt{\frac{k}{m}} \tag{2.21}$$

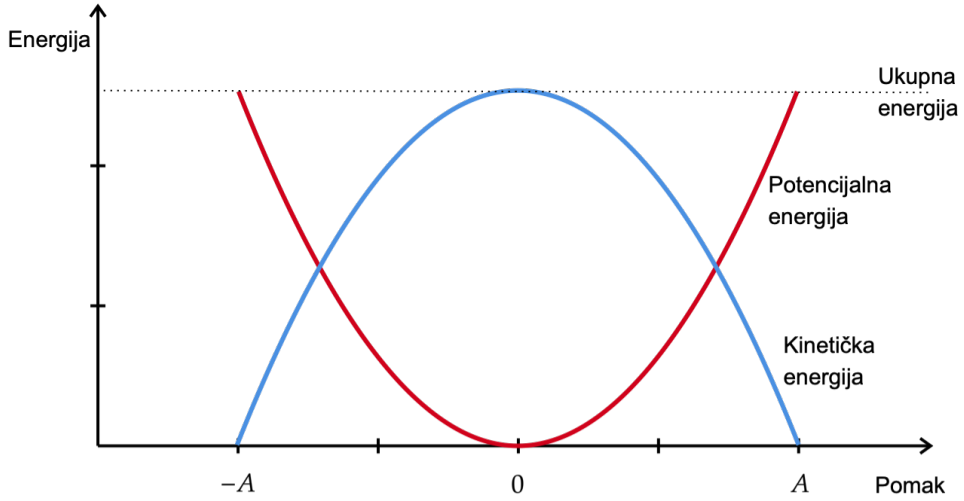
Harmoničko gibanje podrazumijeva periodičku prirodu gibanja, dakle ciklus oscilacija ponavlja se nakon određenog vremenskog razdoblja. Taj vremenski interval nazivamo periodom i označavamo ga s T, a matematički ga možemo izraziti kao:

$$T = \frac{2\pi}{\omega} = 2\pi\sqrt{\frac{m}{k}} \tag{2.22}$$

Druga veličina vezana uz periodičku prirodu harmoničnog gibanja je frekvencija koju označavamo kao f i koja predstavlja broj ciklusa oscilacije u jedinici vremena. Matematički frekvenciju izražavamo kao recipročnu vrijednost perioda:

$$f = \frac{1}{T} = \frac{\omega}{2\pi} = \frac{\sqrt{k}}{2\pi\sqrt{m}} \tag{2.23}$$

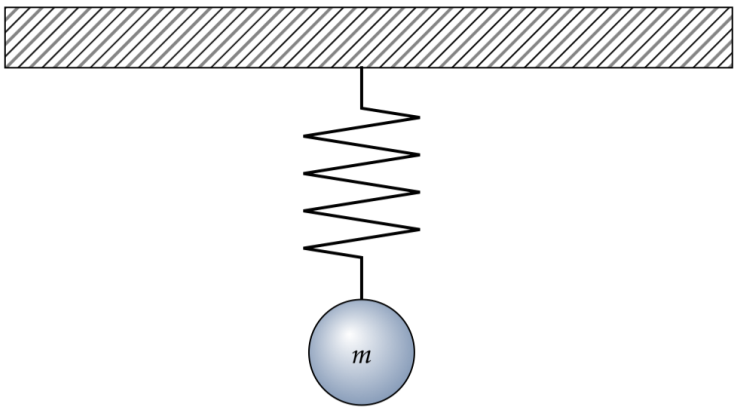
Harmoničko gibanje možemo razmatrati i s energijskog aspekta. Tada možemo zamijetiti da se harmoničko gibanje održava putem izmjene potencijalne i kinetičke energije. Na početku, kada je tijelo u amplitudnom položaju, potencijalna energija je maksimalna, a kinetička energija jednaka je nuli. Tijekom gibanja, kada se tijelo pomiče od amplitudnog položaja prema ravnotežnom položaju, potencijalna se energija smanjuje, a kinetička energija raste. U ravnotežnom položaju, potencijalna energija jednaka je nuli, a kinetička je energija maksimalna. Kako se tijelo vraća prema amplitudnom položaju, kinetička se energija smanjuje, dok istovremeno potencijalna energija raste. Ovu interakciju energija možemo vidjeti na slici (2.2).



Slika 2.2: Graf kinetičke i potencijalne energije harmoničkog oscilatora u ovisnosti o pomaku iz ravnotežnog položaja

2.5 Dinamika mase na opruzi

U ovom potpoglavlju riješiti ćemo dinamiku sustava mase na opruzi. Koristiti ćemo gore opisan lagražijanski pristup problemu. Prvo je potrebno definirati sustav. Naš sustav biti će masa na opruzi obješena sa stropa, kao na slici (2.3). Prvi korak je



Slika 2.3: Masa m na opruzi obješena sa krutog stropa

odrediti potencijalnu i kinetičku energiju mase:

$$T = \frac{1}{2}m\dot{x}^2 \tag{2.24}$$

$$U = \frac{1}{2}kx^2 - mgx \tag{2.25}$$

Iz ovih energija slijedi lagranžijan:

$$\begin{aligned}
\mathcal{L} &= T - U && (2.26)\\
&= \frac{1}{2}m\dot{x}^2 - \left(\frac{1}{2}kx^2 - mgx\right) && (2.27)\\
&= \frac{1}{2}m\dot{x}^2 - \frac{1}{2}kx^2 + mgx && (2.28)
\end{aligned}$$

Kako bi napisali Euler-Lagrangeovu jednadžbu, potrebne su nam sljedeće derivacije:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x} &= \frac{\partial}{\partial x}\left(\frac{1}{2}m\dot{x}^2 - \frac{1}{2}kx^2 + mgx\right) && (2.29)\\
&= -kx + mg && (2.30)\\
\frac{\partial \mathcal{L}}{\partial \dot{x}} &= \frac{\partial}{\partial \dot{x}}\left(\frac{1}{2}m\dot{x}^2 - \frac{1}{2}kx^2 + mgx\right) && (2.31)\\
&= m\dot{x} && (2.32)\\
\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) &= \frac{d}{dt}\Big(m\dot{x}\Big) && (2.33)\\
&= m\ddot{x} && (2.34)
\end{aligned}$$

Sada možemo napisati Euler-Lagrangeovu jednadžbu i u nju uvrstiti gore izračunate derivacije te ju transformirati u standardni oblik:

$$\begin{aligned}
\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) - \frac{\partial \mathcal{L}}{\partial x} &= 0 && (2.35)\\
m\ddot{x} + kx - mg &= 0 && (2.36)
\end{aligned}$$

Ishodište našeg trenutnog koordinatnog sustava $y = 0$ je u razini neopterećene opruge, što znači da je ravnotežni položaj sa mirnom masom na opruzi u $y = y_0$. Stoga, član $-mg$ možemo zamijeniti na sljedeći način:

$$-mg = -kx_0 \quad (2.37)$$

Nadalje, uvrštavanjem (2.37) u (2.36) i daljnjim sređivanjem izraza dobivamo:

$$\begin{aligned}
m\ddot{x} + kx - kx_0 &= 0 && (2.38)\\
m\ddot{x} + k(x - x_0) &= 0 && (2.39)\\
\ddot{x} + \frac{k}{m}(x - x_0) &= 0 && (2.40)
\end{aligned}$$

Sada možemo definirati novu varijablu:

$$\mathcal{X} = x - x_0 \quad (2.41)$$

Ovo je isto kao da smo definirali novu os $\mathcal{X}$ koja je pomaknuta prema dolje za x_0. Drugim riječima, to je isto kao da definiramo novu os $\mathcal{X}$ čiji je početak u ravnotežnom položaju, umjesto na poziciji gdje je sama opruga u stanju mirovanja. Moramo napomenuti i da su druge derivacije x i $\mathcal{X}$ jednake:

$$\frac{d^2x}{dt^2} = \frac{d^2}{dt^2}(\mathcal{X} + x_0) = \frac{d^2\mathcal{X}}{dt^2} \tag{2.42}$$

Sada možemo jednadžbu (2.36) zapisati kao:

$$\ddot{\mathcal{X}} + \frac{k}{m}\mathcal{X} = 0 \tag{2.43}$$

U gornjoj jednadžbi možemo prepoznati faktor k/m kao ω^2 te uvrštavanjem dobivamo standardnu jednadžbu gibanja jednostavnog harmoničnog oscilatora:

$$\ddot{\mathcal{X}} + \omega^2\mathcal{X} = 0 \tag{2.44}$$

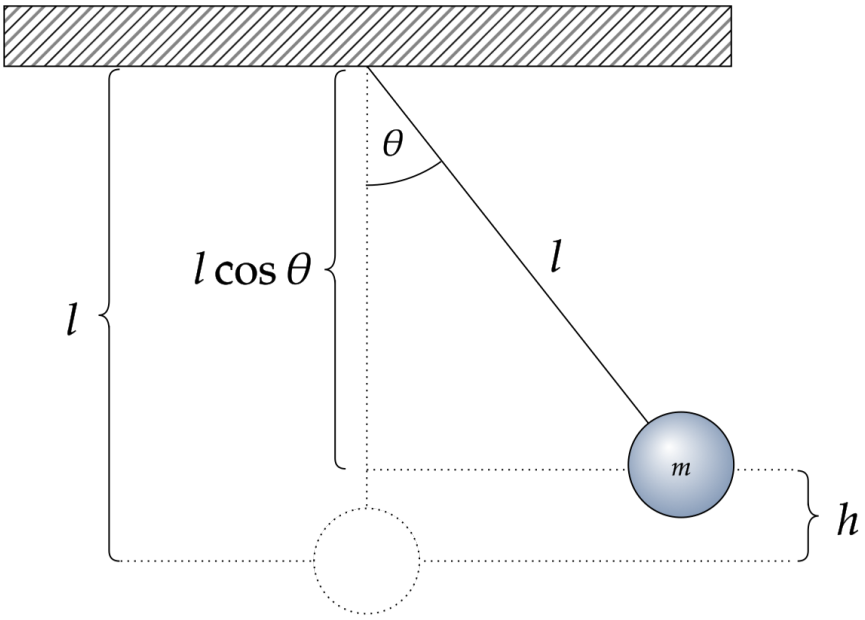
Ako je u sustavu prisutno trenje to mijenja oblik standardne jednadžbe gibanja ovog harmoničkog oscilatora u:

$$m\ddot{x} + \gamma\dot{x} + kx = 0 \tag{2.45}$$

U gornjoj jednadžbi γ je koeficijent prigušenja koji kvantificira dodatnu silu koja djeluje na masu. Sila prigušenja proporcionalna je brzini mase te joj se suprotstavlja. Ova jednadžba opisuje kretanje mase koja je izložena sili obnove od opruge i sili prigušenja ($\gamma\dot{x}$). Sila prigušenja suprotstavlja se kretanju. Prisutnost prigušenja u sustavu rezultira time da se kretanje objekta s vremenom zaustavlja, za razliku od idealnog slučaja ($\gamma = 0$), gdje bi oscilacije trajale neograničeno. Opće rješenje ove jednadžbe prigušenog harmoničkog oscilatora uključuje eksponencijalne članove i trigonometrijske funkcije. Konkretni oblik rješenja varira ovisno o početnim uvjetima i režimu prigušenja.

2.6 Dinamika matematičkog njihala

U ovom potpoglavlju riješiti ćemo dinamiku sustava koji se sastoji od matematičkog njihala i Zemlje. Matematičko njihalo jest konceptualni model u fizici. To je idealizirani sustav bez trenja i unutarnjih gubitaka energije koji se sastoji od točkaste mase obješene o bezmasenu i nerastezljivu nit. Masa se može slobodno gibati u jednoj ravnini. Opet ćemo koristiti gore opisan lagranžijanski pristup problemu. Za početak, potrebna je skica sustava sa svim unaprijed potrebnim oznakama, što možemo vidjeti na slici (2.4). Kako bismo postavili lagranžijan, prvo su nam potrebni izrazi za kinetičku i potencijalnu energiju sustava. S obzirom da u sustavu imamo masu koja se



Slika 2.4: Matematičko njihalo mase m i niti duljine l

giba po kružnoj putanji, kinetičku energiju možemo izraziti preko momenta inercije I:

$$T = I\omega^2 \tag{2.46}$$

Ovaj izraz želimo preoblikovati tako da ovisi o generaliziranoj koordinati koja će u ovom slučaju biti θ, stoga ćemo upotrijebiti definiciju kružne frekvencije:

$$\omega = \frac{d\theta}{dt} \tag{2.47}$$

$$\omega = \dot{\theta} \tag{2.48}$$

Nadalje, uvrstit ćemo izraz (2.48) u izraz (2.46), kako bi dobili kinetičku energiju kao:

$$T = I\dot{\theta}^2 \tag{2.49}$$

Trebamo definirati i izraz za potencijalnu energiju sustava. S obzirom da jedini doprinos potencijalnoj energiji dolazi od strane gravitacije, možemo napisati:

$$U = mgh \tag{2.50}$$

Nadalje, potrebno je definirati h u terminima l i θ, što gledajući sliku (2.4) možemo zapisati kao:

$$h = l - l\cos\theta \tag{2.51}$$

Uvrštavanjem jednadžbe (2.51) u jednadžbu (2.50) dobivamo izraz za potencijalnu energiju sustava:

$$\begin{aligned} U &= mg(l - l\cos\theta) && (2.52)\\ &= mgl(1 - \cos\theta) && (2.53) \end{aligned}$$

Iz ovoga možemo napisati lagranžijan sustava kao:

$$\begin{aligned} \mathcal{L} &= T - U && (2.54)\\ &= I\dot{\theta}^2 - mgl(1 - \cos\theta) && (2.55)\\ &= I\dot{\theta}^2 - mgl + mgl\cos\theta && (2.56) \end{aligned}$$

Nadalje, potrebno je napisati Euler-Lagrangeovu jednadžbu, a za to su nam potrebne sljedeće derivacije:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} &= \frac{\partial}{\partial \theta}\left(I\dot{\theta}^2 - mgl + mgl\cos\theta\right) && (2.57)\\ &= -mgl\sin\theta && (2.58)\\ \frac{\partial \mathcal{L}}{\partial \dot{\theta}} &= \frac{\partial}{\partial \dot{\theta}}\left(I\dot{\theta}^2 - mgl + mgl\cos\theta\right) && (2.59)\\ &= 2I\dot{\theta} && (2.60)\\ &= ml^2\dot{\theta} && (2.61)\\ \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) &= \frac{d}{dt}\left(ml^2\dot{\theta}\right) && (2.62)\\ &= ml^2\ddot{\theta} && (2.63) \end{aligned}$$

S dobivenim derivacijama možemo napisati Euler-Lagrangeovu jednadžbu:

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) - \frac{\partial \mathcal{L}}{\partial x} &= 0 && (2.64)\\ ml^2\ddot{\theta} - (-mgl\sin\theta) &= 0 && (2.65)\\ l\ddot{\theta} + g\sin\theta &= 0 && (2.66) \end{aligned}$$

Kako bi iz jednadžbe (2.66) došli do standardnog oblika jednadžbe gibanja jednostavnog harmoničkog oscilatora, moramo upotrijebiti aproksimaciju malih kutova za funkciju sinus.

Aproksimacija malih kutova za funkciju sinus je matematička tehnika koja se koristi za pojednostavljivanje trigonometrijskih izraza, u slučaju kada su kutovi jako mali. Ova aproksimacija vrlo je korisna u mnogim područjima matematike i fizike jer omogućuje jednostavnije i brže rješavanje problema. Aproksimacija malih kutova za funkciju sinus temelji se na Taylorovom razvoju funkcije oko točke $\theta = 0$. Taylorov razvoj funkcije sinus izgleda ovako:

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \dots \quad (2.67)$$

Kada je kut θ jako mali, možemo zanemariti sve članove koji sadrže θ^n za $n \geq 3$ jer su ti članovi zanemarivo mali. Ostaje samo prvi član θ u Taylorovom razvoju, što rezultira aproksimacijom malih kutova za sinus:

$$\sin\theta \approx \theta \tag{2.68}$$

Ova aproksimacija vrijedi samo za male vrijednosti kuta θ, koje su izražene u radijanima. Što je kut bliži nuli, to je aproksimacija točnija. Za veće kutove, ovaj pristup nije prikladan te moramo koristiti sinus.

Ako sada upotrijebimo aproksimaciju (2.68) u jednadžbi (2.66), dobivamo sljedeći izraz:

$$l\ddot{\theta} + g\theta = 0 \tag{2.69}$$

$$\ddot{\theta} + \frac{g}{l}\theta = 0 \tag{2.70}$$

U gornjem izrazu možemo prepoznati faktor g/l kao ω^2 te sada dobivamo standardnu jednadžbu gibanja jednostavnog harmoničkog oscilatora:

$$\ddot{\theta}(t) + \omega^2\theta(t) = 0 \tag{2.71}$$

3 Pregled osnova strojnog učenja

3.1 Uvod u strojno učenje

Strojno učenje znanstvena je disciplina koja se bavi razvojem algoritama i tehnika, sa svrhom razvijanja sposobnosti računalnih sustava za učenje iz iskustava i podataka. Nadalje, ta se znanja primjenjuju na novim problemima i situacijama. Glavni cilj strojnog učenja je omogućiti računalima da se što bolje generaliziraju, tj. da budu uspješni u redakcijama iz novo dobivenih podataka, bez potrebe za eksplicitnim programiranjem svakog koraka. To se postiže kroz različite metode i modele koji omogućuju računalima da identificiraju obrasce i donose odluke na temelju velikih količine podataka. Umjetna inteligencija, uključujući strojno učenje, nastoji simulirati ljudsko učenje i sposobnost donošenja odluka, no umjesto da programiramo svaki korak i pravilo u računalni program, strojno učenje omogućuje računalu da samostalno "uči iz iskustva". Kroz proces učenja, računalo koristi različite modele i algoritme kako bi prepoznalo obrasce i strukture u podacima. Nakon što je učenje modela izvršeno na velikoj količini podataka, može se primijeniti na novim podacima kako bi se donijeli zaključci ili predviđanja za nove podatke. Strojno učenje ima široku primjenu u mnogim područjima, kao što su primjerice fizika, medicina, financije, robotika, prepoznavanje uzoraka, analiza podataka i predviđanje trendova. Primjenom strojnog učenja, računalni sustavi mogu rješavati složene probleme koji bi inače bili teško ili nemoguće rješivi klasičnim pristupom programiranju. Važno je napomenuti da kvaliteta i pouzdanost strojnog učenja ovise o kvaliteti i količini podataka na kojima se učenje modela vrši. Velika količina raznolikih i kvalitetnih podataka omogućuje bolje generaliziranje i točnije predviđanje. Stoga, prikupljanje, obrada i priprema podataka igraju ključnu ulogu u uspješnoj primjeni strojnog učenja. Nadzirano, nenadzirano i polu-nadzirano učenje tri su osnovne paradigme strojnog učenja, a koriste se za rješavanje različitih problema analize podataka i formiranja predviđanja na temelju istih.

Nadzirano učenje jedna je od glavnih paradigmi strojnog učenja koja se koristi za rješavanje problema predviđanja i klasifikacije. Ova paradigma temelji se na ideji da računalni sustav uči iz podataka koji su označeni. Ti podaci već su obrađeni, a izlazne vrijednosti koje želimo dobiti iz njih jasno su definirane i dostupne računalu. Na temelju ovih označenih podataka, računalni model pokušava naučiti funkcijsku vezu između ulaznih i odgovarajućih izlaznih vrijednosti, kako bi mogao donositi precizne predikcije na novim, neoznačenim, ulaznim podacima. Postupak nadziranog učenja obično se događa u sljedećim koracima:

1. Skup podataka: Prvi korak u nadziranom učenju je prikupljanje skupa podataka koji će se koristiti za obuku modela. Skup podataka sastoji se od parova ulaznih i odgovarajućih izlaznih vrijednosti. Na primjer, ako želimo naučiti model

koji predviđa cijene nekretnina, skup podataka treba sadržavati informacije o različitim nekretninama (ulazni podaci) i njihovim stvarnim cijenama (izlazne vrijednosti).

2. Podjela podataka: Skup podataka obično se dijeli na dva dijela - skup za učenje i skup za vrednovanje. Skup za učenje koristi se za učenje modela, dok se skup za vrednovanje koristi za evaluaciju valjanosti modela na novim podacima koje model nije vidio tijekom učenja.

3. Odabir modela: Sljedeći korak je odabir odgovarajućeg modela za rješavanje specifičnog problema. Za model možemo koristiti neuronsku mrežu, linearni model, stablo odlučivanja, potporne vektore ili neki drugi algoritam strojnog učenja. Odabir modela ovisi o prirodi problema i vrsti podataka koji su dostupni.

4. Učenje modela: Nakon što je model odabran, koristi se skup za učenje, kako bi model naučio funkcijsku ovisnost između ulaznih i izlaznih vrijednosti. Model koristi različite algoritme optimizacije kako bi minimalizirao grešku između predviđenih izlaznih vrijednosti i stvarnih izlaznih vrijednosti.

5. Vrednovanje modela: Nakon učenja, model se vrednuje na skupu za vrednovanje, sa svrhom procjene valjanosti na novim podacima. Koriste se različite metrike evaluacije kako bi se utvrdila točnost i pouzdanost modela.

6. Uporaba modela: Nakon što je model uspješno naučen i vrednovan, može se koristiti za predviđanje izlaznih vrijednosti na novim, neviđenim ulaznim podacima. Model se može primijeniti na stvarne probleme i koristiti za donošenje informiranih odluka na temelju novih podataka.

Nadzirano učenje ima širok spektar primjena, uključujući predviđanje cijena, klasifikaciju slika, prepoznavanje govora (*NLP*), klasifikaciju e-mailova kao neželjenih (eng. *spam*) na temelju sadržaja e-maila, prepoznavanje rukom pisanih slova i pretvaranje istih u tekst, te mnoge druge zadatke koji zahtijevaju predviđanju na temelju postojećih podataka. Ova paradigma ima ključnu ulogu u razvoju inteligentnih sustava koji mogu automatski učiti i poboljšavati svoje performanse kako bi rješavali složene probleme u stvarnom svijetu.

Nenadzirano učenje druga je glavna paradigma strojnog učenja, a koristi se za otkrivanje skrivenih struktura i obrazaca u podacima. Za razliku od nadziranog učenja, u ne nadziranom učenju podaci nisu označeni odgovarajućim izlaznim vrijednostima. Cilj ne nadziranog učenja je identificirati inherentne strukture, grupiranja ili sličnosti unutar skupa podataka. Postupak ne nadziranog učenja događa se u sljedećim koracima:

1. Skup podataka: Kao i u nadziranom učenju, prvi korak je prikupljanje skupa podataka. Međutim, u ne nadziranom učenju, ovaj skup podataka sastoji se samo od ne označenih ulaznih podataka, bez izlaznih vrijednosti. Na primjer, ako imamo skup slika lica, nenadzirano učenje može pomoći u identifikaciji sličnosti ili grupiranju slika lica na temelju karakteristika koje nisu eksplicitno označene.

2. Odabir modela: Nakon što je skup podataka prikupljen, slijedeći korak je odabir odgovarajućeg modela koji će otkriti skriveni obrazac u podacima. Modeli koji se često koriste u nenadziranom učenju uključuju klastere, autoenkodere, PCA (eng. *Principal Component Analysis*), i druge algoritme koji pomažu u identifikaciji struktura u podacima.

3. Učenje modela: Učenje modela u nenadziranom učenju razlikuje se od nadziranog učenja. Ovdje nema izlaznih vrijednosti koje model treba predvidjeti. Umjesto toga, model koristi samo ulazne podatke kako bi naučio strukture i obrasce unutar podataka. Na primjer, u algoritmu klasteriranja, model grupira slične podatke zajedno bez ikakvih prethodnih informacija o tome koje su skupine slične.

4. Vrednovanje modela: U nenadziranom učenju, evaluacija dobrote modela nešto je složenija nego u nadziranom učenju. Metrike evaluacije ovise o vrsti problema koji se rješava i ciljevima analize koja se provodi. U klasterskoj analizi, na primjer, možemo koristiti metrike kao što su *Silhouette Coefficient* ili *Adjusted Rand Index* [1], kako bismo procijenili kvalitetu grupiranja.

5. Interpretacija rezultata: Nakon što je model obučen i vrednovan, slijedi interpretacija rezultata. Cilj je razumjeti koje su strukture i oblici pronađeni u podacima kako bismo dobili uvid u skrivene karakteristike skupa podataka. Ovo je bitan korak jer postoji mogućnost da je računalni model pronašao neke značajke unutar skupa podataka po kojima ih može grupirati, no takvo grupiranje ne mora dati korisne ili točne izlazne podatke. Primjer ovakvog problema bio bi da računalnom modelu damo slike pasa i mačaka i pustimo ga da nauči razlikovati ih. Model je kao značajku mogao izabrati imaju li životinje na slici ogrlicu ili ne, te na temelju toga grupirati ulazne slike. Ovo bi, naravno, rezultiralo netočnim izlaznim podacima, s obzirom da i psi i mačke mogu imati, odnosno ne imati ogrlicu.

Nenadzirano učenje ima široku primjenu u analizi podataka, istraživanju skrivenih zakonitosti, segmentaciji i grupiranju podataka, redukciji dimenzionalnosti problema i drugim područjima gdje nemamo točna očekivanja za izlazne vrijednosti. Ova paradigma dopušta nam da pomoću nje istražujemo masivne skupove podataka i u njima otkrivamo nove zakonitosti, što bi inače bilo nemoguće.

Polunadzirano učenje je paradigma strojnog učenja koja kombinira elemente nadziranog i nenadziranog učenja. U ovom pristupu, skup podataka sadrži neke označene izlazne vrijednosti, ali ne sve. To znači da samo dio podataka ima pridružene točne izlazne vrijednosti, dok su ostali podaci neoznačeni. Glavni cilj polunadziranog učenja je koristiti dostupne označene podatke za izradu i poboljšanje kvalitete modela te učenje struktura, uz pomoć neoznačenih podataka. Ovo može biti korisno u slučajevima kada je označavanje podataka skupo, vremenski zahtjevno ili nije moguće dobiti oznake za sve ulazne podatke. Postupak polunadziranog učenja obično uključuje sljedeće korake:

1. Skup podataka: Skup podataka sastoji se od ulaznih podataka i odgovarajućih izlaznih vrijednosti, ali samo za dio podataka. Ostali podaci nemaju pridružene izlazne vrijednosti.

2. Odabir modela: Odabire se model koji može koristiti dostupne označene podatke za učenje i poboljšanje modela, ali istovremeno može koristiti i neoznačene podatke za otkrivanje dodatnih obrazaca i struktura. Ovo obično uključuje modele koji su fleksibilni i mogu se prilagoditi različitim tipovima podataka.

3. Učenje modela: Učenje modela u polunadziranom učenju obuhvaća korištenje dostupnih označenih podataka za poboljšanje modela. Osim toga, model koristi i neoznačene podatke kako bi otkrio skrivene strukture i grupe u podacima.

4. Vrednovanje modela: Evaluacija valjanosti modela u polunadziranom učenju zahtijeva uzimanje u obzir kako označenih, tako i neoznačenih podataka. Metrike evaluacije mogu biti prilagođene ovisno o količini dostupnih oznaka i ciljevima analize.

5. Interpretacija rezultata: Kako bismo dobili uvid u dobivene rezultate, interpretacija modela i otkrivenih obrazaca važna je faza. Cilj je razumjeti kako model koristi označene i neoznačene podatke.

Polunadzirano učenje ima široku primjenu u područjima gdje su označeni podaci ograničeni, ali su dostupni mnogi neoznačeni podaci. Primjeri primjene polunadziranog učenja uključuju obradu prirodnog jezika (*NLP*), analizu slika, detekciju anomalija, i druge zadatke gdje ne možemo lako dobiti oznake za sve podatke.

3.2 Osnovni koncepti strojnog učenja

3.2.1 Funkcija gubitka

Kako bi strojno učenje bilo učinkovito, bitan korak koji se javlja u svim gore navedenim oblicima jest evaluacija modela. U tom koraku računalni model dobiva povratnu

informaciju glede svoje izlazne vrijednosti. Alat koji koristimo za povratnu informaciju zove se funkcija gubitka (eng. *loss function*). Funkcija gubitka predstavlja matematičku mjeru odstupanja predviđanja modela od stvarnih vrijednosti u skupu podataka. Svrha za koju koristimo funkciju gubitka je minimalizacija te razlike kako bi se model što bolje prilagodio danom zadatku. Funkcija gubitka ima ključnu ulogu u uspješnom učenju modela jer omogućuje kvantificiranje odstupanja izlazne vrijednosti računalnog modela od očekivane ili stvarne vrijednosti. Kako bismo osigurali korištenje najbolje moguće funkcije gubitka, prvo moramo kategorizirati problem koji rješavamo. Problemi koje rješavamo strojnim učenjem mogu se svrstati u dvije velike kategorije, a to su regresijski problemi i kategorizacijski problemi. Regresijski problemi odnose se na predviđanje kontinuiranih izlaznih vrijednosti na temelju ulaznih podataka. Na primjer, možemo koristiti regresijski model kako bismo predvidjeli cijenu nekretnine na temelju njezinih karakteristika, kao što su kvadratura, broj soba, lokacija itd. Klasifikacijski problemi odnose se na predviđanje diskretnih izlaznih vrijednosti, odnosno pripadnost određenoj klasi na temelju ulaznih podataka. Na primjer, možemo koristiti klasifikacijski model kako bismo odredili je li na danoj slici prikazan pas ili mačka. Najčešća funkcija gubitka koja se koristi za regresijske probleme jest srednja kvadratna pogreška (eng. *Mean Squared Error - MSE*), dok se kod klasifikacijskih problema, ukoliko podatci imaju više klasa (grupa) unutar sebe, najčešće koristi kategorizacijska križna entropija (eng. *Categorical Cross- Entropy*), ili binarna križna entropija (eng. *Binary Cross-Entropy*), kada podatci imaju samo dvije klase.

Srednja kvadratna pogreška (MSE) kao funkcija gubitka ima ključnu ulogu u procjeni kvalitete modela kod regresijskih problema strojnog učenja. Njena primjena široko je rasprostranjena jer omogućuje kvantifikaciju preciznosti modela usporedbom stvarnih i predviđenih izlaznih vrijednosti. Ideja iza funkcije gubitka MSE je jednostavna: želimo minimalizirati kvadrat razlike između predviđenih i stvarnih izlaznih vrijednosti. Ako je razlika između tih vrijednosti mala, to znači da model dobro aproksimira stvarne podatke. S druge strane, velika razlika ukazuje na lošu predikciju i potrebu za poboljšanjem modela. MSE se matematički definira kao srednja vrijednost kvadrata razlika između stvarnih izlaznih vrijednosti i predviđenih izlaznih vrijednosti za sve primjere u skupu podataka. Neka je n broj primjera u skupu podataka, y_i stvarna izlazna vrijednost za i-ti primjer, a $\hat{y}_i$ predviđena izlazna vrijednost od strane našeg modela za isti primjer. Tada je kvadratna pogreška za i-ti primjer definirana kao:

$$E_i = (y_i - \hat{y}_i)^2 \tag{3.1}$$

Srednja kvadratna pogreška (MSE) za cijeli skup podataka izračunava se kao srednja

vrijednost svih kvadratnih pogrešaka:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} E_i \tag{3.2}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.3}$$

Cilj je minimalizirati ovu funkciju gubitka kako bismo pronašli optimalne parametre modela koji će dovesti do što preciznijih predikcija. Za postizanje ovog cilja koristimo različite optimizacijske algoritme, pri čemu je jedan od najpoznatijih i najčešće korištenih gradijentni spust. Gradijentni spust je iterativni algoritam koji prilagođava parametre modela u smjeru negativnog gradijenta funkcije gubitka, kako bi se smanjila pogreška. Drugim riječima, koristi se matematički račun za pronalaženje smjera u kojem će funkcija gubitka opadati najbrže, te ažurira parametre modela u tom smjeru. Postupak gradijentnog spusta ponavlja se kroz više iteracija sve dok se ne postigne zadovoljavajuća razina pogreške ili broj iteracija. Važno je napomenuti da izbor funkcije gubitka ima veliki utjecaj na performanse modela i uspješnost rješavanja problema. Za regresijske probleme, MSE je popularan izbor jer je osjetljiv na velika odstupanja, što je važno u predviđanju kontinuiranih izlaznih vrijednosti. Međutim, treba pažljivo odabrati funkciju gubitka - ovisno o prirodi problema, kako bi se postigla što bolja aproksimacija stvarnih podataka i poboljšale performanse modela.

Kategorizacijska križna entropija (eng. *Categorical Cross-Entropy*) još je jedna važna funkcija gubitka koja se često koristi u problemima klasifikacije s više klasa u strojnom učenju. Ova funkcija gubitka igra ključnu ulogu u procjeni kvalitete klasifikacijskoga modela, omogućujući kvantifikaciju razlike između stvarnih i predviđenih vjerojatnosti klasa. Prije nego što uđemo u detalje, važno je razumjeti osnovne pojmove u klasifikacijskim problemima. U klasifikaciji, cilj je dodijeliti svakom primjeru ulaznih podataka odgovarajuću klasu iz skupa mogućih klasa. Na primjer, možemo imati problem klasifikacije slika, gdje želimo prepoznati je li na slici prikazan bicikl, motocikl, ili neki drugi objekt. Ključni korak u klasifikacijskom modeliranju je stvaranje predikcije vjerojatnosti za svaku od mogućih klasa za svaki primjer u skupu podataka. Ove predviđene vjerojatnosti obično su dobivene pomoću neuronskih mreža, logističke regresije ili drugih klasifikacijskih metoda. Kada imamo više od dvije moguće klase, generalno se koristi kategorizacijska križna entropija kao funkcija gubitka. Kategorizacijska križna entropija izvodi se iz teorije informacija i mjeri količinu informacija koja je potrebna za opisivanje različitih vjerojatnosti u odnosu na stvarne vjerojatnosti klasa. Ako su stvarne i predviđene vjerojatnosti iste, križna entropija je minimalna, što znači da model dobro predviđa klasu za svaki dani primjer. S druge strane, veće odstupanje između stvarnih i predviđenih vjerojatnosti rezultira većom vrijednosti križne entropije, što ukazuje na slabije performanse modela. Neka je n broj primjera u skupu podataka, a p_i i $\hat{p}_i$ stvarne i predviđene vjerojatnosti klase

i, redom. Kategorizacijska križna entropija izračunava se prema sljedećoj formuli:

$$\text{CCE} \quad = \quad -\sum_{i=1}^{n} p_i \log(\hat{p}_i) \tag{3.4}$$

Uočimo da se sumacija provodi preko svih mogućih klasa, što omogućuje vrednovanje svake klase pojedinačno i procjenu ukupne pogreške klasifikacije. Kategorizacijska križna entropija široko je korištena jer osjetljivo reagira na razlike u vjerojatnostima i pruža detaljne informacije o performansama modela za svaku klasu. Ova funkcija gubitka igra ključnu ulogu u procesu učenja klasifikacijskih modela, kako bismo postigli što točnije predikcije i bolje razumjeli ponašanje modela u različitim klasama. Odabir kategorizacijske križne entropije kao funkcije gubitka ovisi o prirodi problema klasifikacije i vrsti podataka koju model obrađuje.

Binarna križna entropija još je jedna značajna funkcija gubitka koja se često koristi u binarnim klasifikacijskim problemima strojnog učenja. Ova funkcija gubitka ima ključnu ulogu u evaluaciji performansi binarnih klasifikacijskih modela, pomažući u kvantificiranju odstupanja izlaznih vrijednosti modela od stvarnih vjerojatnosti klase. Prije nego što detaljnije objasnimo binarnu križnu entropiju, važno je shvatiti što podrazumijeva binarna klasifikacija. U binarnoj klasifikaciji, model ima zadatak predviđati jednu od dvije moguće klase za svaki primjer ulaznih podataka. Na primjer, možemo imati problem binarne klasifikacije koji određuje je li pacijent zaražen određenom bolešću (pozitivna klasa) ili nije (negativna klasa) na temelju medicinskih atributa. Ključni aspekt binarne križne entropije je usredotočiti se na vjerojatnosti pripadnosti pozitivnoj klasi, koje označavamo s p_i, i predviđene vjerojatnosti pozitivne klase, koje označavamo s $\hat{p}_i$, za svaki primjer i u skupu podataka. Ovaj pristup temelji se na ideji da se, u binarnoj klasifikaciji, vjerojatnost negativne klase može izravno odrediti kao $1 - p_i$. Binarna križna entropija izračunava se prema sljedećoj formuli:

$$\text{BCE} \quad = \quad -\sum_{i=1}^{n} p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i) \tag{3.5}$$

U ovoj formuli, prvi dio $-p_i \log(\hat{p}_i)$ kvantificira odstupanje u vjerojatnosti pozitivne klase, dok drugi dio $(1 - p_i) \log(1 - \hat{p}_i)$ kvantificira odstupanje u vjerojatnosti negativne klase. Ovako strukturirana funkcija gubitka osigurava uravnoteženu evaluaciju predikcija za obe klase i "potiče" model da precizno procijeni vjerojatnosti za obje klase. Važno je napomenuti da binarna križna entropija ima veći utjecaj na model kada su vjerojatnosti predikcija bliske 0 ili 1, što je ključno u problemima s jasno definiranim granicama između klasa. Također, binarna križna entropija "potiče" model da se fokusira na relevantne značajke i donosi precizne odluke u binarnoj klasifikaciji. Odabir binarne križne entropije kao funkcije gubitka ovisi o specifičnostima binarnih klasifikacijskih problema i prirodi podataka koju model obrađuje. Pravilnim

odabirom ove funkcije gubitka možemo osigurati da naš binarni klasifikacijski model postiže visoke performanse u razlikovanju između dvije moguće klase, kao i da daje precizne i pouzdane predikcije u stvarnim situacijama.

3.2.2 Optimizacija u strojnom učenju

Optimizacija je ključan koncept u strojnom učenju koji se koristi za postizanje najboljih mogućih rezultata modela. Cilj optimizacije je pronaći najbolje parametre modela koji minimiziraju funkciju gubitka, odnosno maksimiziraju funkciju cilja [2]. Ovaj proces igra ključnu ulogu u učenju modela i poboljšavanju njegovih performansi. U strojnom učenju, modeli često imaju mnogo parametara koji se mogu prilagoditi kako bi se poboljšala valjanost njihovih predikcija. Međutim, određivanje optimalnih vrijednosti tih parametara može biti izuzetno složeno zbog visoke dimenzionalnosti prostora parametara i kompleksnosti funkcija gubitka. Upravo zato se koriste različite tehnike optimizacije koje pomažu u pronalasku najboljih parametara, uz što manji utrošak računalnih resursa i vremena.

Gradijentna metoda jedan je od osnovnih i najraširenijih algoritama optimizacije u strojnom učenju. Ova tehnika omogućava pronalazak lokalnih minimuma (ili maksimuma) funkcije gubitka, kako bi se pronašli optimalni parametri modela. Cilj gradijentne metode je prilagoditi parametre modela u smjeru negativnog gradijenta funkcije gubitka, kako bi se minimalizirala greška modela. Osnovni koncept gradijentne metode temelji se na funkciji gubitka $J(\vec{\theta})$, gdje $\vec{\theta}$ predstavlja vektor svih parametara modela koje želimo optimizirati. Gradijent te funkcije, označen sa $\vec{\nabla} J(\vec{\theta})$, predstavlja vektor parcijalnih derivacija funkcije gubitka u odnosu na svaki parametar θ_i i ažurirani vektor $\vec{\nabla}$ računamo na sljedeći način:

$$\vec{\theta}_{\text{novi}} \quad = \quad \vec{\theta}_{\text{stari}} - \alpha \vec{\nabla} J(\vec{\theta}) \tag{3.6}$$

gdje α predstavlja stopu učenja (eng. *learning rate*). Gradijentna metoda koristi ovaj gradijent kako bi pronašla smjer u kojem se funkcija gubitka najbrže smanjuje, što omogućuje pomak parametara modela u smjeru negativnog gradijenta, kako bi se smanjila greška [2]. Stopa učenja je parametar koji kontrolira korak ažuriranja, a odabire se eksperimentalno. Veći α može uzrokovati brže konvergiranje, ali može dovesti i do prevelikih koraka i prelaska preko optimalnih vrijednosti. S druge strane, premali α može uzrokovati sporije konvergiranje i zapinjanje u lokalnim minimumima. Postupak ažuriranja parametara ponavljamo sve dok se ne ispuni unaprijed postavljeni uvjet zaustavljanja, na primjer, određeni broj iteracija ili kada se funkcija gubitka stabilizira. Iako gradijentna metoda ima široku primjenu, suočava se s nekoliko problema i izazova. Prvi problem je zapinjanje u lokalnim minimumima. Gradijentna metoda nije imuna na zapinjanje u lokalnim minimumima, što znači da algoritam može ostati zaglavljen u jednom od lokalnih minimuma funkcije gubitka i doći do suboptimalnih rješenja. Drugi izazov je sporija konvergencija, ako je funkcija gubitka jako zakrivljena i ima uske doline. U takvim slučajevima, gradijentna

metoda može konvergirati vrlo sporo. Također, preveliki koraci ažuriranja parametara mogu dovesti do prelaska preko optimalnih vrijednosti, što rezultira nepreciznim rješenjima. Nestabilnost je još jedan izazov gradijentne metode, a javlja se kada je stopa učenja (α) postavljena previsoko, što može dovesti do oscilacija i nestabilnosti te spriječiti konvergenciju. Kako bi se prevladali ovi izazovi, razvijene su različite varijacije gradijentne metode, uključujući *Stochastic Gradient Descent (SGD)*, *Mini-batch Gradient Descent, Momentum, Adagrad, i Adam* [3]. Svaka od ovih varijacija ima svoje prednosti i mane, pa je važno eksperimentirati s različitim optimizacijskim algoritmima, kako bi se pronašla najbolja metoda za određeni problem strojnog učenja.

3.3 Algoritmi strojnog učenja

Unutar svakog tipa modela strojnog učenja postoje razni algoritmi dizajnirani za rješavanje specifičnih zadataka. Ovdje ćemo izložiti neke od široko korištenih algoritama za različite scenarije učenja:

Stabla odlučivanja fleksibilni su i intuitivni modeli koji se koriste za rješavanje problema klasifikacije i regresije. Funkcioniraju tako da podatke rekurzivno dijele na temelju značajki kako bi se stvorila struktura nalik stablu, gdje svaki list predstavlja klasu ili vrijednost za regresiju. Ova jednostavna struktura omogućuje da modeli budu interpretirani i razumljivi, što je od velike važnosti u primjenama gdje je bitno razumjeti donesene odluke. Osim toga, stabla odlučivanja lako se mogu nositi s podacima koji imaju različite vrste značajki, bilo da su kategoričke ili kontinuirane.

Nasumične šume (eng. *random forest*) jedna je od popularnih metoda u strojnom učenju. Ova tehnika kombinira više stabala odlučivanja kako bi poboljšala točnost predviđanja i smanjila problem prenaučenosti. Kako bismo dobili konačne predviđanje, svako stablo odlučivanja pojedinačno donosi svoju odluku, a zatim se ta odluka agregira između svih stabala. Ovaj ansambl pristup osigurava da modeli bolje generaliziraju na neviđenim podacima, što je ključno za pouzdano predviđanje u stvarnim situacijama.

Metode potpornih vektora (SVM) moćni su algoritmi koji se koriste za rješavanje problema klasifikacije. Cilj ovih metoda je pronaći optimalnu hiperplohu ili granicu koja najbolje razdvaja različite klase u prostoru značajki. SVM je posebno koristan kada se suočavamo s visokodimenzionalnim podacima jer može pronaći optimalnu granicu čak i u prostorima s mnogo značajki. Osim toga, SVM se može nositi s linearnim i nelinearnim granicama između klasa, što ga čini vrlo fleksibilnim za različite zadatke u strojnom učenju.

Neuronske mreže temeljni su modeli u području dubokog učenja, grane strojnog učenja koja je revolucionalizirala mnoga područja primjene. Inspirirane biološkim neuronskim mrežama u mozgu, ove mreže sastoje se od slojeva međusobno povezanih neurona. Svaki neuron obrađuje ulazne podatke i prenosi ih sljedećem sloju, omogućujući mreži da uči i hvata složene obrasce u podacima. Neuronske mreže izuzetno su moćni modeli koji se široko primjenjuju u područjima kao što su prepoznavanje slika, obrada jezika, pronalaženje obrazaca u velikim skupovima podataka i mnoge druge aplikacije koje zahtijevaju obradu velike količine informacija.

Iako su modeli i algoritmi strojnog učenja postigli izniman uspjeh u mnogim primjenama, postoje i neki izazovi, kao što su:

- **Kvaliteta i količina podataka:** Osiguravanje kvalitetnih i dovoljno označenih podataka ključno je za učenje preciznih modela. Međutim, prikupljanje velikih i označenih skupova podataka može biti skupo, vremenski zahtjevno ili čak nemoguće u određenim područjima.
- **Interpretabilnost modela:** Složeni modeli, poput dubokih neuronskih mreža, često su teško interpretabilni, što znači da nije uvijek jasno zašto donose određene odluke. To može biti izazov, posebno u kritičnim primjenama gdje je potrebno razumjeti razloge donesenih odluka.
- **Općenitost i prenaučenost:** Osiguravanje da modeli dobro generaliziraju na neviđenim podacima i da se ne prenauče na skupu za učenje i dalje je izazov koji zahtijeva pažljivo podešavanje parametara i pristupa.
- **Računalni resursi:** Učenje složenih modela može zahtijevati značajne računalne resurse, što može ograničiti njihovu primjenu u velikim razmjerima ili na uređajima s ograničenim resursima.

Iako postoje izazovi, napredak u strojnom učenju i dalje je izvanredan, a istraživači i praktičari intenzivno rade na rješavanju ovih izazova. Očekuje se da će daljnji razvoj u ovom području dovesti do sve robusnijih, interpretabilnijih i efikasnijih algoritama, omogućujući još širu primjenu strojnog učenja u raznim područjima i donoseći nove inovacije u području umjetne inteligencije.

3.4 Neuronske mreže

Istraživanje svijeta neuronskih mreža otkriva očaravajuće putovanje u srce umjetne inteligencije i strojnog učenja. Neuronske mreže predstavljaju vrhunac moderne umjetne inteligencije koja ima moć da razotkrije složene kompleksnosti u različitim područjima poput analize slika, obrade prirodnog jezika (NLP), autonomnih sustava i više [4]. Konceptualno, neuronske mreže digitalne su rekreacije složenih neuronskih mreža unutar ljudskoga mozga. Ove mreže grade se od međusobno povezanih

jedinica, poznatih kao neuroni ili čvorovi, grupiranih u određene slojeve koji zajedno obrađuju i tumače podatke. Svaki neuron apsorbira ulazne podatke, obrađuje ih kroz niz međusobno povezanih čvorova i konačno, generira izlaz koji se dalje širi kroz mrežu. Arhitektura neuronske mreže sastoji se od tri glavna sloja: ulaznog sloja, skrivenih slojeva i izlaznog sloja. Ulazni sloj služi kao vrata za sirove podatke, dok skriveni slojevi obavljaju složenu obradu koja na kraju vodi do željenog izlaza u izlaznom sloju. Središnji element rada neuronske mreže su veze između neurona, svaka s dodijeljenom težinom koja određuje snagu veze. Tijekom faze učenja, ove težine pažljivo se podešavaju kako bi naučile složene obrasce i odnose skrivenih u podacima. Neuroni izvode svoju magiju primjenjujući aktivacijsku funkciju na zbroj svojih ulaza. Ova funkcija uvodi nelinearnost, omogućavajući mreži da uhvati nijanse prisutne u podacima. Kada podaci prolaze kroz mrežu, događa se proces nazvan prosljeđivanje unaprijed. Ulazni podaci putuju kroz slojeve, neuroni ih obrađuju, a izlazi se generiraju na svakom sloju. Suština učenja neuronske mreže leži u fazi dubokog učenja. Učenje uključuje izlaganje mreže ulaznim podacima zajedno s odgovarajućim ciljnim izlazima. Kroz sistematičan proces, mreža usavršava svoju izvedbu iterativnim prilagodbama svojih težina, koristeći algoritme optimizacije. Glavni cilj je minimalizirati razliku između predviđanja mreže i stvarnih ciljnih izlaza. Širenje unazad (eng. *backpropagation*) pojavljuje se kao ključni algoritam u učenju neuronskih mreža. On računa gradijent greške mreže u odnosu na njene težine. Ovaj gradijent usmjerava mrežu da napravi precizne prilagodbe koje postupno smanjuju pogreške i poboljšavaju njene prediktivne sposobnosti. Ovaj iterativni mehanizam fino podešava unutarnje reprezentacije mreže, što rezultira poboljšanom izvedbom. Neuroni koji obuhvaćaju više skrivenih slojeva nazivaju se dubokim neuronskim mrežama ili modelima dubokog učenja. Ovaj pristup, poznat kao duboko učenje, zbog svoje sposobnosti hvatanja složenih obrazaca u podacima donosi značajne inovacije u različitim područjima. Praktične primjene neuronskih mreža su široke, obuhvaćajući prepoznavanje slika, analizu govora, razumijevanje prirodnog jezika (NLP), autonomna vozila, dijagnostiku zdravstvenih stanja i više. Konvolucijske neuronske mreže (CNN) posebno se ističu u obradi slika, dok rekurzivne neuronske mreže (RNN) daju vrlo dobre rezultate u rukovanju sekvencijalnim podacima, poput teksta i govora [4]. Iako neuronske mreže donose nevjerojatan potencijal, suočavaju se s vlastitim izazovima. Kvalitetni označeni podaci ključni su za učenje ovih modela, a njihova složena arhitektura nosi rizik od prenaučenosti (eng. *overfitting*). Osim toga, razumijevanje procesa donošenja odluka dubokih modela ostaje izazov, potičući istraživanja u području objašnjive umjetne inteligencije.

4 PDE-FIND algoritam

Kroz povijest, spoznaja o fizikalnim zakonima često je proizlazila iz opažanja prirodnih fenomena i eksperimentalnih istraživanja. Ovi zakoni potom su formalizirani matematičkim izrazima, omogućujući precizno opisivanje i predviđanje ponašanja prirodnih sustava. Ovaj klasični pristup omogućavao je istraživačima da dublje razumiju svijet oko sebe i pridonesu razvoju znanosti. No, s razvojem tehnologije, računalne znanosti i podatkovne analize, suočavamo se s novim izazovima. Postoje situacije u kojima eksperimenti generiraju obilje podataka, ali nemamo odgovarajuće matematičke jednadžbe koje bi precizno opisivale dinamiku tih sustava. Jednostavno rečeno, posjedujemo rezultate eksperimenata, no nedostaje nam teorijska osnova koja bi te rezultate interpretirala. Primjer za ovu novu paradigmu možemo pronaći u istraživanju kvantne dinamike složenih sustava, poput interakcija među kvarkovima i gluonima. Ovdje postoji mnogo nepoznanica u vezi s točnim međudjelovanjima i kvantnom dinamikom subatomske razine. Eksperimenti pružaju podatke o procesima, ali nemamo precizne jednadžbe koje bi opisivale kompleksne kvantne fenomene na detaljnoj razini. U takvim situacijama, moderni pristupi zasnovani na računalnoj analizi i strojnom učenju postaju ključni. Umjesto da se oslanjamo na klasične analitičke metode, koristimo moć računalnih algoritama za analizu velike količine podataka. Ovi algoritmi mogu otkriti skrivene obrasce i relacije unutar eksperimentalnih rezultata te izvesti matematičke jednadžbe koje najbolje opisuju sustav. Dakle, povijest znanosti često je bila vođena razumijevanjem fizikalnih zakona iz opažanja i matematičke formalizacije, dok moderni pristupi, koji uključuju računalnu analizu i strojno učenje, otvaraju novo područje istraživanja. Ovi pristupi omogućuju nam da koristimo obilje eksperimentalnih podataka, kako bismo otkrili matematičke jednadžbe koje upravljaju sustavima za koje do sada nismo posjedovali precizne teorijske modele. Ova interakcija između eksperimenta i računalnih algoritama otvara vrata novim saznanjima i mogućnostima za dublje razumijevanje prirodnih procesa.

4.1 Teorijska pozadina PDE-FIND algoritma

Ako pretpostavimo da eksperimentalne rezultate nekog pokusa možemo matematički formalizirati u neku funkciju $u(\vec{r}, t)$ gdje je $\vec{r}$ vektor položaja, a t vremenski trenutak tog položaja, tada bi htjeli iz tih rezultata eksperimenata dobiti diferencijalnu jednadžbu koja kao rješenje ima tu istu funkciju $u(\vec{r}, t)$. PDE-FIND algoritam koji je korišten u nastavku ovog rada radi upravo to - iz eksperimentalnih podataka određuje diferencijalnu jednadžbu koja opisuje dinamiku sustava. Algoritam radi na bazi linearne regresije s par modifikacija i napravljen je po uzoru na [5] i [6]. Algoritam koristi sljedeći model:

$$u_{tt} \equiv \ddot{u} \quad = \quad F(u, u_x, u_{xy}, \ldots, \vec{r}, t, \mu) \tag{4.1}$$

U gornjoj jednadžbi $\ddot{u}$ predstavlja drugu vremensku derivaciju neke generalizirane koordinate koju razmatramo, dok s desne strane jednadžbe imamo funkciju F koja ovisi o mnoštvu kombinacija funkcije u, vremenu i raznim drugim parametrima. Sve ove moguće parametre gledamo kako moguće članove u krajnjoj diferencijalnoj jednadžbi, a algoritam PDE-FIND izračunati će koeficijente ispred tih članova. U ovom koraku želimo popuniti funkciju F sa što više mogućih krajnjih parametara za koje smatramo da se mogu pojavljivati u diferencijalnoj jednadžbi koja bi opisivala naš sustav. Na primjer, finalna diferencijalna jednadžba koja opisuje sustav može ovisiti o parametrima $x, y^2, t, x \cdot y, \ldots$, pa nam je stoga važno da naša funkcija F u sebi sadrži sve te moguće parametre koji bi se mogli pojaviti u finalnoj diferencijalnoj jednadžbi.

4.2 Numerička obrada podataka

Kako bismo bili u mogućnosti izvršiti numeričku obradu podataka, potreban nam je diskretizirani vremenski interval. Drugim riječima, imamo vremenske točke t_1, t_2, ..., t_m, gdje je m konačan broj. Slično tome, ovo se odnosi i na prostornu dimenziju, gdje postoji n prostornih točaka x_1, x_2, ..., x_n. U tim specifičnim točkama diskretiziranog vremena i prostora, definirana je funkcija:

$$u_{ij} \equiv u(x_i, t_j) \tag{4.2}$$

Sljedeći korak je sve ove diskretizirane podatke posložiti u vektor-stupac koji će u svakom retku imati izvrjednjenu funkciju u, u određenim točkama $u(x_i, t_j)$. U vektor stupac prvo ćemo staviti vrijednost od $u(x_i, t_0)$ za svaki i. Nadalje, dodati ćemo vrijednosti $u(x_i, t_1)$ za svaki i, sve dok ne prođemo sve t_j. Vektor-stupac ćemo nazvati **U** i on će izgledati ovako:

$$\mathbf{U} \equiv \begin{bmatrix} u(x_0, t_0) \\ u(x_1, t_0) \\ \vdots \\ u(x_{n-1}, t_m) \\ u(x_n, t_m) \end{bmatrix} \tag{4.3}$$

Kako bismo mogli koristi model opisan jednadžbom (4.1), treba nam druga derivacije vektora **U** i tu dolazimo do problema derivacije diskretiziranih podataka koji je obrađen u jednom od sljedećih potpoglavlja. Kako smo ranije spomenuli, algoritam PDE-FIND zasnovan je na linearnoj regresiji, specifično na Lasso regresiji za 1-normu i Ridge regresiji za 2-normu. Sljedeći je korak stoga prikazati funkciju F iz jednadžbe (4.1), u matričnom obliku. Prevođenje modela algoritma PDE-FIND u diskretiziranu formu možemo učiniti ako pretpostavimo:

$$\mathbf{U_{tt}} = \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi \tag{4.4}$$

U gornjoj jednadžbi, $\boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q})$ predstavlja skup funkcija kandidata zajedno s funkcijama koje nisu intrinzične sustavu i kontrolnim varijablama, a ξ predstavlja vektor s koeficijentima pojedinih funkcija kandidata s kojima one ulaze u finalnu diferencijalnu jednadžbu. Konačni rezultat izgledao bi ovako:

$$\begin{bmatrix} | \\ \mathbf{U_{tt}} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | & | & | \\ \mathbf{1} & \mathbf{U} & \mathbf{U^2} & \vdots & \mathbf{UU_t} \\ | & | & | & | & | \end{bmatrix} \begin{bmatrix} | \\ \xi \\ | \end{bmatrix} \tag{4.5}$$

Ako za primjer uzmemo model harmoničnog oscilatora kružne frekvencije $\omega = 2$, naš model trebao bi nam za ξ vektor vratiti:

$$\xi = \begin{bmatrix} 0 \\ -4 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.6}$$

što odgovara diferencijalnoj jednadžbi takvog sustava:

$$\ddot{u} = -\omega^2 u \tag{4.7}$$

$$= -4u \tag{4.8}$$

Kako bi ovu metodu generalizirali sa 1D na višedimenzionalni problem, potrebno je vektore $\mathbf{U}$ i ξ proširiti sa 1D vektora na višedimenzionalne vektore, odnosno matrice.

4.3 Metode za pronalazak koeficijenata

Rješavanje jednadžbe (4.4) optimizacijski je problem, odnosno moramo pronaći koeficijente ξ_0 za čije je rješenje određena funkcija gubitka minimalizirana. Kako bi izračunali ove koeficijente, kao našu funkciju gubitka možemo koristiti metodu najmanjih kvadrata [5]:

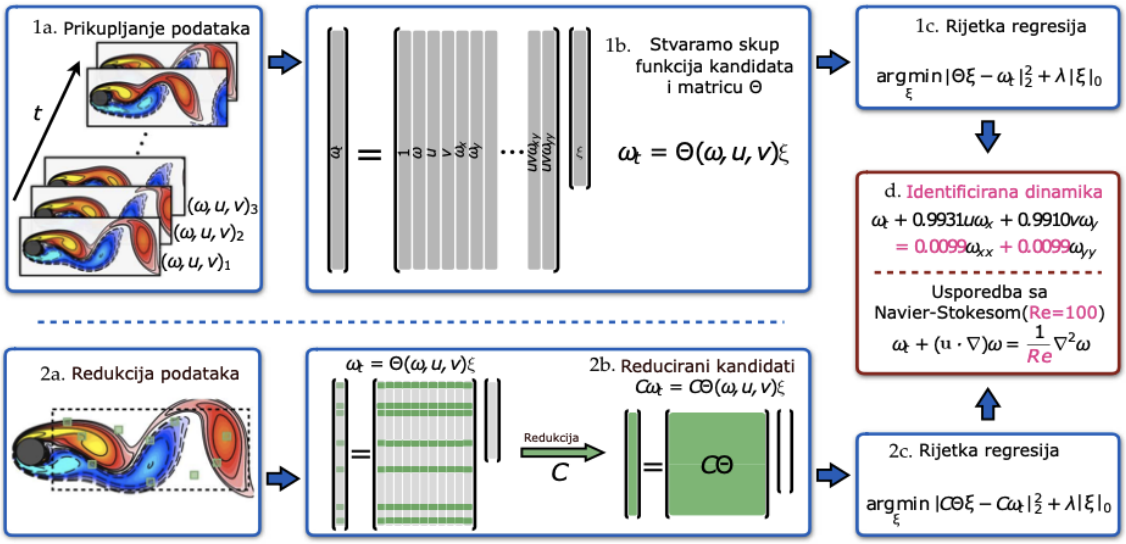
$$\xi_0 = \underset{\xi}{\operatorname{argmin}} ||\boldsymbol{\Theta}\xi - \mathbf{U_{tt}}||_2^2 \tag{4.9}$$

U gornjoj jednadžbi broj koji se nalazi u indeksu norme govori nam o kojoj je normi riječ. Ova metoda najbolje radi kada podatci nemaju šuma u sebi, što je u stvarnosti jako teško ili pak nemoguće ostvarivo. Problem koji se javlja sa šumom jest da neželjene, odnosno netočne funkcije kandidati imaju ξ koeficijente različite od 0. Posljedica toga je da korištenjem metode (4.9) na podacima sa šumom dobivamo vrlo

kompleksne jednadžbe s velikim brojem članova koje ispred sebe imaju jako male koeficijente. Ovo nam sugerira da, kako bismo riješili problem, možemo doraditi našu funkciju gubitka. Dorada dolazi u vidu dodatnog penalizacijskog člana koji će penalizirati vektor ξ prevelike norme. Drugim riječima, ako vektor ξ bude prevelike norme, on neće biti zadovoljavajući. Modificirana funkcija gubitka sada će izgledati ovako:

$$\xi_0 \quad = \quad \underset{\xi}{\operatorname{argmin}} \left\{ ||\boldsymbol{\Theta}\xi - \mathbf{U_{tt}}||_2^2 + \lambda ||\xi||_p^2 \right\} \tag{4.10}$$

U gornjoj jednadžbi možemo vidjeti da je norma drugog člana, označena sa p, varijabilna. Kada je norma drugog člana 1, onda se ona zove Lasso regresija, a kada je norma 2, onda se zove Ridge regresija [5]. U jednadžbi (4.10), parametar λ nazivamo hiperparametrom, zato što je to vanjska konfiguracijska postavka koju postavljamo prije početka procesa obuke algoritma te se ona ne mijenja tokom samog procesa obuke [8]. Parametar λ služi za penalizaciju modela. Veliki λ će ograničiti broj članova u ξ. Drugim riječima, s povećanjem hiperparametra λ, određeni članovi unutar ξ postati će 0, kako bi se smanjila norma vektora ξ, što nama pruža finalnu diferencijalnu jednadžbu s manjim brojem članova. Pronalasci iz rada [5] govore nam da je moguća i dodatna modifikacija modela (4.10) korištenjem STRidge (eng. *Sequential Threshold Ridge Regression*) algoritma. Ovaj algoritam zasnovan je na, kako i ime navodi, Ridge regresiji, i funkcionira tako da se Ridge regresija s predviđenim hipeparametrom λ provodi više puta, a prilikom svakog izvođenja eliminira se član koji je manji od parametra tolerancije. STRidge model inkorporiran je u PDE-FIND algoritam koji je preuzet od [9].



Slika 4.1: Ilustracija PDE-FIND algoritma (preuzeto iz [5] i [6])

4.4 Numerička derivacija diskretiziranih podataka

Kada se suočavamo s diskretnim podacima, kao što su eksperimentalna mjerenja ili rezultati simulacija, zadatak izračunavanja derivacija postaje temeljna nužnost. Derivacije igraju ključnu ulogu u razumijevanju ponašanja, trendova i brzina promjena različitih koeficijenata. Međutim, za razliku od kontinuiranih funkcija koje posjeduju točne analitičke derivacije, diskretni podaci predstavljaju posebne izazove, posebno kada je u pitanju precizno izračunavanje derivacija [5]. Upravo u ovom kontekstu, numeričke metode diferenciranja dolaze kao adekvatno rješenje. Numeričke metode diferenciranja bitni su alati za aproksimaciju derivacija pomoću diskretnih točaka podataka. Ove metode uključuju procjenu derivacija analizirajući vrijednosti funkcija na određenim točkama podataka i njihovim susjednim točkama. Jedna posebno učinkovita tehnika u ovom kontekstu, ujedno i tehnika koju smo koristili, jest metoda centralne diferencijacije. Metoda centralne diferencijacije numerički je pristup koji nudi izbalansiranu razmjenu između preciznosti i jednostavnosti prilikom aproksimiranja derivacija iz diskretnih podataka. Kao takva, u usporedbi s drugim jednostavnijim metodama, ova je metoda posebno korisna kada je potrebna veća preciznost [7]. Njena ugrađena uravnoteženost proizlazi iz korištenja vrijednosti funkcija s obje strane točke interesa, što je čini robusnom i široko primjenjivom. U osnovi, metoda centralne diferencijacije aproksimira derivaciju funkcije u određenoj točki x, uzimajući u obzir razliku između vrijednosti funkcije na dvije susjedne točke, i dijeleći tu razliku sa korakom h:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} + \mathbf{O}(h^2) \tag{4.11}$$

Ovdje, h predstavlja korak, odnosno interval između točaka podataka, a $\mathbf{O}(h^2)$ numeričku pogrešku kod računanja derivacije. Formulacija uključuje računanje razlike između vrijednosti funkcije na $(x-h)$ i vrijednosti funkcije na $(x+h)$, što se zatim dijeli s $2h$ kako bi se dobila aproksimacija derivacije. Metoda centralne diferencijacije nudi nekoliko značajnih prednosti. Prvo, pruža preciznost, s obzirom da uzima vrijednosti funkcija s obje strane točke interesa. To je u kontrastu s jednostranim metodama, gdje je preciznost ograničena samo na jednu stranu. Simetrija formule metode centralne diferencijacije umanjuje određene pogreške i netočnosti, što doprinosi boljoj preciznosti. Dodatno, metoda pokazuje preciznost drugog reda, što znači da se greška kvadratno smanjuje s korakom h, što omogućuje bržu konvergenciju. Unatoč svojim prednostima, metoda centralne diferencijacije nije bez izazova. Jedno ključno razmatranje upravo je odabir koraka h. Odabir malog h poboljšava preciznost, ali može uvesti pogreške pri zaokruživanju, zbog konačne preciznosti računala. Suprotno tome, odabir velikog h može dovesti do manje preciznih aproksimacija derivacija. Još jedan faktor koji treba uzeti u obzir je priroda samih podataka. U slučajevima nepravilno raspoređenih točaka podataka, metoda centralne diferencijacije možda neće pružiti točne rezultate. Tada može biti potrebno koris-

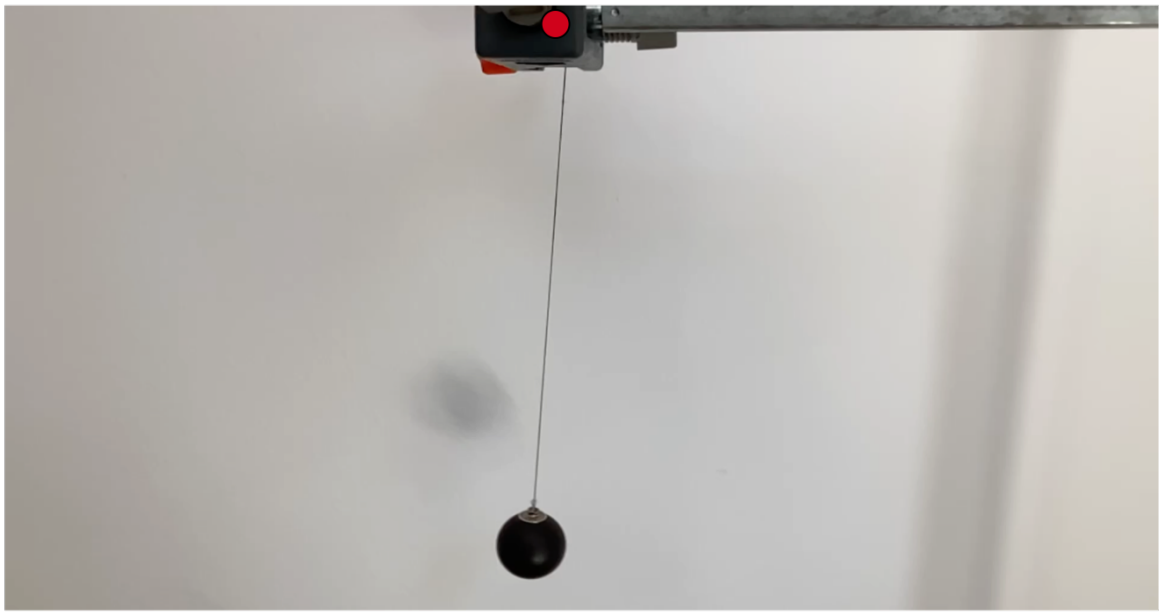
titi i tehnike interpolacije ili neke druge alternativne metode. Osim toga, prisutnost šuma u podacima može negativno utjecati na preciznost aproksimacija derivacija. Šum može unijeti fluktuacije, koje zauzvrat mogu utjecati na kvalitetu aproksimacija. Glađenje ili filtriranje podataka može se pojaviti kao nužan korak obrade, kako bi se ublažili ovi učinci. Također, efekti granice mogu predstavljati izazove, s obzirom da metoda centralne diferencijacije zahtijeva točke podataka s obje strane točke interesa, a samim time i blizu granica raspona podataka. Stoga, precizno izračunavanje derivacija može postati složeniji zadatak. Zaključno, metoda centralne diferencijacije služi kao vrijedan alat za aproksimaciju derivacija iz diskretnih podataka. Ona postiže ravnotežu između preciznosti i jednostavnosti, koristeći vrijednosti funkcija s obje strane točke interesa kako bi, u usporedbi s jednostranim metodama, pružila bolje rezultate. Međutim, prilikom primjene ove metode, nužno je pažljivo razmatranje koraka, kvalitete podataka, šuma i efekata granica. Kako bi obuhvatili ove čimbenike, prilagodba metode osigurava točne aproksimacije derivacija i pouzdane rezultate, pogotovo kada se radi s diskretnim podacima.

5 Mjerenja i rezultati

U ovom poglavlju, usredotočiti ćemo se na detaljno prikazivanje procesa mjerenja i analize rezultata na klasičan način, kao i putem PDE-FIND algoritma u svrhu testiranja algoritma na stvarnim podacima koji će u sebi sadržavati šum. Za postizanje ovog cilja, koristiti ćemo OpenCV (eng. *Open Source Computer Vision Library*) paket u programskom jeziku *Python*, kako bismo prikupili što preciznije podatke o pozicijama matematičkog njihala i mase na opruzi iz snimljenih videozapisa. Kroz ovu metodu, dobiveni podaci postati će osnova za izvođenje kutova njihala i identifikaciju parcijalnih diferencijalnih jednadžbi, koristeći gore opisan PDE-FIND algoritam. Ovaj dio ima poseban značaj jer omogućuje uvid u to kako smo dobili podatke koji su osnova za našu analizu, te istovremeno naglašava važnost OpenCV paketa u našem istraživačkom pristupu.

OpenCV moćan je alat za računalni vid, temeljen na otvorenom kodu (eng. *open-source*) koji se široko koristi za detekciju i praćenje objekata u videozapisima. Za ekstrapolaciju pozicija objekata putem neuronskih mreža, OpenCV pruža nam robusan set alata. Osnova OpenCV-a su unaprijed naučeni modeli dubokog učenja, posebno konvolucijske neuronske mreže (CNN) za prepoznavanje objekata i obrazaca u slikama i videozapisima. Proces ekstrapolacije pozicija objekata kroz vrijeme uključuje detekciju, analizu okvira, izvlačenje pozicija, vremensku analizu te korake interpolacije i ekstrapolacije. Videozapis obrađuje se okvir po okvir,a neuronska mreža analizira svaki okvir i identificira prisutne objekte. Izvučene pozicije iz uzastopnih okvira koriste se za praćenje kretanja objekata. Kombinacija OpenCV-a i neuronskih mreža pruža učinkovito rješenje za ekstrapolaciju pozicija objekata u videozapisima, što ima primjene ne samo u našem slučaju već i u različitim područjima kao što su nadzor, autonomna vozila, analiza sportskih događaja *itd*.

Videozapisi snimljeni su u kontroliranim laboratorijskim uvjetima u praktikumu PMF Fizike. Snimljeni su videozapisi matematičkog njihala i mase na opruzi u *Full HD* rezoluciji od 1920x1080 *pixela*. Matematičko njihalo prikazuje vrlo malu količinu precesije, što je očekivano zbog eksperimentalnog postava. U nastavku se mogu vidjeti slike eksperimentalnog postava, pri čemu crvene točke označavaju ovjesište oba sustava.



Slika 5.1: Eksperimentalni postav matematičkog njihala s označenim ovjesištem

Slika 5.2: Eksperimentalni postav mase na opruzi s označenim ovjesištem

5.1 Laboratorijska mjerenja

Eksperimentalni postav identičan je onome u videozapisima i vidljiv je na slikama (5.1) i (5.2). U ranijem poglavlju, putem lagranžijana izračunali smo diferencijalne jednadžbe za oba promatrana slučaja i dobili sljedeće:

$$\ddot{y} + \omega^2 y = 0 \tag{5.1}$$

$$\ddot{\theta} + \omega^2 \theta = 0 \tag{5.2}$$

Gdje jednadžba (5.1) predstavlja parcijalnu diferencijalnu jednadžbu koja opisuje masu na opruzi, a jednadžba (5.2) predstavlja parcijalnu diferencijalnu jednadžbu koja opisuje matematičko njihalo. U nastavku ovog potpoglavlja opisati ćemo kako smo proveli mjerenja za period oba oscilatora te iz istih izračunali pojedine kružne frekvencije sa pripadnim nesigurnostima.

5.1.1 Masa na opruzi

Kako bismo odredili koeficijent ω^2 za parcijalnu diferencijalnu jednadnadžbu koja opisuje dinamiku mase na opruzi, koristit ćemo se sljedećom relacijom:

$$\omega^2 = \frac{4\pi^2}{T^2} \tag{5.3}$$

Iz ovoga je vidljivo da prvo moramo izmjeriti period T. Period ćemo mjeriti 5 puta. Kako bismo minimalizirali grube greške, mjeriti ćemo vrijeme potrebno da oscilator napravi 10 punih titraja te dobiveno vrijeme podijeliti sa 10, kako bismo dobili period. Rezultati mjerenja vidljivi su u tablici (5.1).

n	$T_{10}[\mathrm{s}]$	$T\ [\mathrm{s}]$
1	7.325	0.7325
2	7.431	0.7431
3	7.127	0.7127
4	7.509	0.7509
5	7.702	0.7702

Tablica 5.1: Rezultati mjerenja perioda mase na opruzi

Nadalje, trebamo izračunati srednju vrijednost perioda koristeći standardni izraz:

$$\overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i \tag{5.4}$$

Koristeći izraz (5.4) dobivamo sljedeću srednju vrijednost za period:

$$\overline{T} = 0.74188\,\mathrm{s} \tag{5.5}$$

Sljedeći korak je izračunati nepreciznost mjerenja, odnosno standardnu devijaciju, što radimo putem sljedećeg izraza:

$$\sigma_T = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})}{n-1}} \tag{5.6}$$

$$\sigma_T = 0.019101\,\mathrm{s} \tag{5.7}$$

Sada računamo srednju kvadratnu pogrešku aritmetičke sredine M_T kao:

$$M_T = \frac{\sigma_T}{\sqrt{n}} \tag{5.8}$$
$$M_T = 0.008524\,\mathrm{s} \tag{5.9}$$

Rezultat (5.5) omogućuje nam da izračunamo prosječni kvadrat kružne frekvencije kao:

$$\overline{\omega^2} = \frac{4\pi^2}{\overline{T}^2} \tag{5.10}$$
$$\overline{\omega^2} = 71.72861\,\mathrm{s}^{-1} \tag{5.11}$$

Dalje računamo nepouzdanost M_ω kao:

$$M_\omega = \sqrt{\left(\frac{\partial\omega}{\partial T}M_T\right)^2} \tag{5.12}$$
$$M_\omega = \sqrt{\left(-\frac{8\pi^2}{\overline{T}^3}M_T\right)^2} \tag{5.13}$$
$$M_\omega = 1.6482\,\mathrm{s}^{-2} \tag{5.14}$$

Što nam za krajnji rezultat kružne frekvencije daje izraz:

$$\omega^2 = (72 \pm 2)\,\mathrm{s}^{-2} \tag{5.15}$$
$$R_\omega = 3\% \tag{5.16}$$

Naposljetku dolazimo do parcijalne diferencijalne jednadžbe:

$$\ddot{y}(t) + 72y(t) = 0 \tag{5.17}$$

5.1.2 Matematičko njihalo

U ovom potpoglavlju pokazati ćemo postupke koje smo poduzeli za izračunavanje kvadrata kružne frekvencije matematičkog njihala i time ćemo odrediti parcijalnu diferencijalnu jednadžbu koja opisuje dinamiku tog sustava. Koristiti ćemo isti izraz za izračun kružne frekvencije kao i kod mase na opruzi:

$$\omega^2 = \frac{4\pi^2}{T^2} \tag{5.18}$$

Kao i u slučaju s masom na opruzi. mjeriti ćemo period titranja. Period ćemo mjeriti 5 puta, a kako bismo minimalizirali grube greške, mjeriti ćemo vrijeme potrebno da

n	T_{10}[s]	T [s]
1	10.324	1.0324
2	10.756	1.0756
3	9.879	0.9879
4	10.028	1.0028
5	9.781	0.9781

Tablica 5.2: Rezultati mjerenja perioda matematičkog njihala

oscilator napravi 10 punih titraja, te to dobiveno vrijeme podijeliti sa 10 kako bismo dobili period. Rezultati mjerenja vidljivi su u tablici (5.2).
Sljedeći korak je izračunati srednju vrijednost perioda koristeći standardni izraz:

$$\overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i \tag{5.19}$$

Putem izraza (5.19) dobivamo srednju vrijednost za period:

$$\overline{T} = 1.0188\,\mathrm{s} \tag{5.20}$$

Sada računamo nepreciznost mjerenja, odnosno standardnu devijaciju, putem sljedećeg standardnog izraza:

$$\sigma_T = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})}{n-1}} \tag{5.21}$$

$$\sigma_T = 0.036856\,\mathrm{s} \tag{5.22}$$

Nadalje, računamo srednju kvadratnu pogrešku aritmetičke sredine M_T kao:

$$M_T = \frac{\sigma_T}{\sqrt{n}} \tag{5.23}$$

$$M_T = 0.016482\,\mathrm{s} \tag{5.24}$$

Rezultat (5.20) omogućuje izračunavanje prosječnog kvadrata kružne frekvencije kao:

$$\overline{\omega^2} = \frac{4\pi^2}{\overline{T}^2} \tag{5.25}$$

$$\overline{\omega^2} = 38.03863\,\mathrm{s}^{-2} \tag{5.26}$$

Nadalje, računamo nepouzdanost M_ω kao:

$$M_\omega = \sqrt{\left(\frac{\partial\omega}{\partial T}M_T\right)^2} \tag{5.27}$$

$$M_\omega = \sqrt{\left(-\frac{8\pi^2}{\overline{T}^3}M_T\right)^2} \tag{5.28}$$

$$M_\omega = 1.23064\,\mathrm{s}^{-2} \tag{5.29}$$

Što nam za krajnji rezultat kružne frekvencije daje:

$$\omega^2 = (38 \pm 1)\,\mathrm{s}^{-2} \tag{5.30}$$

$$R_\omega = 3\% \tag{5.31}$$

Te sada dobivamo parcijalnu diferencijalnu jednadžbu koja opisuje sustav:

$$\ddot{\theta}(t) + 38\,\theta(t) = 0 \tag{5.32}$$

5.2 Određivanje dinamike sustava putem PDE-FIND algoritma

5.2.1 Ekstrakcija pozicija centra mase matematičkog njihala i mase na opruzi

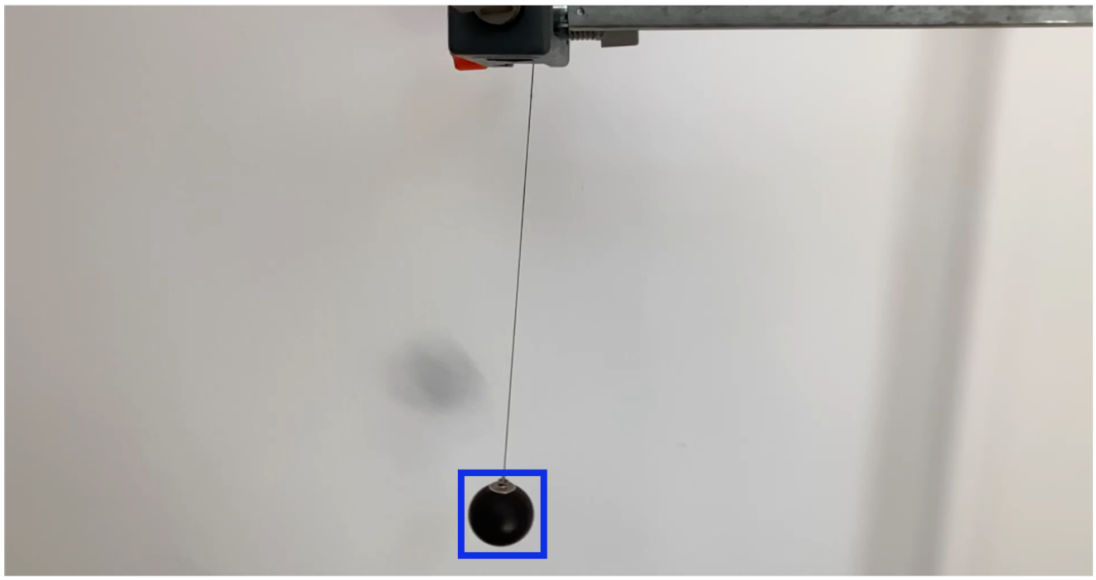
U daljnjem tekstu spominjati ćemo samo matematičko njihalo, no proces ekstrakcije pozicije za masu na opruzi je analogan. Kako bismo ekstrahirali poziciju centra mase matematičkog njihala, moramo učitati videozapis i inicijalizirati prateći okvir (eng. *tracker*), što radimo sljedećim naredbama:

```
cap = cv2.VideoCapture('masafs.mp4')
tracker = cv2.TrackerCSRT_create()
```

Nadalje, trebamo učitati prvi okvir videozapisa i na njemu označiti objekt koji želimo pratiti, što se izvodi sljedećim naredbama:

```
ret, frame = cap.read()
bbox = cv2.selectROI(frame, False)
tracker.init(frame, bbox)
```

Gore navedene naredbe će prilikom izvođenja koda otvoriti prozor u kojem ćemo plavim pravokutnikom označiti objekt koji želimo pratiti. Primjer toga vidljiv je na slici (5.3).
Nadalje, trebamo proći kroz svaki okvir videozapisa te ažurirati prozor za praćenje u tom novom okviru te zabilježiti njegovu poziciju u formi x i y koordinata koje spremamo u listu *positions*. Naredbe za izvedbu ovoga sa popratnim komentarima su:

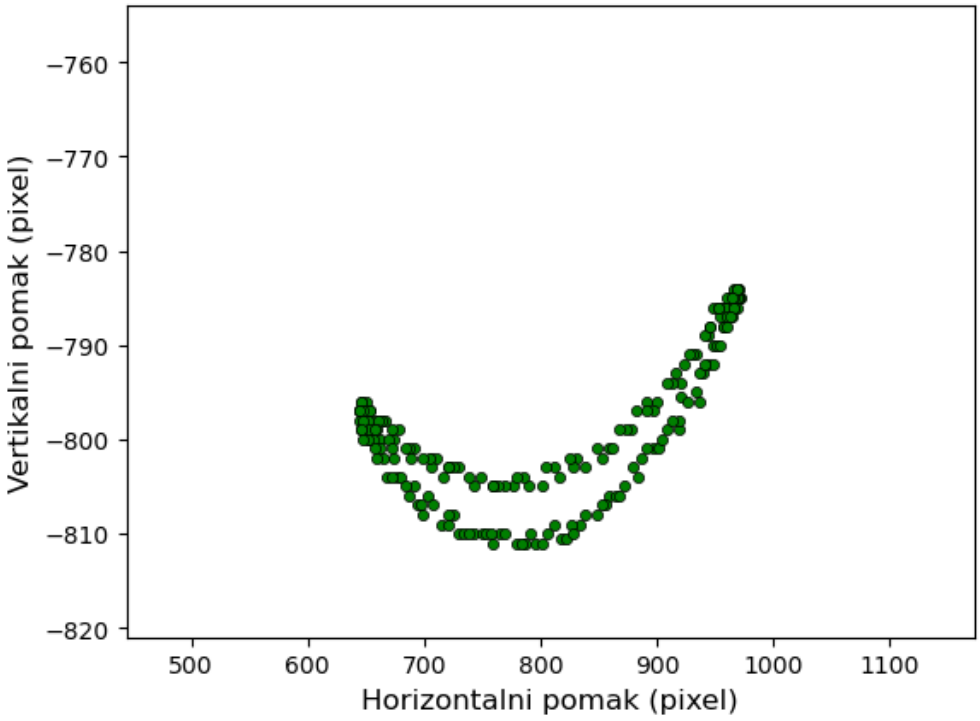


Slika 5.3: Odabir objekta za praćenje

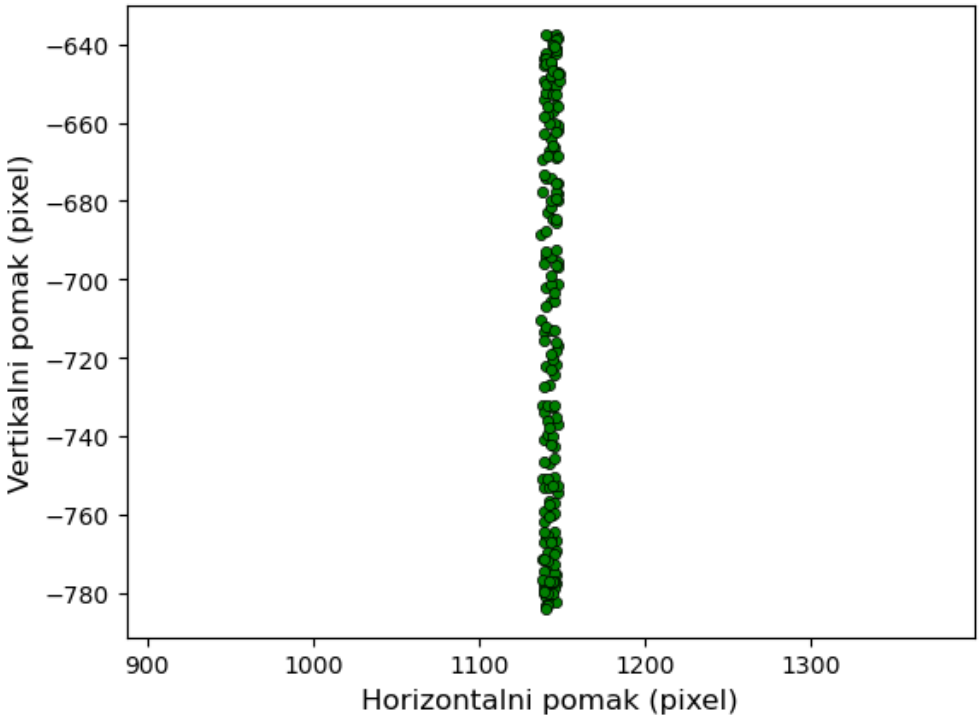
```
while True:

    ret, frame = cap.read()

    # break kad nema okvira
    if not ret:
        break
    # ažuriramo tracker u novom okviru
    ok, bbox = tracker.update(frame)

    if ok:
        x, y, w, h = [int(i) for i in bbox]
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        position = (x + w/2, y + h/2)  # centar tracking box-a
        positions.append(position)

    cv2.imshow('frame', frame)
```

U potonjem dijelu koda možemo vidjeti naredbe koje izračunavaju trenutnu poziciju centra mase u svakom okviru videozapisa. Ovaj izračun je potreban zato što *OpenCV* prati poziciju gornjeg lijevog kuta okvira za praćenje pa je potrebno uz pomoć visine i širine okvira za praćenje izračunati poziciju sredine okvira, što odgovara poziciji centra mase matematičkog njihala. Sada kada imamo sve x i y koordinate matematičkog njihala spremljene u listu, možemo nacrtati grafove svih zabilježenih pozicija. Ti grafovi vidljiv su na slikama (5.4) i (5.5). Iz priloženog možemo uočiti da stvarni podatci uistinu imaju nezanemarivu količinu šuma. Na grafu je šum dodatno izražen zbog

skale x i y koordinata.



Slika 5.4: Sve zabilježene pozicije centra mase matematičkog njihala



Slika 5.5: Sve zabilježene pozicije centra mase mase na opruzi

5.2.2 Izračun kuta otklona

Kako bismo mogli izračunati kut otklona, potrebno je odrediti točku uporišta i ravnotežni položaj centra mase. Da naš kod bude što lakše primjenjiv, potrebno je izbjeći izravno kodiranje ovih točaka i matematički izračunati pozicije ove dvije točke interesa. Izračun x i y koordinate centra mase iz svih položaja radimo sljedećom funkcijom:

```
def ravnotezniPolozaj(x,y):

    sx = x.copy()
    sy = y.copy()

    # uzmemo 10 maksimalnih x-eva s lijeve i desne strane te ih uprosječimo,
    # pa na polovištu leži xrp
    sx.sort()
    tmpx1 = [sx[i] for i in range(10)]
    tmpx2 = [sx[-i] for i in range(1,11,1)]
    tmpx1 = sum(tmpx1) / len(tmpx1)
    tmpx2 = sum(tmpx2) / len(tmpx2)
    xrp = 0.5*(tmpx1+tmpx2)

    # isti postupak sa y-ima samo uzmemo maksimume
    sy.sort()
    yrp = [sx[-i] for i in range(1,11,1)]
    yrp = sum(yrp) / len(yrp)

    return xrp,yrp
```

Gornja funkcija prvo uzima polje sa svim x koordinatama koje je matematičko njihalo imalo u videozapisu te ga sortira po veličini. Zatim uzimamo 10 ekstrema s lijeve i 10 ekstrema s desne strane te izračunamo aritmetičku sredinu kako bi pronašli x koordinatu ravnotežnog položaja. Za izračun y koordinate ravnotežnog položaja imamo skoro analogan postupak, samo ovaj put uzimamo samo 10 maksimuma i računanjem njihove aritmetičke sredine dobivamo y koordinatu ravnotežnog položaja.

Izračun položaja uporišta matematičkog njihala malo je kompliciraniji proces i temelji se na putanji matematičkog njihala. Tokom svoje putanje u idealnom slučaju matematičko njihalo iscrtava dio kružnice i tu činjenicu možemo iskoristiti. Svaka kružnica jednoznačno je određena trima nekolinearnim točkama i iz koordinata tih točaka centar kružnice može se odrediti sljedećim jednadžbama:

$$A = (x_1^2 + y_1^2)(y_2 - y_3) + (x_2^2 + y_2^2)(y_3 - y_1) + (x_3^2 + y_3^2)(y_1 - y_2) \quad (5.33)$$

$$B = (x_1^2 + y_1^2)(x_3 - x_2) + (x_2^2 + y_2^2)(x_1 - x_3) + (x_3^2 + y_3^2)(x_2 - x_1) \quad (5.34)$$

$$C = 2(x_1(y_2 - y_3) - y_1(x_2 - x_3) + x_2y_3 - y_2x_3) \quad (5.35)$$

$$D = (x_1^2 + y_1^2)(x_2y_3 - y_2x_3) + (x_2^2 + y_2^2)(x_3y_1 - y_3x_1) + (x_3^2 + y_3^2)(x_1y_2 - y_1x_2) \quad (5.36)$$

$$E = (x_1(y_2 - y_3) - y_1(x_2 - x_3))(x_2y_3 - y_2x_3) + (x_2(y_3 - y_1) - y_2(x_3 - x_1))(x_3y_1 - y_3x_1) + (x_3(y_1 - y_2) - y_3(x_1 - x_2))(x_1y_2 - y_1x_2) \quad (5.37)$$

$$F = (x_1(y_2 - y_3) - y_1(x_2 - x_3))^2 + (x_2(y_3 - y_1) - y_2(x_3 - x_1))^2 + (x_3(y_1 - y_2) - y_3(x_1 - x_2))^2 \quad (5.38)$$

$$x_{\text{kr}} = \frac{D + E}{2F} \quad (5.39)$$

$$y_{\text{kr}} = \frac{A + B}{2C} \quad (5.40)$$

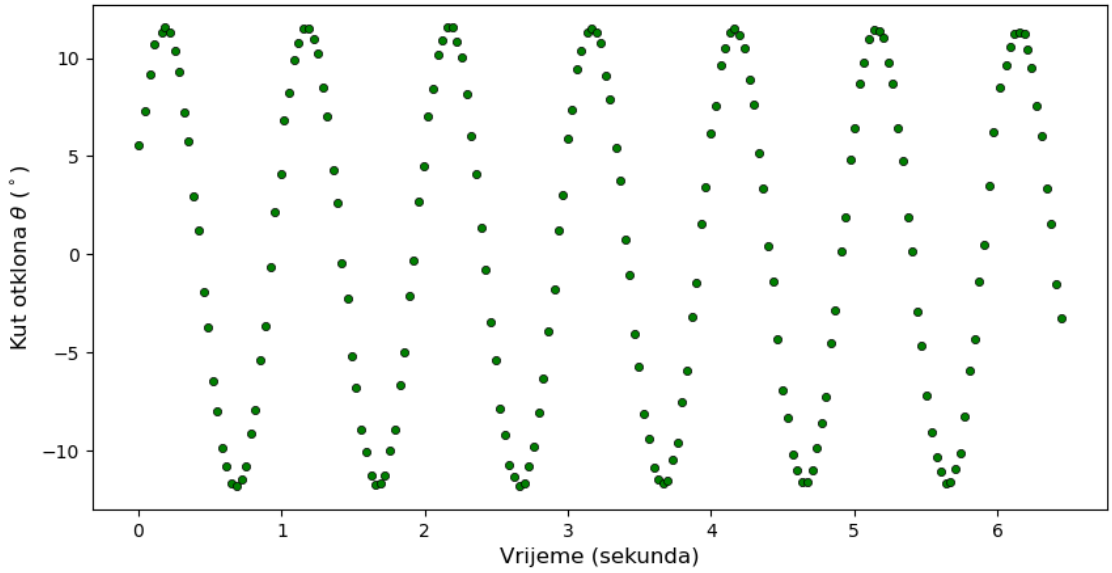
Gdje su x_{kr} i y_{kr} koordinate središta kružnice koje, u našem slučaju, odgovara ovjesištu matematičkog njihala. Kako bi ove jednadžbe implementirali u naš kod, prvo moramo imati tri nekolinearne točke i u tu svrhu radimo funkciju koja će uzeti tri nasumične točke i testirati jesu li te točke kolinearne. Implementacija te funkcije izgleda ovako:

```
def randNCL(points):

    # randomiziramo točke
    random.shuffle(points)

    for i in range(len(points)-2):
        x1, y1 = points[i]
        x2, y2 = points[i+1]
        x3, y3 = points[i+2]

        # preko determinante provjerimo jesu li kolinearne
        det = x1*(y2-y3) - y1*(x2-x3) + x2*y3 - y2*x3
        if det != 0:
            return x1, y1, x2, y2, x3, y3

    return None
```

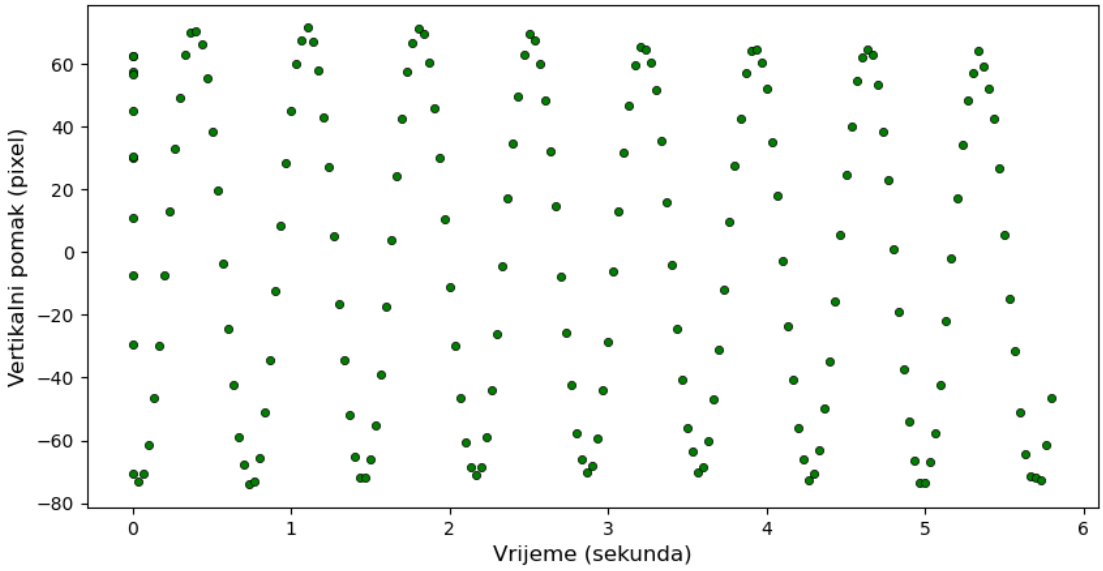
Gore navedena funkcija prvo nasumično pomiješa točke u ulaznom polju te putem *for* petlje uzima tri po tri točke i provjera jesu li kolinearne. Kolinearnost se provjerava izvrjednjavanjem determinante:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{5.41}$$

Determinanta različita od 0 naznačuje da točke nisu kolinearne i da ih možemo koristiti kako bi izračunali koordinate središta kružnice koristeći gore navedene jednadžbe i samim time koordinate uporišta našeg matematičkog njihala. Sada možemo ispisati kutove na $\theta \backslash t$ graf, s obzirom da su kutovi θ izračunati za svaki okvir sekvencijalno, poredani su u točnom vremenskom poretku. Vremenski razmak između dva zabilježena kuta θ jednak je 30^{-1} sekundi, s obzirom da je naša kamera snimala 30 okvira po sekundi. Ovaj graf vidljiv je na slici (5.7).



Slika 5.6: Graf ovisnosti kuta otklona θ i vremena (t) za matematičko njihalo



Slika 5.7: Graf ovisnosti vertikalnog pomaka (y) i vremena (t) za masu na opruzi

5.2.3 Provedba PDE-FIND algoritma

Sada imamo listu s vremenski korektnim poretkom generalizirane varijable θ (u slučaju mase na opruzi generalizirana varijabla je y), pa možemo provesti PDE-FIND algoritam. Korištenje PDE-FIND algoritma može se vidjeti u sljedećem isječku koda:

```
ut = np.zeros((m,n), dtype=np.complex64)
utt = np.zeros((m,n), dtype=np.complex64)

for i in range(m):
    ut[i,:] = FiniteDiff(u[i,:], dt, 1)
    # koristi se kasnije kod funkcije TrainSTRidge
    utt[i,:] = FiniteDiff(u[i,:], dt,2)

ut = np.reshape(ut, (n*m,1), order='F')
utt = np.reshape(utt, (n*m,1), order='F')

X_ders = np.hstack([np.ones((n*m,1)),ut])
X_data = np.hstack([np.reshape(u, (n*m,1), order='F')])
derivatives_description = ['','u_t']

X, descr = build_Theta(X_data, X_ders, derivatives_description,
P=2, data_description = ['u'])
descr

w = TrainSTRidge(X,utt,100,10)
```

```
print("PDJ dobivena pomoću STRidge:")
print_pde(w, descr)
```

U ovom isječku koda prvo se inicijaliziraju funkcije kandidati i za ovaj slučaj odabrane su funkcije kandidati: 1, u, $\dot{u}$, $u\dot{u}$, u^2 i u^3. Zatim se ove varijable napune s adekvatnim podacima i preoblikuju u adekvatnu formu za nastavak rada algoritma. Dalje se stvaraju različite matrice X_{ders} i X_{data} koje će se koristiti za izgradnju matrice X koja je dio ulaza za STRidge algoritam. Matrica X_{data} sadrži podatke generalizirane varijable u, sinusa, kosinusa i eksponencijalne funkcije, dok matrica X_{ders} sadrži derivacije generalizirane varijable u. Koristeći funkciju *build_Theta*, izgrađuje se matrica X koja će se koristiti za treniranje STRidge algoritma. Ova matrica kombinira podatke iz matrica X_{data} i X_{ders}. STRidge algoritam trenira se pozivom funkcije *TrainSTRidge* koja uzima matricu X, ciljni niz `untt`, težinu i parametar za regularizaciju kao ulaz. Rezultat treniranja (koeficijenti `w`) pohranjuju se pa se ispisuje dobivena parcijalna diferencijalna jednadžba koristeći funkciju *print_pde*.

5.2.4 Rezultat za matematičko njihalo

Dolje možemo vidjeti ispis funkcije PDE-FIND koji u sebi sadrži parcijalnu diferencijalnu jednadžbu za matematičko njihalo koju je algoritam izračunao:

```
PDJ dobivena pomoću STRidge:
u_tt = (-33.566908 +0.000000i)u
```

Jednadžba koju smo izračunali klasičnim pristupom bila je:

$$\ddot{\theta}(t) \quad = \quad -\omega^2\theta(t), \quad \omega^2 = (38 \pm 1)\,\mathrm{s}^{-2} \tag{5.42}$$

Parcijalna diferencijalna jednadžbe izračunate putem PDE-FIND algoritma odstupa od klasično izračunate parcijalne diferencijalne jednadžbe s relativnog greškom od 11.7% za faktor ω^2.

5.2.5 Rezultat za masu na opruzi

U donjem ispisu vidimo parcijalnu diferencijalnu jednadžbu za masu na opruzi koju je algoritam izračunao:

```
PDJ dobivena pomoću STRidge:
u_tt = (-0.641566 +0.000000i)u_t
    + (-65.139293 +0.000000i)u
    + (0.005660 +0.000000i)u^2
```

Jednadžba izračunata klasičnim pristupom bila je:

$$\ddot{y}(t) \;=\; -\omega^2\, y(t), \quad \omega^2 = (72 \pm 2)\,\mathrm{s}^{-2} \tag{5.43}$$

Gušenje smo zanemarili u klasičnim mjerenjima, no možemo primijetiti i da je PDE-FIND algoritam pronašao i koeficijent koji stoji uz prvu vremensku derivaciju pomaka. To nam govori da u sustavu zbilja postoji mala količina gušenja. Dobili smo i mali koeficijent uz u^2, što nam govori da opruga nije u sasvim linearnom režimu. Drugim riječima, kako se pomak iz ravnotežnog položaja povećava ili smanjuje, sila ne raste sasvim linearno. To znači da, kako se sve više isteže ili stišće, opruga postaje čvršća, što je karakteristično za nelinearne opruge. Parcijalna diferencijalna jednadžba izračunata putem PDE-FIND algoritma odstupa od klasično izračunate parcijalne diferencijalne jednadžbe s relativnom pogreškom od 9.53% za faktor ω^2.

5.2.6 Izvori pogreške

U našem eksperimentalnom postavu identificirano je nekoliko potencijalnih izvora pogreške koji su mogli utjecati na točnost naših rezultata. Prvo, pogreške praćenja u algoritmu koji smo koristili mogu unijeti netočnosti u podatke o položaj koje bi se dalje propagirale kroz naš izračun pozicije centra mase njihala. Nadalje, pojava paralakse mogla je utjecati na naše rezultate, s obzirom da svaka nesrazmjernost između kuta snimanja kamere i gibanja njihala može dovesti do razlika u vidljivom položaju njihala. Treba također razmotriti rezoluciju videozapisa jer ona može biti ograničavajući faktor preciznosti naših mjerenja položaja, posebno za male pomake njihala. Konačno, utjecaj okolišnih čimbenika, kao što su vibracije eksperimentalnog postava također su mogle utjecati na ponašanje njihala i samim time na naša mjerenja. Važno je prepoznati ove potencijalne izvore pogreške i uzeti ih u obzir u našoj analizi, važno je i ne zaboraviti moguće ljudske greške koje su se mogle dogoditi tijekom postavljanja eksperimenta, prikupljanja podataka ili analize, koje bi također doprinijele netočnostima u našim rezultatima.

6 Zaključak

U ovom diplomskom radu provedeno je istraživanje korištenjem PDE-FIND algoritma za pronalazak parcijalnih diferencijalnih jednadžbi koje opisuju dinamiku matematičkog njihala i mase na opruzi. Napravljena je video snimka navedenih sustava te kako bi se izračunali x i y koordinate centra mase iz svih položaja, koristila se funkcija koja se temelji na sortiranju i uzimanju ekstrema x i y koordinata. Za izračun položaja uporišta matematičkog njihala koristila se analiza putanje matematičkog njihala, pri čemu su korištene determinante za provjeru kolinearnosti točaka i izračun koordinata uporišta. Nakon pripreme podataka, primijenjen je PDE-FIND algoritam kako bi se pronašle diferencijalne jednadžbe koje opisuju dinamiku sustava. Rezultati su uspoređeni s klasičnim pristupom, gdje su izvedene teoretske diferencijalne jednadžbe za oba sustava. Usprkos jednostavnosti načina snimanja, uspješno je rekonstruirana dinamika oba promatrana sustava te je pružen i uvid u dodatne članove, kao primjerice gušenje, koji su zanemareni u klasičnoj obradi. Rezultati ukazuju na ograničenja PDE-FIND algoritma u preciznom modeliranju dinamike sustava. Međutim, PDE-FIND algoritam može biti koristan alat za brzo generiranje parcijalnih diferencijalnih jednadžbi za kompleksne sustave gdje je teško ili nemoguće ručno izvesti analizu. Osim što je PDE-FIND algoritam pokazao svoja ograničenja u vrlo preciznom modeliranju dinamike sustava, važno je istaknuti i nekoliko drugih bitnih aspekata ovog istraživanja. Prvo, ovakvi rezultati ukazuju na važnost razumijevanja ograničenja i pretpostavki koje se koriste u primjeni PDE-FIND algoritma i njemu sličnih algoritama. PDE-FIND algoritam oslanja se na određene pretpostavke o obliku diferencijalnih jednadžbi, a odstupanja u rezultatima mogu proizaći iz nepreciznih pretpostavki ili šuma u podacima. Drugo, PDE-FIND algoritam može biti koristan alat za generiranje početnih modela diferencijalnih jednadžbi ili za brzo istraživanje dinamike nekog kompleksnoga sustava. Algoritam može služiti kao polazna točka za daljnju razradu modela ili za brzo generiranje aproksimacija sustava, što se može pokazati kao koristan alat za fizičare. Treće, važno je napomenuti da je izazovno dobiti precizne modele dinamike nekog sustava iz stvarnih podataka jer su stvarni sustavi često izloženi raznim vanjskim utjecajima. U takvim slučajevima, PDE-FIND algoritam može pružiti korisne smjernice i pomoći istraživačima u boljem razumijevanju sustava. U konačnici, ovo istraživanje ukazuje na važnost kombiniranja različitih pristupa i tehnika prilikom modeliranja i analize dinamike sustava. Klasični pristupi ostaju pouzdani, ali alati poput PDE-FIND algoritma mogu dopuniti istraživanje i ubrzati proces generiranja parcijalnih diferencijalnih jednadžbi za složene sustave.

Dodaci

Dodatak A Python kod

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import random

# funkcija za pronalazak ravnotezni položj njihala
def ravnotezniPolozaj(x,y):

    sx = x.copy()
    sy = y.copy()

    # uzmemo 10 maksimalnih x-eva s lijeve i desne strane te ih uprosječimo,
    # pa na polovištu leži xrp
    sx.sort()
    tmpx1 = [sx[i] for i in range(10)]
    tmpx2 = [sx[-i] for i in range(1,11,1)]
    tmpx1 = sum(tmpx1) / len(tmpx1)
    tmpx2 = sum(tmpx2) / len(tmpx2)
    xrp = 0.5*(tmpx1+tmpx2)

    # isti postupak sa y-ima samo uzmemo maksimume
    sy.sort()
    yrp = [sx[-i] for i in range(1,11,1)]
    yrp = sum(yrp) / len(yrp)

    return xrp,yrp


#   funkcija za traženje centra kružnice preko tri ne kolinearne točke
def centar(x1, y1, x2, y2, x3, y3):

    A = (x1 ** 2 + y1 ** 2) * (y2 - y3) + (x2 ** 2 + y2 ** 2) * (y3 - y1)
    + (x3 ** 2 + y3 ** 2) * (y1 - y2)
    B = (x1 ** 2 + y1 ** 2) * (x3 - x2) + (x2 ** 2 + y2 ** 2) * (x1 - x3)
    + (x3 ** 2 + y3 ** 2) * (x2 - x1)
    C = 2 * (x1 * (y2 - y3) - y1 * (x2 - x3) + x2 * y3 - y2 * x3)
```

```
    D = (x1 ** 2 + y1 ** 2) * (x2 * y3 - y2 * x3)
    + (x2 ** 2 + y2 ** 2) * (x3 * y1 - y3 * x1)
    + (x3 ** 2 + y3 ** 2) * (x1 * y2 - y1 * x2)
    E = (x1 * (y2 - y3) - y1 * (x2 - x3)) * (x2 * y3 - y2 * x3)
    + (x2 * (y3 - y1) - y2 * (x3 - x1)) * (x3 * y1 - y3 * x1)
    + (x3 * (y1 - y2) - y3 * (x1 - x2)) * (x1 * y2 - y1 * x2)
    F = (x1 * (y2 - y3) - y1 * (x2 - x3)) ** 2 + (x2 * (y3 - y1)
    - y2 * (x3 - x1)) ** 2 + (x3 * (y1 - y2) - y3 * (x1 - x2)) ** 2

    h = (D + E) / (2 * F)
    k = (A + B) / (2 * C)

    return h, k

def randNCL(points):

    # randomiziramo točke
    random.shuffle(points)

    for i in range(len(points)-2):
        x1, y1 = points[i]
        x2, y2 = points[i+1]
        x3, y3 = points[i+2]

        # preko determinante provjerimo jesu li kolinearne
        det = x1*(y2-y3) - y1*(x2-x3) + x2*y3 - y2*x3
        if det != 0:
            return x1, y1, x2, y2, x3, y3

    return None

cap = cv2.VideoCapture('masafs.mp4')

tracker = cv2.TrackerCSRT_create()
# učitamo prvi okvir i označimo masu
ret, frame = cap.read()
bbox = cv2.selectROI(frame, False)
tracker.init(frame, bbox)

positions = []
```

```
# prođemo cijeli video
while True:

    ret, frame = cap.read()

    # break kad nema okvira
    if not ret:
        break
    # ažuriramo tracker u novom okviru
    ok, bbox = tracker.update(frame)

    if ok:
        x, y, w, h = [int(i) for i in bbox]
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        position = (x + w/2, y + h/2)  # centar tracking box-a
        positions.append(position)

    cv2.imshow('frame', frame)

    # q za break
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()

graphx=[]
graphy=[]
for i in range(len(positions)):
    graphx.append(positions[i][0])
    graphy.append(-positions[i][1])

plt.scatter(graphx, graphy, s=20, color='green', edgecolor='black',
linewidth=0.5)
plt.xlim(min(graphx) - 250, max(graphx) + 250)
plt.xlabel("Horizontalni pomak (pixel)", fontsize=12)
plt.ylabel("Vertikalni pomak (pixel)", fontsize=12)
plt.show()


cap = cv2.VideoCapture('masafs.mp4')
```

```
if not cap.isOpened():
    print("Error: Could not open video.")
    exit()

timestamps = []

while True:
    ret, frame = cap.read()

    if not ret:
        break

    timestamp = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0
    timestamps.append(timestamp)

timestamps.pop()
cap.release()

gy=[]
for i in range(len(graphy)):
    gy.append(graphy[i] + 709.5)

plt.figure(figsize=(10, 5))
plt.scatter(timestamps, gy, s=20, color='green', edgecolor='black',
linewidth=0.5)
plt.xlabel("Vrijeme (sekunda)", fontsize=12)
plt.ylabel("Vertikalni pomak (pixel)", fontsize=12)
plt.show()

import sys; sys.path.append('../')
from PDE_FIND_CM import *

#funkcija koja je potrebna kasnije kako bi algoritam funkcionirao
def trim(u, d):
    ind = np.arange(0, len(u))
    u_trim = np.delete(u, np.concatenate((ind[:d],ind[-d:])))
    return u_trim


gy=[]
```

```
for i in range(len(graphy)):
    gy.append(graphy[i] + 709)
t=0

m = 1
n = len(positions)

u = [gy]
u = np.array(u)
t = np.array(timestamps)
dt = t[1]-t[0]

un = u
ut = np.zeros((m,n), dtype=np.complex64)
utt = np.zeros((m,n), dtype=np.complex64)

for i in range(m):
    ut[i,:] = FiniteDiff(u[i,:], dt, 1)
    # koristi se kasnije kod funkcije TrainSTRidge
    utt[i,:] = FiniteDiff(u[i,:], dt,2)

ut = np.reshape(ut, (n*m,1), order='F')
utt = np.reshape(utt, (n*m,1), order='F')

X_ders = np.hstack([np.ones((n*m,1)),ut])
X_data = np.hstack([np.reshape(u, (n*m,1), order='F')])
derivatives_description = ['','u_t']

X, descr = build_Theta(X_data, X_ders, derivatives_description,
P=2, data_description = ['u'])
descr

w = TrainSTRidge(X,utt,100,10)
print("PDJ dobivena pomoću STRidge:")
print_pde(w, descr)

# učitamo video
cap = cv2.VideoCapture('fsmat.webm')

tracker = cv2.TrackerCSRT_create()
times = []
```

```
# učitamo prvi okvir i označimo masu
ret, frame = cap.read()
bbox = cv2.selectROI(frame, False)
tracker.init(frame, bbox)

positions = []

# prođemo cijeli video
while True:

    ret, frame = cap.read()

    if not ret:
        break

    ok, bbox = tracker.update(frame)

    if ok:
        x, y, w, h = [int(i) for i in bbox]
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        position = (x + w/2, y + h/2)  # centar tracking box-a
        positions.append(position)

    cv2.imshow('frame', frame)

    # q za break
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()

graphx=[]
graphy=[]
for i in range(len(positions)):
    graphx.append(positions[i][0])
    graphy.append(-positions[i][1])


plt.scatter(graphx, graphy, s=20, color='green', edgecolor='black',
linewidth=0.5)
```

```
plt.xlabel("Horizontalni pomak (pixel)",fontsize=12)
plt.ylabel("Vertikalni pomak (pixel)", fontsize=12)
plt.ylim(min(graphy) - 10, max(graphy) + 30)
plt.xlim(min(graphx) - 200, max(graphx) + 200)
plt.show()

cap = cv2.VideoCapture('fsmat.webm')

if not cap.isOpened():
    print("Error: Could not open video.")
    exit()

timestamps = []

while True:
    ret, frame = cap.read()

    if not ret:
        break
    timestamp = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0
    timestamps.append(timestamp)

timestamps.pop()
cap.release()


x1, y1, x2, y2, x3, y3 = randNCL(positions)

# izračunamo poziciju uporišta
h,k = centar(x1, y1, x2, y2, x3, y3)

# izračunamo ravnotežni položaj
ref_x,ref_y = ravnotezniPolozaj(graphx,graphy)

# računamo kut za svaki okvir
theta = []
for x, y in zip(graphx, graphy):
    # x i y pomak od ravnotežnog položaja
    dx = x - ref_x
    dy = (-y) - h
```

```
    # koristimo arctg kako bi dobili thetu
    angle = np.arctan2(dy, dx)
    angle_degrees = np.degrees(angle)
    theta.append(angle_degrees)

# transformacije polja za potrebe crtanja grafa
frames = [i for i in range(len(theta))]
thetacp = theta.copy()
for i in range(len(thetacp)):
    thetacp[i] -= 90


plt.figure(figsize=(10, 5))
plt.scatter(timestamps, thetacp, s=20, color='green', edgecolor='black',
linewidth=0.5)
plt.xlabel("Vrijeme (sekunda)", fontsize=12)
plt.ylabel("Kut otklona $\\theta$ ($^\circ$)", fontsize=12)
plt.show()

import sys; sys.path.append('../')
from PDE_FIND_CM import *

def trim(u, d):
    ind = np.arange(0, len(u))
    u_trim = np.delete(u, np.concatenate((ind[:d],ind[-d:])))
    return u_trim


m = 1
n = len(positions)

podatci = []
run = 10
width = 5 # sirina prozora kod PolyDiff funkcije

temp = thetacp
temp = np.copy(thetacp)
for i in range(len(temp)):
    temp[i] = temp[i] + 0.3

u = [temp]
```

```
u = np.array(u)
t = np.array(timestamps)

un = u
unt = np.zeros((m,n-2*width), dtype=np.complex64)
untt = np.zeros((m,n-2*width), dtype=np.complex64)
ones = np.ones((m,n), dtype=np.complex64)

for i in range(m):
    unt[i,:] = PolyDiff(un[i,:], t, diff=2, width=width)[:,0]
    untt[i,:] = PolyDiff(un[i,:], t, diff=2, width=width)[:,1]

un = trim(un[0,:], width)
ones = trim(ones[0,:], width)

unt = np.reshape(unt, (-1,1), order='F')
untt = np.reshape(untt, (-1,1), order='F')

X_ders = np.hstack([np.ones((n*m-2*width,1)), unt])
X_data = np.hstack([np.reshape(un, (-1,1), order='F'),])
derivatives_description = ['','u_t']

X, descr = build_Theta(X_data, X_ders, derivatives_description,
P=2, data_description = ['u'])

w = TrainSTRidge(X,untt,100,10)
podatci.append(w)

for i in range(len(podatci)):
    print("PDJ dobivena pomoću STRidge:")
    print_pde(podatci[i], descr)
```

Literatura

[1] Jung, A. ”Machine Learning: The Basics.” Springer, Singapore, 2022.

[2] Goodfellow, I.; Bengio, Y.; Courville, A. ”Deep Learning.” MIT Press, 2016. (Chapter 8 - Optimization for Training Deep Models)

[3] Ruder, S. ”An Overview of Gradient Descent Optimization Algorithms.” arXiv preprint arXiv:1609.04747, 2016.

[4] Schmidhuber, J. ”Deep learning in neural networks: An overview.” Neural Networks, 61, 85-117, 2015.

[5] Dragušica, V. ”Determining the Dynamics of Physical Systems Using Machine Learning.” Works - Department of Physics, Faculty of Science, Bijenička 32, Zagreb, 2023. Supervisor: Assoc. Prof. Dr. Davor Horvatić.

[6] Rudy, S. H.; Brunton, S. L.; Proctor, J. L.; Kutz, J. N. ”Data-driven discovery of partial differential equations.” arXiv:1609.06401v1 [nlin.PS], 2016.

[7] Kopecky, K. A. (2007). ”ECO 613/614 Fall 2007: Numerical Differentiation [Lecture notes].

[8] Brunton, S. L.; Proctor, J. L.; Kutz, J. N. ”Discovering governing equations from data by sparse identification of nonlinear dynamical systems.” Proceedings of the National Academy of Sciences, 113, 3932, 2016.

[9] https://github.com/VinkoGitHub/Samostalni-seminar (pristupljeno 1.8.2023.)