
Физички поткрепљене неуронске мреже - Практикум

Издање јрво

Милош Ивановић

2022.

Милош Ивановић

**Физички поткрепљене неуронске мреже
Практикум**

Рецензенти

проф. др

доц. др

Издавач

Природно-математички факултет Крагујевац
Радоја Домановића 12
Крагујевац

Штампа

Графички атеље Сквер, Крагујевац

Тираж

100 примерака

ISBN XXX-XX-XXXX-XXX-X

Садржај

1 Увод	3
1.1 Кратка историја методе	3
1.2 Физички поткрепљене неуронске мреже	4
1.3 Пример конструкције функције губитка	7
2 Провођење топлоте	9
2.1 Једначина провођења топлоте	9
2.2 Једнодимензиони директни проблем	10
2.3 Једнодимензиони инверзни проблем	17
2.4 Стефанов проблем у моделима топљења	18
3 Механичке осцилације	27
3.1 Осцилатор	27
3.2 Имплементација	33
3.3 Решавање директног проблема	36
3.4 Инверзни проблем	39
4 Хидрологија	45
4.1 Увод	45
4.2 Филтрација подземних вода	45
4.3 Пропагација поплавног таласа у отвореном каналу	53
5 Акустика	61
5.1 Увод	61
5.2 Решавање на домену облика квадрата	62
5.3 Решавање на домену облика квадрата са шупљином	67
5.4 Дводимензиони проблем са сочивом	71
6 Моделовање мишћа	79

6.1	Увод	79
6.2	Изометријски случај	81
7	Моделовање производње соларних електрана	87
7.1	Увод	87
7.2	Основне једначине модела	96
7.3	Пример прорачуна производње	97
7.4	Резултати	103
Литература		105

Предговор

Физички поткрепљене неуронске мреже (*Physics-Informed Neural Networks* - PINN) или на српском језику ФПНМ су тип универзалних функција апроксимације које могу да се тренирају тако да усвоје познавање било ког физичког закона који се може описати парцијалним диференцијалним једначинама, а који важи у одређеном просторно-временском домену. Обука овог типа неуронских мрежа се поставља на тај начин да поштује симетрије, инваријантност или одржавање принципа који почивају на физичким законима исказаним у облику парцијалних диференцијалних једначина.

Обичне дубоке неуронске мреже нису довољно робусне у већини случајева када се везују за математички исказане законе у биологији, механици, електротехници, итд. С друге стране, код ФПНМ мрежа, претходно знање општих физичких закона се у процесу тренирања неуронских мрежа поставља као регуларизациони агент који ограничава простор дозвољених решења, што повећава тачност апроксимиране функције. На овај начин, уградњивањем физичких законова описаних парцијалним диференцијалним једначинама у неуронску мрежу добијамо побољшање, што олакшава алгоритму учења да добије што тачније решење и да добро генерализује, чак и са веома малом количином тзв. колокационих тачака.

Традиционално, за решавање (система) диференцијалних једначина нумеричким путем, већи деценијама се користе класичне методе, као што су метода коначних разлика (енг. *Finite Difference Method*), метода коначних елемената (енг. *Finite Element Method*), метода коначних запремина (енг. *Finite Volume Method*), итд. Упркос њиховој популарности, све ове методе имају и одређена ограничења. Пре свих, ту су два проблема. Први је велика рачунска комплексност, а други асимилација екстерних извора података добијених мерењем на моделованом систему. Такође, решавање инверзних проблема тј. претрага непознатих или несигурних параметара класичним методама постаје готово немогућа за иоле комплексније проблеме из инжењерске праксе. ФПНМ мреже нуде једноставне и флексибилне механизме који све ове проблеме узимају у обзир од почетка, самим својим концептом.

Овај практикум је настало из мотивације да се овом релативно новом концепту, који је први

пут публикован 2017. године од стране Raissi et al.¹ посвети дужна пажња на српском говорном подручју и тиме у одређеној мери сниси баријера за улазак у ову „хибридну” област на граници између нумеричког моделовања и машинског учења. Проближавањем ове две области би се, по мишљењу аутора, пуно тога добило. Физичари и инжењери би добили флексибилан алат за приближно решавање директних и инверзних проблема који им омогућава да брзо провере хипотезе, интегришу мерења и идентификују параметре. С друге стране, истраживачима из поља машинског учења се нуди начин да већ постојећа знања о физичким појавама и инжењерским законитостима на релативно једноставан начин интегришу у своје, до сада чисто статистичке, моделе и тиме им побољшају поузданост и могућност предвиђања.

Збирка се састоји из детаљно постављених и решених примера из праксе аутора и сарадника. Дотакли смо проблеме из области класичне механике, провођења топлоте и неке једноставније хидролошке проблеме. Такође предлажемо начине за интеграцију мерења и оптимизацију хипер-параметара попут броја слојева и неурона у неуронској мрежи, активационих функција итд.

Аутор се захвальује сарадницима Центра за рачунарско моделовање и оптимизацију (ЦЕРАМО) Природно-математичког факултета у Крагујевцу². Пре свега, захвалност дугујем др Бобану Стојановићу, редовном професору ПМФ-а, руководиоцу више пројеката у којима је примену нашла метода која је тема овог практикума. Млади колега Филип Бојовић до-принео је успешној имплементацији проблема *Пропагација поплавног таласа у отвореном каналу*, а колегиница Бранка Андријевић у проблему *Филтрација подземних вода* и *Моделовање производње соларних панела*. Богдан Милићевић је значајно допринео решавању проблема моделовања мишића помоћу ФПНМ, а одговоран је и за српски превод назива ове методе, на чemu му се захваљујем. На крају, али не и најмање важно, захвалност дугујем колеги Владимиру Миловановићу са Факултета инжењерских наука у Крагујевцу, који ми је указао да савремени наставни материјал треба да буде отворен и да је данас веома важно да буде доступан и на вебу и као класично штампано издање.

У Крагујевцу,
децембар 2022. године

– аутор

¹ <https://maziarraissi.github.io/PINNs/>

² <https://www.pmf.kg.ac.rs/>

1.1 Кратка историја методе

Током последње деценије се убрзано развијају различите методе дубоког учења решавање различитих врста проблема из области машинског учења, као што су препознавање слике, препознавање говора, обрада природног језика (*Natural Language Processing - NLP*), претраживање, системи за препоруке, биоинформатика, итд. Међутим, класично надгледано дубоко учење као метода није погодно за решавање баш свих врста проблема, без обзира на довољну количину доступних података који описују понашање моделованог система. На пример, проблеми описани кроз линеарне и нелинеарне једначине и системе једначина никад и нису били у фокусу дубоког учења. Класичне нумеричке методе и даље држе апсолутни примат у нумеричком решавању парцијалних диференцијалних једначина. Методе као што су коначне разлике (*Finite Difference Method*), коначне запремине (*Finite Volume Method*) и коначни елементи (*Finite Element Method*) се и даље сматрају најсавременијим методама због њихове робусности, ефикасности и могућности примене у широком спектру проблема.

Са друге стране, решавање нелинеарних инверзних проблема класичним нумеричким методама захтева прорачуне који су изузетно рачунарски и временски захтевни. Претрага оптималних параметра модела укључује итеративни поступак који често укључује десетине и стотине пролаза директне симулације. Због тога се тачност често жртвује за ефикасност, те ове врсте претрага често нису искрпне и укључују тзв. мета-хеуристике, без гаранције да ће се пронаћи најбоље (оптимално) решење. Примери из праксе аутора и сарадника најчешће користе генетске алгоритме (*Genetic Algorithm - GA*), спроводе оптимизацију базирану на симулацији (*Simulation Based Optimization*) и захтевају употребу више стотина процесора да би се до решења (које није гарантовано оптимално) дошло у иоле разумном временском року, према Ivanovic *et al.* [ISS+15], Simic *et al.* [SSI19] и Ivanovic and Simic [IS22]. Поред кардиналоних потешкоћа при решавању инверзних проблема, очигледни су још и следећи недостаци класичних нумеричких метода:

- Није могуће на лак начин у модел укључити **податке са шумом**.

- Генерисање прорачунске мреже остаје сложена мануелна операција, често подложна грешкама.
- Проблеме који укључују **већи број димензија** није могуће решити у реалним временским оквирима рачунарским ресурсима којима тренутно располажемо.

Да би се елиминисали ови недостаци класичних нумеричких метода, развијена је нова метода дубоког учења за решавање парцијалних диференцијалних једначина. Та метода, под именом **Физички поткрепљене неуронске мреже** (*Physics Informed Neural Networks* - PINN), користи се за решавање проблема надгледаног учења уз поштовање било ког закона физике описаног општим нелинеарним парцијалним диференцијалним једначинама Raissi *et al.* [RPK19].

Главна иновација Физички поткрепљених неуронских мрежа (ФПНМ) у поређењу са класичним дубоким неуронским мрежама је увођење функције губитка која кодира основне једначине физичких законова, узима излаз дубоке мреже, која се зове апраксиматор, и израчунава вредност губитка Markidis [Mar21]. Функција губитка која се односи на диференцијалну једначину се минимизира обуком апраксиматорске неуронске мреже, где се диференцијални оператори примењују коришћењем аутоматске диференцијације. Аутоматска диференцијација је предуслов да би се уопште обавило тренирање пропагацијом уназад и поседују је све библиотеке за машинско учење, као што су *Tensorflow*, *PyTorch*, *Theano*, и друге.

1.2 Физички поткрепљене неуронске мреже

Физички поткрепљена неуронска мрежа (у даљем тексту ФПНМ) је техника машинског учења која може се користити за апраксимацију решења парцијалне диференцијалне једначине. Парцијалне диференцијалне једначине са одговарајућим почетним и граничним условима могу се изразити у општем облику као:

$$\begin{aligned} u_t + \mathcal{N}[u] &= 0, \quad X \in \Omega, \quad t \in [0, T], \\ u(X, 0) &= h(X), \quad X \in \Omega, \\ u(X, t) &= g(X, t), \quad X \in \Omega_g, \quad t \in [0, T]. \end{aligned} \tag{1.1}$$

Овде је \mathcal{N} диференцијални оператор, $X \in \Omega \subseteq R^d$ и $t \in R$ представљају просторне и временске координате, респективно, док је $\Omega \subseteq R$ целокупни домен проблема. $\Omega_g \subseteq \Omega$ представља рачунски домен граничних услова, $u(X, t)$ је решење парцијалне диференцијалне једначине са почетним условом $h(X)$ и граничним условом $g(X, t)$. Оваква формулатија се такође може применити и на парцијалне диференцијалне једначине вишег реда, пошто се једначине вишег реда могу написати и у облику система једначина првог реда.

У оригиналној формулацији Raissi *et al.* [RPK19], ФПНМ се састоји од две подмреже:

- апраксиматор мреже и
- резидуалне мреже.

Апраксиматор мрежа прима улаз (X, t) , пролази кроз процес обуке и као излаз даје приближно решење $\hat{u}(X, t)$ парцијалне диференцијалне једначине. Мрежа апраксиматора се тренира на мрежи тачака, тзв. **колокационих тачака**, узоркованих из домена проблема.

Тежине и пристрасности апраксиматор мреже су параметри који се могу тренирати минимизирањем композитне функције губитка у следећем облику:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0 + \mathcal{L}_b, \quad (1.2)$$

где су

$$\begin{aligned} \mathcal{L}_r &= \frac{1}{N_r} \sum_{i=1}^{N_r} |u(X^i, t^i) + \mathcal{N}[u(X^i, t^i)]|^2, \\ \mathcal{L}_0 &= \frac{1}{N_0} \sum_{i=1}^{N_0} |u(X^i, t^i) - h^i|^2, \\ \mathcal{L}_b &= \frac{1}{N_b} \sum_{i=1}^{N_b} |u(X^i, t^i) - g^i|^2. \end{aligned} \quad (1.3)$$

Овде, \mathcal{L}_r , \mathcal{L}_0 и \mathcal{L}_b представљају резидуале основне диференцијалне једначине, почетних и граничних услова, респективно. Поред тога, N_r , N_0 и N_b су бројеви колокационих тачака домена проблема, домена почетних и граничних услова, респективно. Ови резидуали се израчунају компонентом ФПНМ модела који се не обучава, а зове се **резидуална мрежа**. Да би се израчунат губитак \mathcal{L}_r , ФПНМ захтева изводе излаза у односу на улазе. Тада се постиже тзв. аутоматском диференцијацијом.

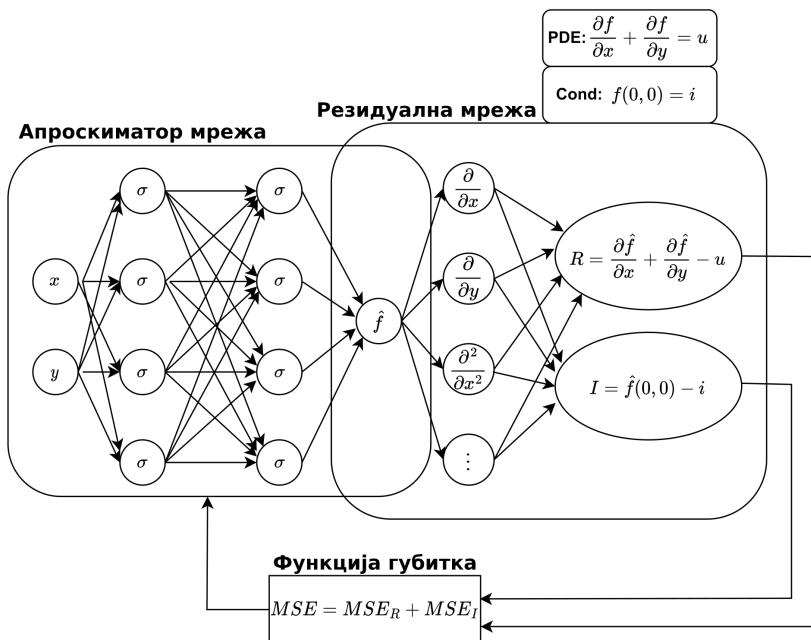
Аутоматска диференцијација је кључни покретач развоја ФПНМ-а и кључни је елемент који разликује ФПНМ од сличних настојања 90-их година прошлог века. На пример, Psycho-gios and Ungar [PU92] и Lagaris *et al.* [LLF98] су се ослањали на мануелно извођење правила пропагације уназад. Данас се рачуна на аутоматску диференцијацију која је имплементирана у већини оквира за дубоко учење, као што су [Tensorflow](#)³ и [PyTorch](#)⁴. На овај начин избегавамо нумеричку дискретизацију током рачунања извода свих редова у простор-времену.

Шема типичног ФПНМ-а је приказана на [Сл.1.1](#) на којој је једноставна парцијална диференцијална једначина $\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} = 0$ искоришћена као пример. Као што је приказано, мрежа апраксиматора се користи за апраксимацију решења $u(X, t)$, које затим иде на резидуалну мрежу за израчунање функције губитка диференцијалне једначине \mathcal{L}_r , губитка граничних услова \mathcal{L}_b , и губитка почетних услова \mathcal{L}_0 . Тежине и пристрасности апраксиматорске мреже обучени су коришћењем прилагођене функције губитка која се састоји од резидуала \mathcal{L}_r , \mathcal{L}_0 , и \mathcal{L}_b кроз технику градијента спуштања засновану на пропагацији уназад.

У наредној секцији [Пример конструкције функције губитка](#) (страна 7) описаћемо како би изгледала конструкција композитне функције губитка за логистичку једначину.

³ <https://www.tensorflow.org/>

⁴ <https://pytorch.org/>



Слика1.1: Архитектура ФПНМ-а и стандардна петља за обуку ФПНМ-а конструисана за решавање једноставне парцијалне диференцијалне једначине, где PDE и $Cons$ означавају једначине, док R и I представљају њихове резидуале. Мрежа апроксиматора је подвргнута процесу обуке и даје приближно решење. Резидуална мрежа је део ФПНМ-а који се не обучава и који је способан да израчуна изводе излаза апроксиматорске мреже у односу на улазе, што резултира композитном функцијом губитка, означеном са MSE .

1.3 Пример конструкције функције губитка

Резоновање из претходног одељка *Физички поткрепљене неуронске мреже* (страна 4) ћемо поткрепити једноставним примером. Започнимо прво тако што ћемо узети неку једноставну диференцијалну једначину. Изабраћемо *логистичку једначину*⁵ (*Logistic Equation*) која представља модел раста популације. Та једначина гласи:

$$\frac{df}{dt} = R \cdot t(1 - t) \quad (1.4)$$

Функција $f(t)$ представља стопу раста популације током времена t , док параметар R одређује максималну стопу раста. Како бисмо од фамилије кривих које задовољавају решење ове обичне диференцијалне једначине изабрали једну криву као решење, морамо поставити гранични, тј. почетни услов. Нека то буде:

$$f(t = 0) = \frac{1}{2}. \quad (1.5)$$

Како је аналитичко решење ове једначине познато, поређењем са њим можемо проверити тачност ФПНМ методе. Тиме ћемо почети да разоткривамо потенцијал примене и на комплексније обичне и парцијалне диференцијалне једначине. Да се подсетимо, ФПНМ су засноване на две фундаменталне особине неуронских мрежа:

- Формално је показано да су **неуронске мреже универзалне функције апраксимације**, тако да неуронска мрежа може да апраксимира било коју функцију, а самим тим и решење за нашу логистичку једначину.
- Једноставно је и јефтино израчунати изводе било ког реда излаза из неуронске мреже за било који дати улаз коришћењем **автоматске диференцијације**.

Дакле, као што је речено, можемо да конструишимо функцију губитка тако да, када се минимизује, диференцијална једначина буде задовољена:

$$\mathcal{L}_r = \frac{df_{NN}(t)}{dt} - R \cdot t(1 - t) = 0, \quad (1.6)$$

где је $f_{NN}(t)$ излаз неуронске мреже са једним улазом чији се извод израчунава аутоматским диференцирањем. Одмах видимо да, уколико излаз из мреже задовољава (1.6), заправо се та једначина решава. Да би се израчунао стварни допринос функцији губитка који се добија из диференцијалне једначине, потребно је специфицирати скуп тзв. колокационих тачака у домену проблема и проценити средњу квадратну грешку (*Mean Squared Error - MSE*) или неку другу функцију губитка:

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} \left(\frac{df_{NN}}{dt} \Big|_{t_i} - Rt_j(1 - t_i) \right)^2, \quad (1.7)$$

где је N_r број колокационих тачака t_j у којима се рачуна функција губитка. Губитак заснован само на резидуалима не осигурува јединствено решење, па стога укључујемо и гранични

⁵ https://en.wikipedia.org/wiki/Logistic_function

услов, тако што га додајемо функцији губитка на исти начин као у једначини (1.7):

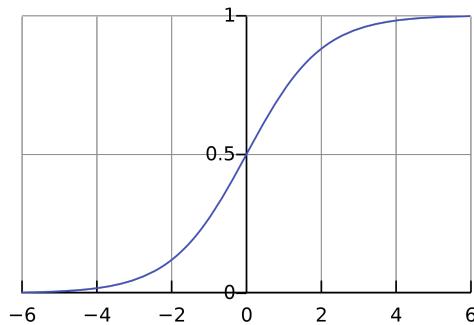
$$\mathcal{L}_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} \left(f_{NN}(t)|_{t_i} - \frac{1}{2} \right)^2, \quad t_i \approx 0. \quad (1.8)$$

Сада имамо оба елемента да дефинишемо укупну функцију губитка \mathcal{L} :

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0. \quad (1.9)$$

Током оптимизације (тј. тренинга у терминологији неуронских мрежа), горњи израз се минимизује и излаз из мреже тренира да задовољи диференцијалну једначину и задат гранични услов, чиме се апроксимира коначно, јединствено решење диференцијалне једначине. Концепт ФПНМ је веома једноставан, и користећи идеју описану у претходном тексту, можемо додати више граничних услова, додати комплексније или решавати временски зависне вишдимензионалне проблеме користећи мрежу са вишеструким улазима.

Решење наше логистичке једначине је добро позната сигмоид функција приказана на Сл.1.2.



Слика1.2: Сигмоидна функција која се добија као решење једначине (1.4) са почетним условом (1.5)

1.3.1 Начин излагања примера

Излагање ћемо наставити конкретним примерима. Свако поглавље практикума (Провођење топлоте, Осцилације, ...) бави се посебном феноменологијом и садржи по један или више директних или инверзних решених проблема. Сваки пример је поткрепљен теоријском по-задином, припадајућим програмским кодом који имплементира ФПНМ решење, као и анализом тачности и ефикасности ФПНМ решења у односу на аналитичка решења, уколико постоје, или решења добијена класичним нумериčким методама.

Ваља напоменути да примери нису сотирани по тежини, већ искључиво по феноменологији која се моделује. Ако ипак треба да препоручимо читаоцу одакле да крене са практичним радом, рецимо да [Механичке осцилације](#) (страна 27) и [Једнодимензиони директни проблем](#) (страна 10) представљају добру основу.

Провођење топлоте

2.1 Једначина провођења топлоте

У физици, хемији и инжињерству, феномени преноса се баве механизмима преноса неке физичке величине са једне локације на другу. Три основа механизма преноса су провођење (дифузија), преношење (конвекција) и зрачење (радијација). Три основне величине које се третирају у феноменима преноса су:

- пренос топлоте,
- пренос масе и
- пренос количине кретања.

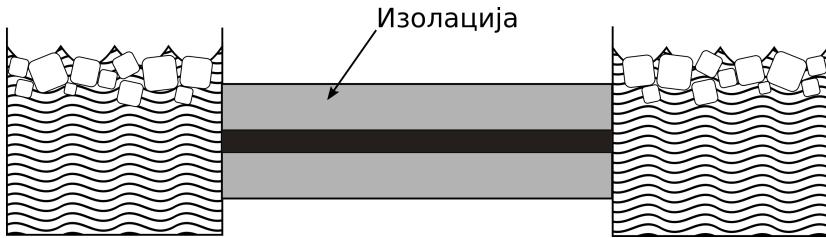
Све три величине преносе се на сличан начин. Ако говоримо о топлоти, Фуријеов закон опијује пропорционалност флукса топлоте и градијента температуре. Коефицијент пропорционалности назива се коефицијентом топлотне проводљивости. Феномен провођења топлоте у времену у једној просторној димензији дефинисан је следећом једначином:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial t^2}, \quad 0 < x < L, t \geq 0 \quad (2.1)$$

где је $u(x, t)$ температура, а α константан коефицијент. У наредном одељку *Једнодимензиона директни проблем* (страна 10) дефинисаћемо једноставан проблем, решити га помоћу ФПНМ и упоредити добијено решење са аналитичким изразом.

2.2 Једнодимензиони директни проблем

Танак штап од хомогеног материјала је окружен изолацијом, тако да се промене температуре у штапу дешавају само као последица размене топлоте ка крајевима штапа и провођења топлоте дуж штапа. Штап је јединичне дужине. Оба краја су изложена мешавини воде и леда температуре 0. Почетна температура на растојању x од левог краја штапа је $\sin(\pi x)$, као што се види на Сл.2.1.



Слика2.1: Експериментална поставка проблема провођења топлоте дуж штапа. На крајевима штапа налази се мешавина воде и леда. Штап је изолован од утицаја спољашње средине.

Ради поређења, показаћемо сада како се овај релативно једноставан проблем формулише помоћу класичне методе коначних разлика, а затим ћемо решити директни и инверзни проблем описане поставке ФПНМ методом.

2.2.1 Решавање методом коначних разлика (МКР)

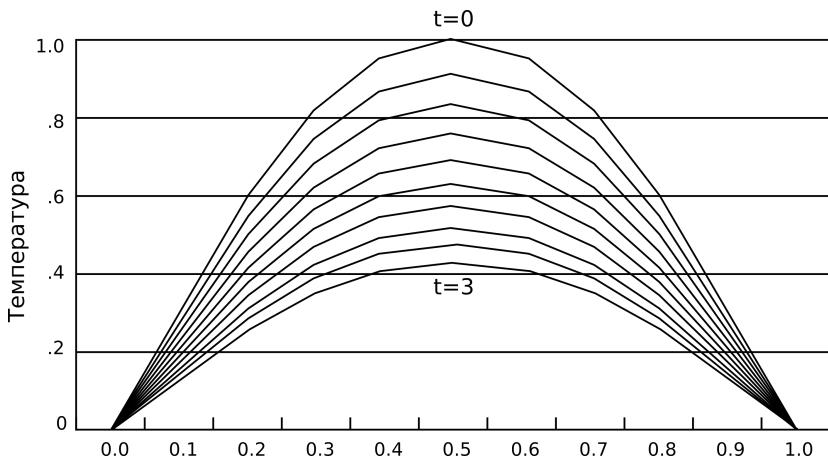
Парцијална диференцијална једначина (2.1) моделује температуру у било којој тачки штапа у било ком временском тренутку према Recktenwald [Rec04]. Ова једначина се решава методом коначних разлика, која даје апроксимацију решења за распоред температуре, примењујући просторну и временску дискретизацију. Програмска имплементација решења чува температуру сваке тачке дискретизације у дводимензионој матрици. Сваки ред садржи температурну дистрибуцију штапа у неком тренутку времена. Штап је подељен на n делова дужине h , па стога сваки ред има $n + 1$ елемената. Начелно, што је веће n , мања је грешка апроксимације. Време од 0 до T је подељено у m дискретних интервала дужине k , па стога матрица има $m + 1$ редова, као што је приказано на Сл.2.3.

Свака тачка $u_{i,j}$ представља елемент матрице који садржи температуру на позицији $i \cdot h$, у тренутку $j \cdot k$. На крајевима штапа је температура увек нула. У почетном тренутку, температура у тачки x је, као што је већ речено, $\sin(\pi x)$. Алгоритам иде корак по корак кроз време, користи вредности из тренутка j да би израчунато вредности у тренутку $j + 1$. Формула која представља варијанту апроксимације FTCS (Forward Time Centered Space) као у Recktenwald [Rec04] се овде даје без извођења и гласи:

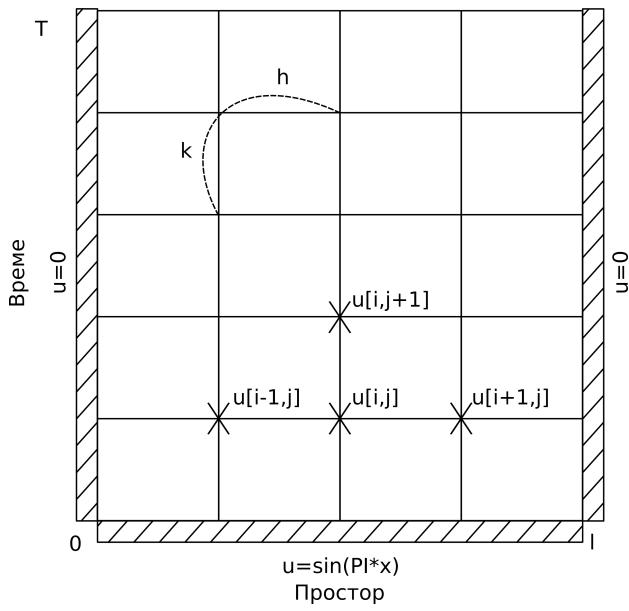
$$u_{i,j+1} = R \cdot u_{i-1,j} + (1 - 2R) \cdot u_{i,j} + R \cdot u_{i+1,j}, \quad (2.2)$$

где је

$$R = \alpha \frac{k}{h^2}.$$



Слика2.2: Како време тече, штап се хлади. Метода коначних разлика омогућава израчунавање температуре у фиксном броју тачака у равномерним временским интервалима. Смањење просторног и временског корака углавном доводи до прецизнијег решења.



Слика2.3: Дискретизација једначине провођења топлоте методом коначних разлика

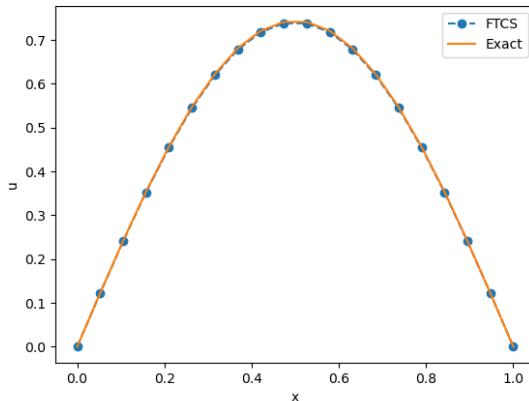
Целокупна анализа различитих експлицитних и имплицитних метода дата је у поменутој референци, а кључни део кода у програмском језику Пајтон имплементиран је на следећи начин:

```

1 def heatFTCS(nt=10, nx=20, alpha=0.3, L=1, tmax=0.1):
2     h = L / (nx - 1)
3     k = tmax / (nt - 1)
4     r = alpha * k / h**2
5
6     x = np.linspace(0, L, nx)
7     t = np.linspace(0, tmax, nt)
8     U = np.zeros((nx, nt))
9
10    # Почетни услов
11    U[:, 0] = np.sin(np.pi * x / L)
12
13    # Главна петља за МКР
14    for m in range(1, nt):
15        for i in range(1, nx-1):
16            U[i, m] = r * U[i - 1, m - 1] + (1-2*r) * U[i, m-1] + r * U[i+1, m-
17            ↪1]
18
19    # Егзактно решење за поређење
20    ue = np.sin(np.pi * x / L) * \
         np.exp(-t[nt - 1] * alpha * (np.pi / L) * (np.pi / L))

```

Као што детаљно објашњава Recktenwald [Rec04], ако се МКР петља формулише експлицитно као што је то случај код FTCS технике, мора се пажљиво изабрати временски и просторни корак, како би нумеричка пропагација била „бржа” од физичке. Решење које се добија помоћу МКР шеме се може видети на Сл.2.4.



Слика2.4: Решење које се добија МКР методом користећи експлицитну FTCS технику у тренутку $t = 0.1s$

Овај проблем има и аналитичко решење, па је погодан за тестирање различитих нумеричких

метода. То решење гласи:

$$u(x, t) = \sin\left(\frac{\pi x}{L}\right) \cdot e^{-\frac{\alpha\pi^2}{L^2}t}. \quad (2.3)$$

или у нашем случају, када је $L = 1$:

$$u(x, t) = \sin(\pi x) \cdot e^{-\alpha\pi^2 t}.$$

Експлицитне технике попут FTCS не гарантују конзистентност решења коју гарантују имплицитне технике као што је BTCS (*Backward Time Centered Space*). MKP је устаљени приступ који за већину правилно дефинисаних просторних домена ради веома добро. За овако једноставну поставку као што је једнодимензионо провођење топлоте и када су сви параметри проблема познати (овде је то α), MKP је често оптимална метода. Међутим, код већине проблема из праксе то није случај. Хајде да размотримо како да овај проблем решимо користећи ФПНМ и директно упоредимо са MKP.

2.2.2 Решавање помоћу ФПНМ

Решење једначине Сл.2.1 са већ постављеним граничним и почетним условима:

$$\begin{aligned} u(x=0, t) &= u(x=1, t) = 0, \forall t \\ u(x, t=0) &= \sin(\pi x) \end{aligned} \quad (2.4)$$

потражићемо помоћу ФПНМ приступа. Иако је могуће да методе имплементирамо директно као Raissi *et al.* [RPK19] користећи оквир за дубоко учење као што је Tensorflow⁶, ипак ћемо искористити помоћ библиотека које додатно апстрахију ФПНМ ентитете и омогућавају кориснику да се фокусира на проблем који решава. Овај пример решићемо користећи библиотеку SCIANN⁷ аутора Haghhighat and Juanes [HJ21]. Поступак решавања објаснићемо директно кроз програмски код:

Листинг 2.1: ФПНМ - провођење топлоте

```

1 import numpy as np
2 import sciann as sn
3 from sciann.utils.math import diff, sign, sin, sqrt, exp
4 from numpy import pi
5
6 x = sn.Variable('x')
7 t = sn.Variable('t')
8 u = sn.Functional('u', [x,t], 3*[20], 'tanh')
9 alpha = 0.3
10
11 L1 = diff(u, t) - alpha * diff(u, x, order=2)
12
13 TOLX = 0.011
14 TOLT = 0.0011
15 C1 = (1-sign(t - TOLT)) * (u - sin(pi*x))

```

(наставак на следећој страни)

⁶ <https://www.tensorflow.org/>

⁷ <https://www.sciann.com/>

(настављено са претходне стране)

```

16 C2 = (1-sign(x - (0+TOLX))) * (u)
17 C3 = (1+sign(x - (1-TOLX))) * (u)
18
19 m = sn.SciModel([x, t], [L1, C1, C2, C3], 'mse', 'Adam')
20
21 x_data, t_data = np.meshgrid(
22     np.linspace(0, 1, 101),
23     np.linspace(0, 0.1, 101)
24 )
25
26 h = m.train([x_data, t_data], 4*['zero'], learning_rate=0.002, batch_size=256, ↴
27 epochs=500)
28
29 # Test
30 nx, nt = 20, 10
31 x_test, t_test = np.meshgrid(
32     np.linspace(0.01, 0.99, nx+1),
33     np.linspace(0.01, 0.1, nt+1)
34 )
u_pred = u.eval(m, [x_test, t_test])

```

Варијабле x и t се на почетку дефинишу на прописани начин. Основни појам који се користи у SCIANN библиотеци за апстракцију ФПНМ је функционал, који је овде означен са u , који као улаз узима x и t , има 3 скривена слоја са по 20 неурона и као активацију свих тих неурона узима функцију хиперболичког тангена. Први сабирац копозитне функције губитка произилази из саме диференцијалне једначине (2.1). Као што се види, за диференцирање се користи специјални оператор `diff()` из библиотеке:

```
L1 = diff(u, t) - alpha * diff(u, x, order=2)
```

Најзанимљивији и не баш тако очигледан је начин дефинисања почетног условия $C1$ и граничних услова $C2$ и $C3$:

```

C1 = (1-sign(t - TOLT)) * (u - sin(pi*x))
C2 = (1-sign(x - (0+TOLX))) * (u)
C3 = (1+sign(x - (1-TOLX))) * (u)

```

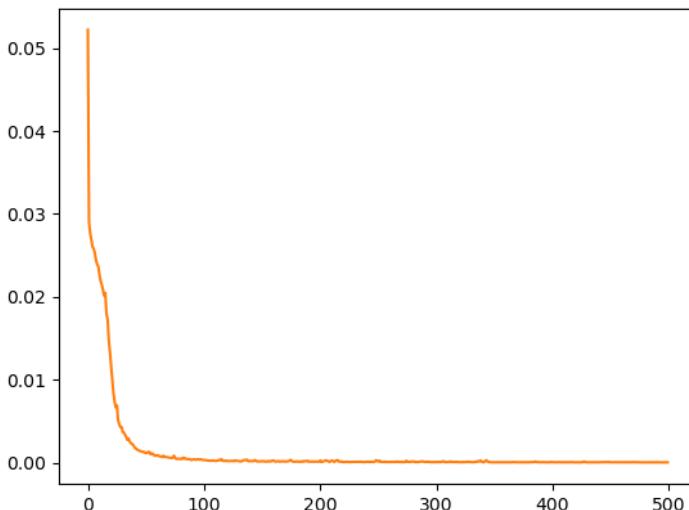
Овде је $C1$ једнак нули у свим тачкама узорковања осим за $t \leq TOLT$. Толеранције $TOLX$ и $TOLT$ су постављене тако да „хватају” прву/последњу врсту или колону колокационих тачака, у зависности шта је потребно. Уместо функције знака `sign()`, могу се користити и глаткије функције, као што је хиперболички тангенс. ФПНМ модел се формира помоћу `SciModel` конструктора који дефинише и тип функције губитка и алгоритам оптимизације, тј. обучавања:

```
m = sn.SciModel([x, t], [L1, C1, C2, C3], 'mse', 'Adam')
```

Обучавање модела се покреће методом `train()`, при чему се наводе следећи параметри:

- Скуп колокационих тачака за тренирање.** Овде је то правилна еквидистантна мрежа тачака по обе варијабле. Хоризонтала је простор, а вертикална време.
- Почетне вредности компоненти функције губитка.** Уобичајено је да се на почетку поставе на нуле.

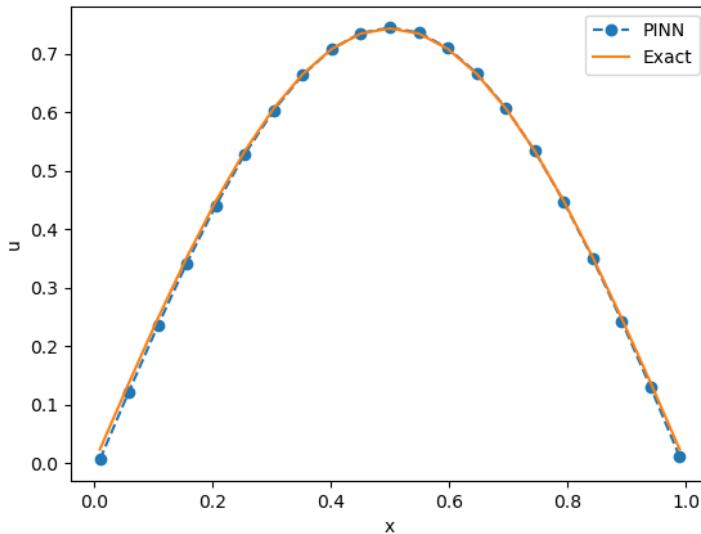
3. Стопа учења,
4. Величина batch-a. Треба имати на уму да ако је број тачака домена у којима се налазе гранични услови значајно мањи у односу на укупан број колокационих тачака, параметар оптимизације `batch_size` треба да буде подешен на велики број који гарантује доследну оптимизацију мини batch-а. У супротном, може да се деси да неки мини batch-еви не добију ниједну колокациону тачку која припада граничним условима и стога не генеришу тачан градијент за ажурирање методом градијентног спуста.
5. Број епоха.



Слика2.5: Историја обуке једнодимензионог модела провођења топлоте.

Ток обуке можемо да испратимо кроз стандардне *Tensorflow* објекте, као што је `h.history['loss']`, као што се види на [Сл.2.5](#). Пошто се заврши обука ФПНМ-а, можемо формирати тестни скуп тачака слично као што смо то учинили и са колокационим тачкама и проверити резултате предикције позивом методе `eval()` на објекту истренираног модела. Резултат поља температуре дуж штапа у тренутку $t = 0, 1$ и његово поређење са аналитичким решењем види се на [Сл.2.6](#).

Чисто практично гледано, **ФПНМ решење једноставног директног проблема као што је овај и не пружа никакве посебне предности у односу на класичну МКР методу**. Прво, решавање дуже траје и захтева упошљавање више рачунарских ресурса и зависности у виду додатних библиотека за тензорски рачун. Даље, спецификација почетних и граничних услова код ФПНМ има своје специфичности. Треће, неопходно је методом пробе и грешке подесити хипер-параметре модела, као што су: број скривених слојева, број неурона по слоју, активациона функција, брзина учења (*Learning Rate*) итд. Од избора хипер-параметара конвергенција решења може значајно да зависи.



Слика2.6: Полье температуре дуж штапа у тренутку $t = 0, 1$ добијено методом ФПНМ.

Са друге стране, за разлику од МКР и МКЕ (*Метода Коначних Елемената*), ФПНМ нам дозвољава да проблем дефинишемо чистим диференцијалним једначинама и произвољним граничним условима (Дирихлеови, Нојманови, периодични, скуп тачака). **Нема потребе за специфицирањем алгебарске везе између чворова** (тј. колокационих тачака у ФПНМ) и решавањем тако постављеног система једначина. Захваљујући овој чинjenци, било која нова физика у виду новог граничног услова или промена у самој диференцијалној једначини може да се изведе веома лако, омогућавајући брзу проверу хипотеза и израду прототипова.

Друго, док све класичне методе прорачун морају да изведу кроз временске кораке (*time stepping*), **ФПНМ омогућава брзу инференцију** на већ обученој мрежи за било који временски тренутак t постављен на улазу мреже. За неке примене у реалном времену где је брзина од кључног значаја, ово може да буде пресудно.

Треће, ФПНМ методолошки не разликује **директне проблеме** (у којима се решава позната диференцијална једначина) од **инверзних проблема**, код којих су неки од параметара непознати, али постоје додатни услови из којих се непознати параметри могу добити процесом тренинга. У наредној теми *Једнодимензиони инверзни проблем* (страна 17) демонстрираћемо један такав проблем.

2.3 Једнодимензиони инверзни проблем

Поставка инверзног проблема је потпуно иста као и у претходном поглављу *Једнодимензиони директни проблем* (страна 10), тј. описана је [Сл.2.1](#), једначином (2.1) и граничним условима (2.4). Међутим, овога пута нам параметар проблема α на почетку није познат и покушаћемо да га добијемо уз помоћ метода обучавања пропагацијом уназад. Наравно, чим смо увели нову непознату, морамо да уведемо и нови гранични услов. Рецимо, можемо да поставимо да је у једној тачки у неком временском тренутку, температура u одговарала некој нумеричкој вредности која се поклапа са аналитичким решењем (2.3). Рецимо, поставимо температуру на средини штапа у $x = 0,5$ у тренутку $t = 0,05$ на:

$$u(x = 0,5, t = 0,05) = 0,8623931,$$

и покушајмо да решимо проблем постављајући граничне услове на следећи начин:

Листинг 2.2: Проналажење непознатог параметра α

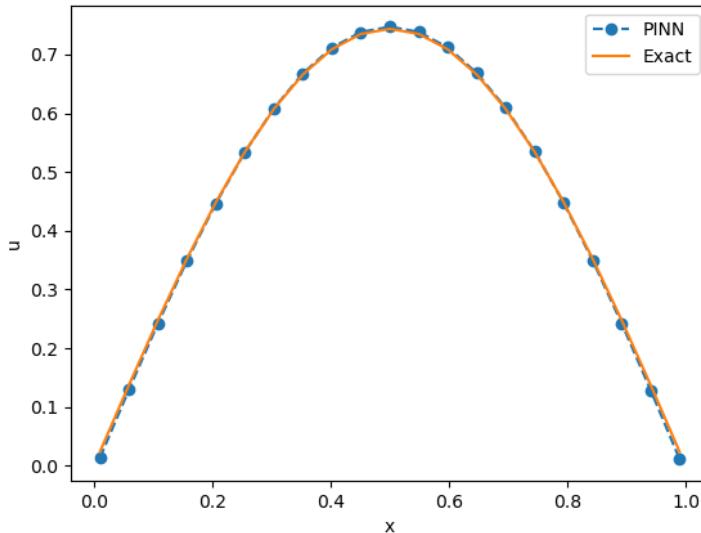
```

1 x = sn.Variable('x')
2 t = sn.Variable('t')
3 u = sn.Functional('u', [x,t], 3*[20], 'tanh')
4 alpha = sn.Parameter(0.5, inputs=[x,t], name="alpha")
5
6 L1 = diff(u, t) - alpha * diff(u, x, order=2)
7
8 TOL = 0.011
9 TOLT= 0.0011
10 C1 = (1-sign(t - TOLT)) * (u - sin(pi*x))
11 C2 = (1-sign(x - (0+TOL))) * (u)
12 C3 = (1+sign(x - (1-TOL))) * (u)
13 C4 = (1 + sign(t-0.049)) * (1 - sign(t-0.051)) * (1 + sign(x-0.49)) * (1 -
   ↳sign(x-0.51)) * (u-0.8623931)
14
15 m = sn.SciModel([x, t], [L1, C1, C2, C3, C4], 'mse', 'Adam')
16
17 x_data, t_data = np.meshgrid(
18     np.linspace(0, 1, 101),
19     np.linspace(0, 0.1, 101)
20 )
21
22 h = m.train([x_data, t_data], 5*['zero'], learning_rate=0.002, batch_size=512, ↳
   ↳epochs=1200,
23     adaptive_weights={'method':'NTK', 'freq':100})
24
25 # Test
26 nx, nt = 20, 10
27 x_test, t_test = np.meshgrid(
28     np.linspace(0.01, 0.99, nx+1),
29     np.linspace(0.01, 0.1, nt+1)
30 )
31 u_pred = u.eval(m, [x_test, t_test])
32
33 print(alpha.value)

```

Очигледно је да је код готово исти као претходни у коме се решава директни проблем, јер је и методологија за решавање директних и инверзних проблема код ФПНМ идентична. Једи-

на разлика је у поставци. Линија 4 поставља α као непознати параметар и даје му почетну вредност. У линији 13 се поставља додатни гранични услов у тачки $u(x=0.4, t=0.05)$, који ће постати још једна компонента композитне функције губитка која се формира у линији 15. Вредност непознатог параметра се штампа у последњој линији и у нашем тесту износи око 0,308, што је доволно близко реалној вредности од 0,3. Потврда задовољавајућег решења инверзног проблема приказана је и графички на Сл.2.7.



Слика2.7: Поље температуре дуж штапа у тренутку $t = 0, 1$ добијено решавањем инверзног проблема

2.4 Стефанов проблем у моделима топљења

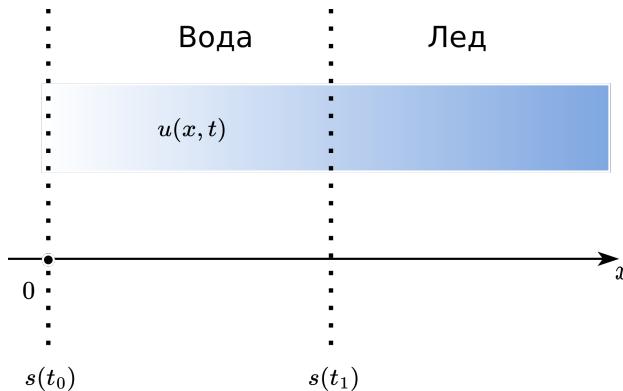
Након што смо успешно решили и директни и инверзни проблем провођења топлоте који је описан једноставном параболичном парцијалном диференцијалном једначином, позабавићемо се мало компликованијом поставком сличног проблема, пре свега имајући у виду граничне услове. Овде ће се они динамички мењати у зависности од градијента температуре. Показаћемо да је заправо овако постављен проблем лакше и природније решавати помоћу ФПНМ-а него помоћу класичних нумеричких метода базираних на мрежи интеграционих тачака (МКЕ, МКР).

Стефанови проблеми фазних промена имају примену у разним областима науке и инжењерства, када год се фаза посматране супстанце мења између течног, чврстог или гасовитог стања. Претпоставља се да материјал пролази кроз фазну промену са кретањем границе, чији је положај непознат и мора се одредити као део саме нумеричке анализе. Пошто проблеми са померањем граница захтевају решавање топлотне једначине у непознатој области која се

такође мора одредити као део решења, они су инхерентно нелинеарни.

Проблем једнодимензионалне промене фазе могао би се демонстрирати помоћу полубесковачног чврстог тела, као што је танак блок леда који заузима $0 \leq x < \infty$, на температури очвршћавања. На фиксној граници танког блока леда $x = 0$, могу да делују различите врсте флуksa. У овом примеру користимо исти гранични услов као Ivanovic *et al.* [ISS17], Savovic and Caldwell [SC09], тако се температура при $x = 0$ повећава експоненцијално са временом. Такође, прописујемо да се читава чврста фаза налази на температури топљења. Стога сводимо проблем на одређивање расподела температуре у течној фази у време t_0 , где је $x < s(t_0)$, као и положаја границе између фаза $s(t_0)$.

У неком каснијем временском тренутку $t_1 > t_0$, покретна граница $s(t)$ креће се удесно, заузимајући позицију $s(t_1) > s(t_0) = s_0$, као што је приказано на Сл.2.8. Део танког блока леда од позиције $s(t_0)$ до позиције $s(t_1)$, отопио се током временског интервала (t_0, t_1) .



Слика 2.8: Стефанов проблем у једној димензији. $s(t)$ означава покретну границу, а $u(x, t)$ температуру течне фазе (за $x < s$).

Дистрибуција температуре $u(x, t)$ у региону у коме влада течно стање $0 \leq x \leq s(t)$ дата је топлотном једначином:

$$\frac{\partial u}{\partial t} = \alpha \cdot \frac{\partial^2 u}{\partial x^2},$$

која може да се напише на следећи начин:

$$\frac{\partial u}{\partial t} - \alpha \cdot \frac{\partial^2 u}{\partial x^2} = 0, \quad (2.5)$$

под следећим граничним условима:

$$\begin{aligned} u(x, t) &= e^{\alpha t}, & x = 0, & t > 0 \\ u(x, t) &= 1, & x = s(t), & t > 0. \end{aligned} \quad (2.6)$$

Овде α , као и у претходним примерима означава физички параметар који комбинује топлотну проводност, густину и специфичну топлоту. Позиција покретне границе дата је једначином која је позната као Стефанов услов:

$$\frac{1}{\alpha} \cdot \frac{ds}{dt} = -\frac{\partial u}{\partial x}, \quad x = s(t), \quad t > 0. \quad (2.7)$$

У општем случају, почетни услов за положај границе фаза дат је као:

$$s(0) = 0. \quad (2.8)$$

За овако постављен проблем познато је и аналитичко решење, и то:

$$\begin{aligned} u(x, t) &= e^{\alpha t - x} \\ s(t) &= \alpha t. \end{aligned}$$

Решавање овог проблема ФПНМ приступом подразумева конструкцију две неуронске мреже. Прва ће апроксимирати дистрибуцију температуре $u(x, t)$ док ће друга апроксимирати положај слободне границе између фаза $s(t)$. Апроксимативна решења биће аутоматски диференцирана у односу на улазне варијабле од којих зависе, за вредности дефинисане скупом колокационих тачака из домена $[0, T] \times \mathcal{D}$, где је $\mathcal{D} \subset \mathbb{R}^d$ ограничени домен, а T означава коначно време симулације. Функција губитка састоји се из компоненти изведенних из (2.5), (2.6) и (2.8), користећи апроксимације за u и s у колокационим тачкама, које покривају како унутрашњост домена, тако и домене у којима важе почетни и гранични услови.

2.4.1 Конструкција функције губитка

Као што је већ речено, прва мрежа апроксимира функцију температуре $u(x, t)$, а друга мрежа апроксимира положај слободне границе између фаза $s(t)$. Функција губитка састоји се из разлике u и s и њихових апроксимација \hat{u} и \hat{s} које даје ФПНМ и који представљају резиду-уме које даје главна диференцијална једначина, почетни и гранични услови. Дакле, укупан губитак \mathcal{L} одређен је сумом резидуума:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0 + \mathcal{L}_{b_1} + \mathcal{L}_{b_2} + \mathcal{L}_{b_3}, \quad (2.9)$$

где су:

$$\begin{aligned} \mathcal{L}_r &= \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \frac{\partial \hat{u}(x, t)}{\partial t} - \alpha \frac{\partial^2 \hat{u}(x, t)}{\partial x^2} \right|^2, \\ \mathcal{L}_0 &= \frac{1}{N_0} \sum_{i=1}^{N_0} |\hat{s}(0) - s(0)|^2, \\ \mathcal{L}_{b_1} &= \frac{1}{N_{b_1}} \sum_{i=1}^{N_{b_1}} \left| \frac{1}{a} \frac{\partial \hat{s}(t)}{\partial t} + \frac{\partial \hat{u}}{\partial \hat{s}(t)} \right|^2, \\ \mathcal{L}_{b_2} &= \frac{1}{N_{b_2}} \sum_{i=1}^{N_{b_2}} |\hat{u}(0, t) - u(0, t)|^2, \\ \mathcal{L}_{b_3} &= \frac{1}{N_{b_3}} \sum_{i=1}^{N_{b_3}} |\hat{u}(\hat{s}(t), t) - u(s(t), t)|^2. \end{aligned} \quad (2.10)$$

Први члан \mathcal{L}_r пенализује по главној диференцијалној једначини (2.5), где је N_r величина *batch*-а колокационих талака које се случајно узоркују из домена простор-временских

координата које узимају вредности $0 \leq x \leq 1$ и $0s \leq t \leq 0,5s$, респективно. $\hat{u}(x, t)$ је апроксимативна неуронска мрежа температурског поља $u(x, t)$. Други члан \mathcal{L}_0 одређује испуњеност граничног услова (2.8). Испуњеност Стефановог граничног услова (2.7) дат је резидуумом \mathcal{L}_{b_1} , где $\hat{s}(t)$ означава ФПНМ апроксимацију положаја покретне границе. Последња два члана \mathcal{L}_{b_2} и \mathcal{L}_{b_3} одређују резидуале граничних услова (2.6), где N_0 , N_{b_1} , N_{b_2} , и N_{b_3} означавају број колокационих тачака у којима важе почетни и гранични услови.

2.4.2 Имплементација

Решење које користи функционалност већ познате библиотеке SCIAANN дато је у следећем листингу:

Листинг 2.3: Решење Стефановог проблема у 1Д коришћењем SCI-ANN библиотеке

```

1 alpha = 1.0
2
3 # Pocetni uslovi
4 t0 = 0.1
5 s0 = alpha * t0
6
7 # Varijable
8 x = sn.Variable('x')
9 t = sn.Variable('t')
10 u = sn.Functional ([ "u" ], [x, t], 3*[30] , 'tanh')
11 s = sn.Functional ([ "s" ], [t], 3*[30] , 'tanh')
12
13 # Glavna dif. jednacina
14 L1 = diff(u, t) - alpha * diff(u, x, order=2)
15
16 TOLX=0.004
17 TOLT=0.002
18
19 # Stefanov uslov
20 C1 = (1/alpha*diff(s, t) + diff(u,x)) * (1 + sign(x - (s-TOLX))) * (1-sign(x-s))
21 # Pocetno s u trenutku t=t0
22 C2 = ( s - s0 ) * (1-sign(t - (t0+TOLT)))
23 # Granicni uslov za u kada je x=0
24 C3 = ( u - exp(alpha*t) ) * (1-sign(x - (0 +TOLX)))
25 # Temperatura na granici izmedju faza je 1
26 C4 = (u-1) * (1-sign(x - (s+TOLX))) * (1+sign(x-s))
27
28 x_data, t_data = [], []
29
30 # Trening skup
31 x_train, t_train = np.meshgrid(
32     np.linspace(0, 1, 300),
33     np.linspace(t0, 0.5, 300)
34 )
35
36 x_data, t_data = np.array(x_train), np.array(t_train)
37
38 m = sn.SciModel([x, t], [L1,C1,C2,C3,C4], 'mse', 'Adam')
39 h = m.train([x_data, t_data], 5*['zero'], learning_rate=0.002, batch_size=1024,_

```

(наставак на следећој страни)

(настављено са претходне стране)

```

40 ↪epochs=200, adaptive_weights={ 'method':'NTK', 'freq':20})
41 # Test
42 x_test, t_test = np.meshgrid(
43     np.linspace(0, 1, 30),
44     np.linspace(0.01, 0.5, 30)
45 )
46 u_pred = u.eval(m, [x_test, t_test])
47 s_pred = s.eval(m, [x_test, t_test])
48
49 s=[]
50 for e in s_pred:
51     s.append(e[0])

```

На почетку (линије 1-11) постављамо константе и варијабле. Приметимо да моделовање креће од временског тренутка $t_0 = 0, 1s$. Прва заграда у изразу за C1

```
C1 = (1/alpha*diff(s, t) + diff(u, x)) * (1 + sign(x - (s-TOLX))) * (1-sign(x-s))
```

представља сам Стефанов услов. Као што је уобичајено за постављање граничних услова, друга заграда поставља правило где тај гранични услов важи. За C1 је тај израз мало компликованији због толеранције, али заправо представља услов да је $x \approx s$. Услов

```
C2 = (s - s0) * (1-sign(t - (t0+TOLT)))
```

дефинише положај границе у почетном тренутку. Даље, услов

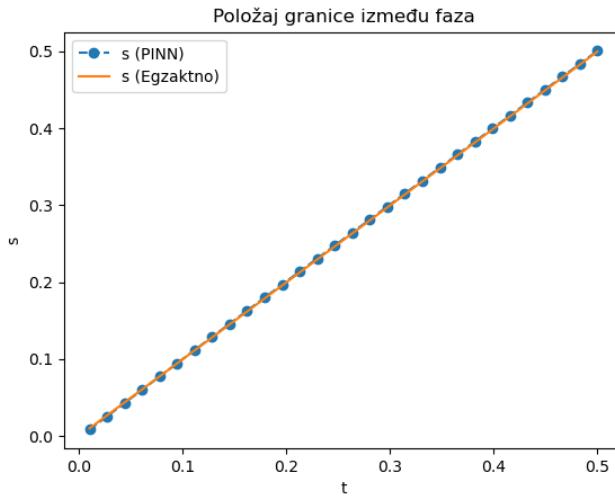
```
C3 = (u - exp(alpha*t)) * (1-sign(x - (0 +TOLX)))
```

поставља динамички услов промене температуре у тачки $x = 0$ према једначини (2.6). Коначно, последњи услов

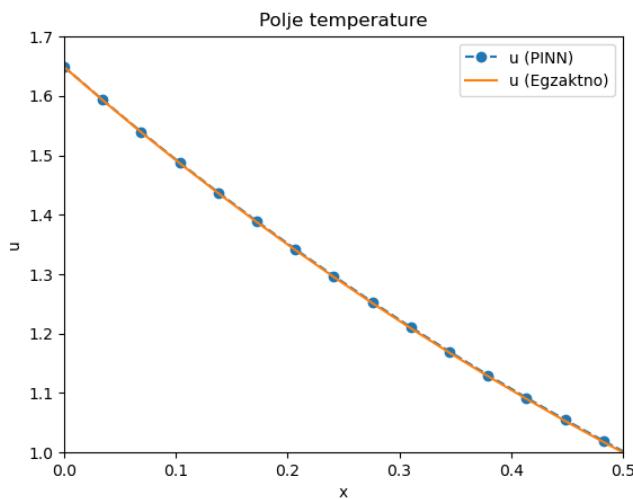
```
C4 = (u-1) * (1-sign(x - (s+TOLX))) * (1+sign(x-s))
```

у истом скупу колокационих тачака као што је то био случај са условом C1, тј. на граници између фаза $x \approx s$ поставља вредност температуре на 1. Остатак кода је мање-више стандардно тренирање, формирање тестног скупа и екстракција података о кретању границе s кроз време. Кретање границе између фаза кроз време може се видети на [Сл.2.9](#), док се поље температуре $u(x, t = 0, 5)$ може видети на [Сл.2.10](#).

Са дијаграма је очигледно да се ФПНМ (PINN) решења у задовољавајућој мери слажу са аналитичким решењем за посматрану појаву. У складу са претходном општот дискусијом о употребној вредности ФПНМ приступа, код директних проблема као што је овај, додатна вредност у односу на класичне нумеричке методе може се наћи у једноставнијој формулацији. Међутим, тек код инверзних проблема, када су и неки од параметара недовољно познати, приступ учења пропагацијом уназад показује своју праву снагу. Следећи одељак уводи непознати параметар у овај исти проблем.



Слика2.9: Кретање границе између фаза $s(t)$ током времена.



Слика2.10: Поље температура $u(x, t)$ у тренутку $t = 0, 5$

2.4.3 Инверзни 1-Д Стефанов проблем

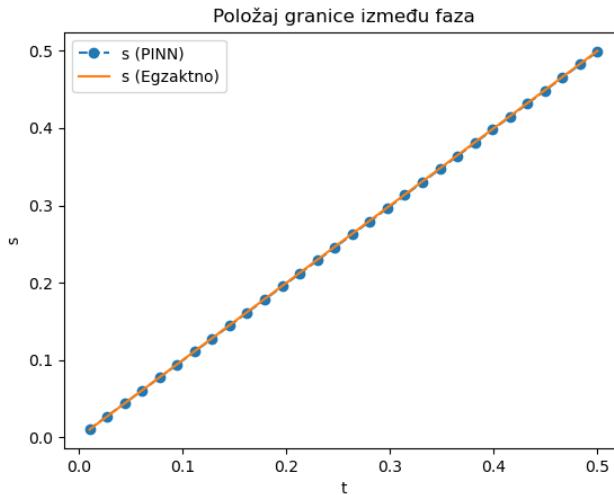
Претпоставим идентичну поставку Стефановог проблема, осим што сада није позната вредност материјалног параметра α , па тиме ни потпуни облик диференцијалне једначине, али је зато познато да је, на пример, у тренутку $t = 0,2$ граница између фаза уочена на координати $x = 0,2$. Непознати параметар увешћемо формулацијом:

```
# Nepoznati parameter
alpha = sn.Parameter(2.5, inputs=[x,t])
```

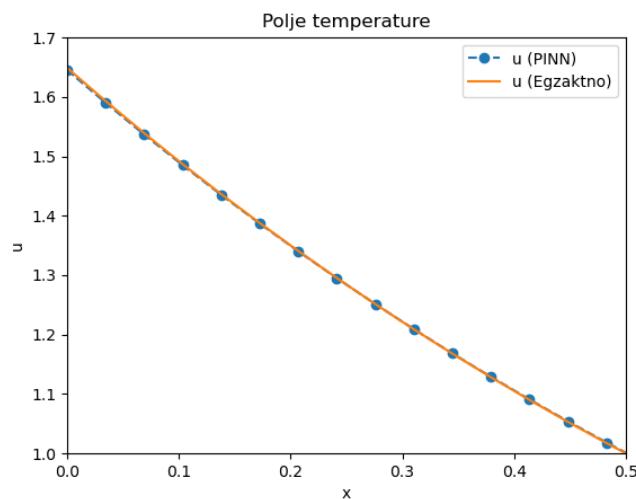
Параметру је дата почетна вредност од 2,5 и постављена зависност од обе улазне варијабле x и t . Додатни гранични услов C5 дат је стандардно:

```
# Dodatni uslov u tacki s(t=0.2)=0.2
C5 = (1-sign(t - (0.2+TOLT))) * (1+sign(t-0.2)) * (s-0.2)
```

Иако је овог пута потребно нешто више епоха у процесу тренинга, јер је почетна вредност непознатог параметра α (2,5) прилично далеко од реалне вредности (1), алгоритам који користи брзину учења од 0,001 у 100-инак епоха долази до вредности $\alpha = 0,99$ и заиста задовољавајућег поклапања са аналитичким решењем, као што се види на [Сл.2.11](#) за кретање границе и за поље температуре у $u(x, t = 0, 5)$ на [Сл.2.12](#).



Слика2.11: Кретање границе између фаза $s(t)$ током времена за инверзни проблем



Слика2.12: Поље температура $u(x, t)$ у тренутку $t = 0,5$ за инверзни проблем

Механичке осцилације

3.1 Осцилатор

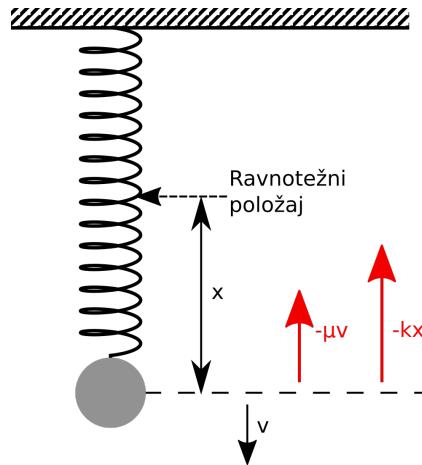
Једно од најједноставнијих периодичних кретања у механици је осцилаторно кретање (осциловање), где се тело креће по истој путањи, али мења смер кретања. Ако бисмо разматрали осцилаторно кретање у идеалним условима, без трења и отпора средине, оно би трајало бесконачно дуго. При таквим слободним, **непригушеним осцилацијама**, осцилатор не губи енергију, а амплитуда се не мења у току времена. Међутим, у реалним условима, мора да се узме у обзир утицај околине на кретање тела. Осциловање успорава са временом, смањују се амплитуде, јер се укупна механичка енергија троши на савладавање отпора средине. То су тзв. **пригашене осцилације**, када осцилатор губи енергију и амплитуда се смањује у току времена. Трећи случај би био када губитак енергије осцилатора може да се надокнади деловањем спољашње периодичне сile. Амплитуда осциловања ће при оваквом начину кретања остати константна, уколико се при сваком циклусу кретања у систем дода енергија једнака оној енергији коју је систем изгубио. Овакво кретање се назива **принудно осциловање**.

Проблем који решавамо представља моделовање осциловања тега који виси на опрузи, као што је приказано на Сл.3.1. Параметри који утичу на кретање су маса тега, кефицијент трења и коефицијент еластичности опруге. Силе које утичу на кретање осцилатора су:

- **Повратна сила опруге** $F_f = k \cdot x$ вуче тег ка тачки мiroвања и директно је пропорционална отклону клатна. Њен смер је супротан од смера отклона. Параметар k представља константу опруге.
- **Сила трења** $F_r = -\mu \cdot \dot{x}$ је пропорционална брзини тега, док је смер увек супротан смеру кретања тега. Параметар μ представља коефицијент трења.

Дакле, сила инерције је супротстављена двема силама:

$$m\ddot{x} = -\mu\dot{x} - kx$$



Слика3.1: Поставка експеримента са опругом и тегом. На тег делују сила еластичности опруге и сила трења.

или трансформисано:

$$\ddot{x} + \frac{\mu}{m} \dot{x} + \frac{k}{m} x = 0 \quad (3.1)$$

Ова једначина је линеарна хомогена диференцијална једначина другог реда са константним коефицијентима. За аналитичко решавање диференцијалне једначине овог типа користи се експоненцијална функција облика:

$$x(t) = Ce^{\lambda t}$$

Први и други извод ове једначине гласе:

$$\dot{x}(t) = \lambda C e^{\lambda t}, \quad \ddot{x}(t) = \lambda^2 C e^{\lambda t} \quad (3.2)$$

Заменом једначина (3.2) у диференцијалну једначину (3.1) и скраћивањем добијамо:

$$\lambda^2 + \frac{\mu}{m} \lambda + \frac{k}{m} = 0 \quad (3.3)$$

Ова једначина се назива **карактеристичном једначином**. Како је у питању квадратна једначина, разматрамо два решења:

$$\lambda_{1,2} = -\frac{\mu}{2m} \pm \sqrt{\left(\frac{\mu}{2m}\right)^2 - \frac{k}{m}}.$$

Да бисмо додатно упростили израз, уводимо нове константе δ и ω_0 :

$$\delta = \frac{\mu}{2m}, \quad \omega_0 = \sqrt{\frac{k}{m}},$$

па се решења карактеристичне једначине могу изразити у облику:

$$\lambda_{1,2} = -\delta \pm \sqrt{\delta^2 - \omega_0^2}. \quad (3.4)$$

У зависности од избора константи δ и ω_0 , дискриминанта може бити: већа од нуле, мања од нуле или једнака нули. Стога λ_1 и λ_2 могу бити:

- два реална различита решења,
- два конјуговано комплексна решења и
- два једнака реална решења.

Сваки од ових случајева захтева другачији приступ аналитичком решавању. Уопштено решење хомогене једначине има облик:

$$x(t) = C_1 \cdot x_1(t) + C_2 \cdot x_2(t), \quad (3.5)$$

где функције $x_1(t)$ и $x_2(t)$ зависе од вредности детерминанте у једначини (3.4). Сада ћемо размотрити све наведене случајеве.

3.1.1 Препригушени случај

Уколико је $\delta > \omega_0$, онда доминира сила трења. Стога је дискриминанта у једначини (3.4) позитивна и постоје два различита реална решења $\lambda_1 \neq \lambda_2$:

$$x_1(t) = C_1 e^{\lambda_1 t}, \quad x_2(t) = C_2 e^{\lambda_2 t}.$$

Заменом у једначину (3.5) добијамо опште решење диференцијалне једначине:

$$x(t) = C_1 e^{(-\delta + \sqrt{\delta^2 - \omega_0^2})t} + C_2 e^{(-\delta - \sqrt{\delta^2 - \omega_0^2})t}.$$

Услов који смо навели за препригушени случај налаже да је

$$\delta > \sqrt{\delta^2 - \omega_0^2},$$

па је стога решење збир двеју експоненцијалних опадајућих функција. Како бисмо даље упростили израз, заменићемо корен новом константом:

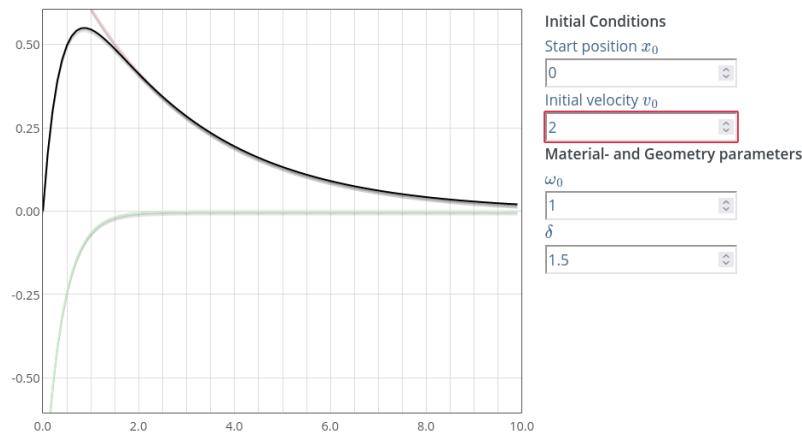
$$\alpha = \sqrt{\delta^2 - \omega_0^2},$$

па коначно решење за препригушени случај гласи:

$$x(t) = e^{-\delta t} (C_1 e^{\alpha t} + C_2 e^{-\alpha t}).$$

Интеграционе константе C_1 и C_2 можемо добити из почетних услова, тј. почетног положаја и почетне брзине тега:

$$x(0) = x_0, \quad \dot{x}(0) = v_0.$$



Слика3.2: Промена положаја тега у току времена за препригушени случај осциловања. До-
бијено помоћу симулатора^{Страна 30, 8}.

На Сл.3.2 можемо видети графички приказ општег аналитичког решења за препригушени случај осцилатора. Овај дијаграм приказује кретање тега током времена. Црвена линија означава компоненту решења $C_1 e^{\lambda_1 t}$, док зелена означава другу компоненту решења $C_2 e^{\lambda_1 t}$. Црна линија је збир ова два парцијална решења и представља укупно решење диференцијалне једначине препригашеног случаја за задате почетне услове.

3.1.2 Критично-пригушени случај

Овај случај се дешава када је $\delta = \omega_0$. У овом случају једначина (3.4) има само једно решење

$$\lambda = \lambda_1 = \lambda_2 = -\delta$$

Две компоненте решења диференцијалне једначине су онда:

$$x_1(t) = C_1 e^{\lambda t}, \quad x_2(t) = tC_2 e^{\lambda t}.$$

Заменом ових израза у опште решење (3.5) добијамо:

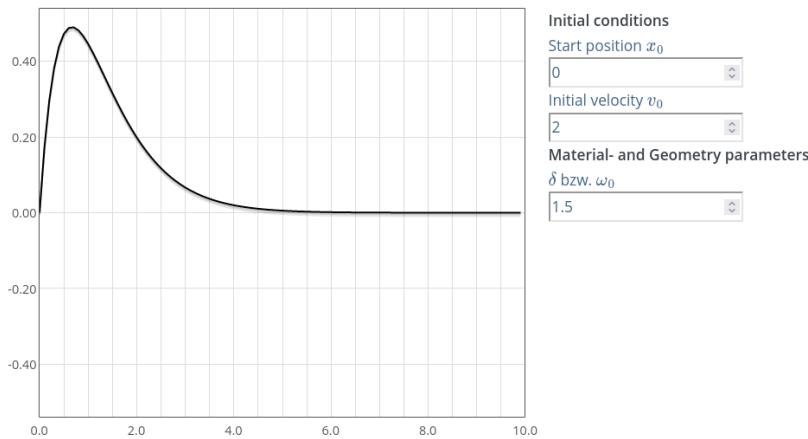
$$x(t) = e^{-\delta t}(C_1 + tC_2)$$

Интеграционе константе C_1 и C_2 можемо добити из почетних услова, тј. почетног положаја и почетне брзине тега.

На Сл.3.3 видимо графички приказ општег решења критично пригашеног случаја. На дијаграму опажамо кретање клатна током времена за задате почетне услове.

⁸ https://beltoforion.de/en/harmonic_oscillator/

⁹ https://beltoforion.de/en/harmonic_oscillator/



Слика3.3: Промена положаја тега у току времена за критично пригушени случај осциловања. Добијено помоћу [симулатора](#)⁹.

3.1.3 Подпригушки случај

Подпригушки случај наступа када је $\delta < \omega_0$, тј. дискриминанта једначине (3.4) је негативна. Стога су λ_1 и λ_2 комплексни бројеви. Експоненцијални израз

$$x(t) = Ce^{\lambda t}$$

се поново користи за добијање компоненти решења ове диференцијалне једначине:

$$x_1(t) = C_1 e^{\lambda_1 t}, \quad x_2(t) = C_2 e^{\lambda_2 t}.$$

Заменом у израз (3.5) и заменом λ добијамо:

$$x(t) = e^{-\delta t} \left(C_1 e^{\sqrt{\delta^2 - \omega_0^2} t} + C_2 e^{-\sqrt{\delta^2 - \omega_0^2} t} \right). \quad (3.6)$$

Сада радимо са комплексним решењима јер су вредности негативне. Стога константе C_1 и C_2 имају комплексне вредности. За рад са комплексним вредностима користимо Ојлерову формулу:

$$e^{i\phi} = \cos \phi + i \sin \phi$$

Корисно би било да изменимо једначину тако што бисмо раздвојили имагинарне делове:

$$\sqrt{\delta^2 - \omega_0^2} = \sqrt{-1 \cdot (\omega_0^2 - \delta^2)} = i\sqrt{\omega_0^2 - \delta^2}.$$

Добијамо део који се састоји од имагинарне вредности i помноженом кореном реалних вредности. Да бисмо упростили даља израчунавања, замењујемо корен новом константом

$$\omega = \sqrt{\omega_0^2 - \delta^2}$$

Ова константа представља **природну фреквенцију хармонијског осцилатора**. Тада једначина (3.6) може да се трансформише у:

$$x(t) = e^{-\delta t} (C_1 e^{i\omega t} + C_2 e^{-i\omega t}).$$

Са физичке стране, интересују нас само реалне вредности. Да бисмо их пронашли, неопходно је да раздвојимо имагинарни и реални део. Као што је већ поменуто, константе C_1 и C_2 су константе са комплексним вредностима, а њихов поларни облик је:

$$C_1 = \hat{C}_1 e^{i\phi_1}, \quad C_2 = \hat{C}_2 e^{i\phi_2}$$

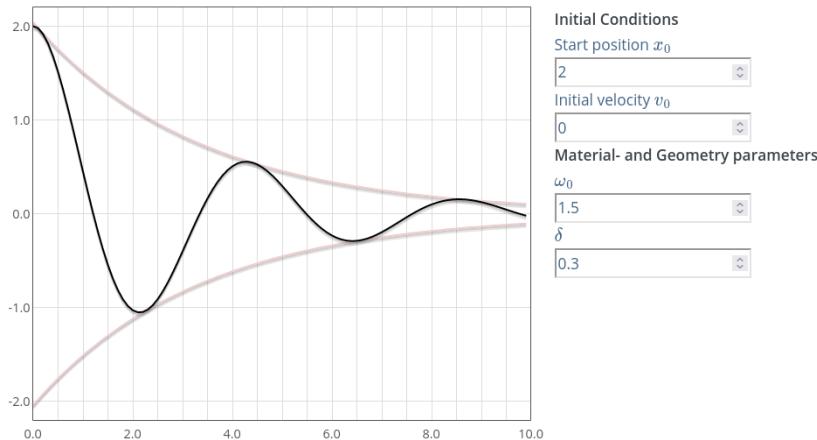
Како аналитичко решавање овог проблема није у фокусу овог материјала, нећемо до краја аналитички изводити израз, већ само навести крајње решење за подпригушени случај, након одабира реалних решења и трансформације и одговарајућих константи:

$$x(t) = e^{-\delta t} (2A \cos(\phi + \omega t))$$

A је амплитуда, а ϕ је фазни померај. Константе се могу добити из почетних услова, као што су:

$$x(0) = x_0, \quad \dot{x}(0) = v_0.$$

Овим смо извели сва потребна аналитичка решења са којима ћемо поредити решење добијено помоћу ФПНМ приступа.



Слика3.4: Промена положаја тега у току времена за подпригушени случај осциловања. Добијено помоћу симулатора¹⁰.

¹⁰ https://beltoforion.de/en/harmonic_oscillator/

3.2 Имплементација

Да бисмо реализовали предложени модел пригушених осцилација у једној димензији описан обичном диференцијалном једначином у секцији *Осцилатор* (страна 27), уместо библиотеке SCIAANN користићемо нешто новију библиотеку DeepXDE Lu *et al.* [LMMK21]. Идеја је да овим материјалом покријемо све значајне софтверске оквире који подржавају ФПНМ методологију у тренутку писања (2022-2023). Како аутори кажу, DeepXDE библиотека је дизајнирана да служи и као образовно средство које ће се користити у високом школству и као истраживачки алат за решавање проблема у компјутерским наукама и инжењерству. Конкретно, DeepXDE може да решава проблеме за унапред дате почетне и граничне услове, као и инверзне проблеме уз нека додатна мерења. Подржава домене сложене геометрије и омогућава да кориснички код буде компактан, веома сличан математичкој формулатици.

У односу на SCIAANN, приступ DeepXDE подразумева нешто виши степен апстракције, па тиме и лакоће употребе. На пример, компликације око постављања колокационих тачака у којима важе гранични и почетни услови, као и величина *batch*-а, не потпадају под бригу корисника, већ се сама библиотека стара о томе да специфицирани број тачака подлеже граничним условима. Тај ниво апстракције у неким специфичним случајевима може представљати препреку, али у великој већини случајева доприноси јаснијем дефинисању проблема.

На следећем листингу дати су значајни делови имплементације:

Листинг 3.1: Решење проблема пригушених осцилација у 1Д коришћењем DeepXDE библиотеке

```

1 import deepxde as dde
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 m = 1
6 mu = 0.1
7 k = 2
8
9 # Pocetni uslovi
10 x0, v0 = 0, 2
11
12 delta = mu / (2*m)
13 w0 = np.sqrt(k/m)
14
15 # Da li je tacka blizu t=0 (provera pocetnog uslova)
16 def boundary_l(t, on_initial):
17     return on_initial and np.isclose(t[0], 0)
18
19 # Jednacina ODE
20 def Ode(t, x):
21     dxdt = dde.grad.jacobian(x,t)
22     dxdtt = dde.grad.hessian(x,t)
23     return m * dxdtt + mu * dxdt + k*x
24
25 # x(0)=x0
26 def bc_func1(inputs, outputs, X):
27     return outputs - x0
28

```

(наставак на следећој страни)

(настављено са претходне стране)

```

29 # x'(0)=v0
30 def bc_func2(inputs, outputs, X):
31     return dde.grad.jacobian(outputs, inputs, i=0, j=None) - v0
32
33 # Resava se na domenu t=(0,10)
34 interval = dde.geometry.TimeDomain(0, 10)
35
36 # Pocetni uslovi
37 ic1 = dde.icbc.OperatorBC(interval, bc_func1, boundary_1)
38 ic2 = dde.icbc.OperatorBC(interval, bc_func2, boundary_1)
39
40 # Defisanje problema, granicnih uslova, broja kolokacionih tacaka
41 data = dde.data.TimePDE(interval, Ode, [ic1, ic2], 100, 20, solution=func, num_
42 ↪test=100)
43
44 layers = [1] + [30] * 2 + [1]
45 activation = "tanh"
46 init = "Glorot uniform"
47 net = dde.nn.FNN(layers, activation, init)
48
49 model = dde.Model(data, net)
50
51 model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1], metrics=["l2 relative_
52 ↪error"])
53 losshistory, train_state = model.train(epochs=30000)
54 T = np.linspace(0, 10, 100).reshape(100,1)
x_pred = model.predict(T)

```

На почетку се дефинишу импорти и константе проблема, као и додатне константе `delta` и `w0` које смо користили приликом извођења аналитичког решења. Функцију

```

def boundary_1(t, on_initial):
    return on_initial and np.isclose(t[0], 0)

```

ћемо искористити за тестирање да ли је дата колокациона тачка близу тачке $t=0$, тј. да ли за дату тачку важи почетни услов. Употребићемо је при формирању почетних услова за позицију и брзину. Даље, као што назив сугерише, наредна функција

```

def Ode(t,x):
    dxdt = dde.grad.jacobian(x,t)
    dxdtt = dde.grad.hessian(x,t)
    return m * dxdt + mu * dxdtt + k*x

```

представља поставку проблема у свом изворном облику једначине (3.1). Овде је очигледна једна од главних предности ФПНМ, тј. да неке додатне трансформације у интеграционим тачкама нису потребне, већ само поставка диференцијалне једначине у виду функције губитка и граничних услова у истом облику. Услужне методе `dde.grad.jacobian` и `dde.grad.hessian` враћају прве, односно друге изводе по улазним варијаблама примењуући тзв. аутоматску диференцијацију. Подразумевано се у позадини користи `Tensorflow` за тензорске операције ниског нивоа.

Поставка два почетна услова у форми функције губитка дата је у следеће две методе, за координату и брзину респективно:

```
def bc_func1(inputs, outputs, x):
    return outputs - x0

def bc_func2(inputs, outputs, x):
    return dde.grad.jacobian(outputs, inputs, i=0, j=None) - v0
```

Након поставке једнодимензионог временског домена у коме се проблем решава:

```
interval = dde.geometry.TimeDomain(0, 10)
```

можемо да формирамо и објекте граничних услова комбинујући функције губитка са функцијом локације `boundary_1()`:

```
ic1 = dde.icbc.OperatorBC(interval, bc_func1, boundary_1)
ic2 = dde.icbc.OperatorBC(interval, bc_func2, boundary_1)
```

Сада имамо све елементе да формирамо објекат проблема који решавамо. Овде ћемо то учинити методом `dde.data.TimePDE` за временски зависне проблеме:

```
data = dde.data.TimePDE(interval, Ode, [ic1, ic2], 100, 20, solution=func, num_
→test=100)
```

Специфицирамо редом рачунски домен, основну једначину, листу граничних услова, број колокационих тачака за основни домен (100), број колокационих тачака за граничне услове (20), егзактно решење (ако постоји) и број тестних тачака (за поређење са егзактним решењем). У овом примеру ћемо игнорисати егзактно решење. Одмах се види разлика у поставци у односу на SCANN приступ у начину навођења колокационих тачака. Наиме, код DeepXDE колокационе тачке се не генеришу мануелно, већ се препушта библиотеци да то уради за нас, што указује на један виши ниво апстракције.

Наредне линије кода конструишу неуронску мрежу која ће се користити као апроксимација проблема, са свим својим хипер-параметрима:

```
layers = [1] + [30] * 2 + [1]
activation = "tanh"
init = "Glorot uniform"
net = dde.nn.FNN(layers, activation, init)
```

Наша мрежа има један улаз, један излаз и два скривена слоја од по 30 неурона, са активацијом скривених слојева у виду `tanh` функције и одговарајућом иницијализацијом. На крају, можемо да кренемо у обучавање, када спојивши проблем и генерисану неуронску мрежу формирамо модел:

```
model = dde.Model(data, net)
model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1], metrics=["l2 relative_
→error"])
losshistory, train_state = model.train(epochs=30000)
```

Ово су стандардне методе које се широко користе у области дубоког учења, па је доволјно само поменути да се наводи алгоритам оптимизације, брзина учења и начин прорачуна грешке која управља овим процесом. Специфичност за ФПНМ је што овде листом

`loss_weights` можемо и да „пондеришемо“ тежине основне диференцијалне једначине, првог и другог граничног услова, респективно. У наредној секцији *Решења директног проблема* (страница 36) ћемо размотрити решења за сва три случаја пригушеног осциловања.

3.3 Решења директног проблема

Сада ћемо се позабавити резултатима процеса учења који смо успоставили у претходној секцији *Имплементација* (страница 33).

3.3.1 Подпригушени случај

За случај да је:

$$m = 1$$

$$\mu = 0, 1$$

$$k = 2$$

имамо да су:

$$\delta = \frac{\mu}{2m} = 0, 05$$

$$\omega_0 = \sqrt{\frac{k}{m}} = \sqrt{2}$$

Како је $\delta^2 - \omega_0^2 < 0$, имаћемо два конјуговано-комплексна решења, тј. подпригушени случај описан у секцији *Подпригушени случај* (страница 31). Решења добијена скриптом датом у *Имплементација* (страница 33), где су почетни услови постављени тако да је $(x_0 = 0, v_0 = 2)$ приказана су на [Сл.3.5](#).

На графику се може видети како се резултати разликују у односу на то колико епоха је мрежа тренирана. Наиме, резултати за 10000 епоха су значајно лошији него они за 20000 и 30000 епоха. Дакле, као и код готово свих проблема дубоког учења и код ФПНМ тај процес треба пратити ([Сл.3.6](#)) и тренинг прекинути тек када је досегнут одговарајући минимум и учење даље не напредује значајно.

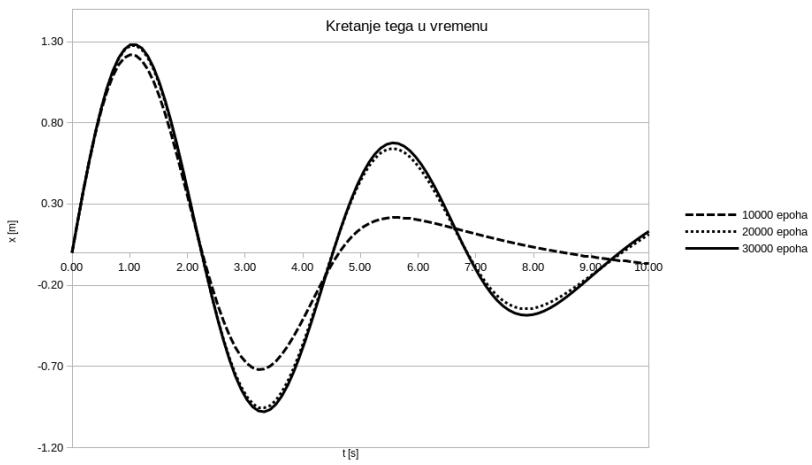
3.3.2 Препригушени случај

Уколико је, на пример:

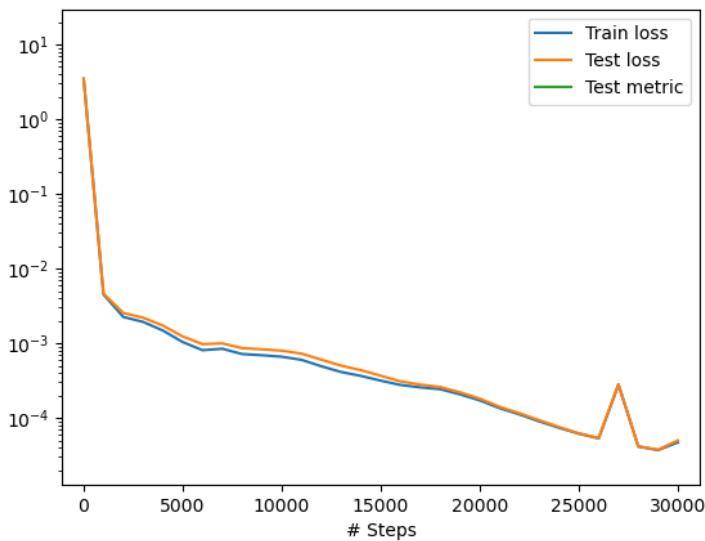
$$m = 1$$

$$\mu = 3$$

$$k = 1$$



Слика3.5: ФПНМ решење промене положаја тега у току времена за подпригушени случај осциловања.



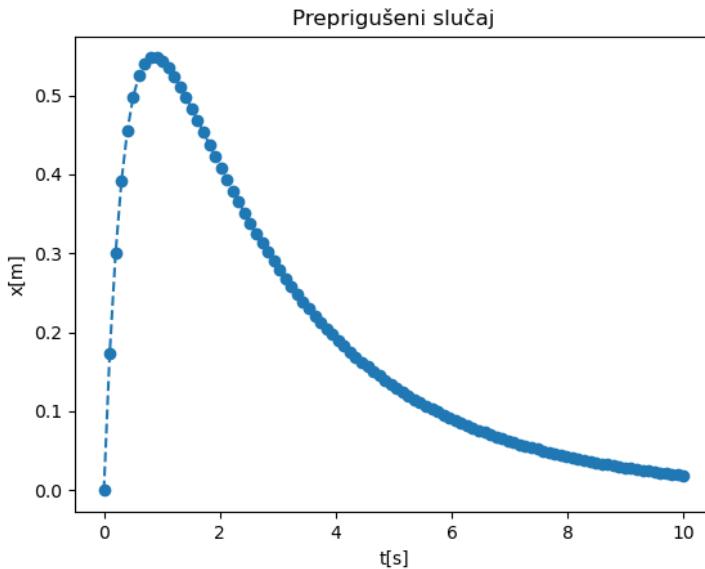
Слика3.6: Функција губитка у току процеса учења.

имамо да су:

$$\delta = \frac{\mu}{2m} = 1,5$$

$$\omega_0 = \sqrt{\frac{k}{m}} = 1$$

Како је $\delta^2 - \omega_0^2 > 0$, имаћемо два различита реална решења, тј. препригушени случај описан у секцији *Препригушени случај* (страница 29). Решења добијена скриптом датом у *Имплементација* (страница 33) за почетне услове ($x_0 = 0, v_0 = 2$) приказана су на Сл.3.7.



Слика3.7: ФПНМ решење промене положаја тега у току времена за препригушени случај осциловања.

3.3.3 Критично-пригушени случај

Преостао је још критично-пригушени случај, који ће се добити уколико поставимо следеће параметре проблема:

$$m = 1$$

$$\mu = 3$$

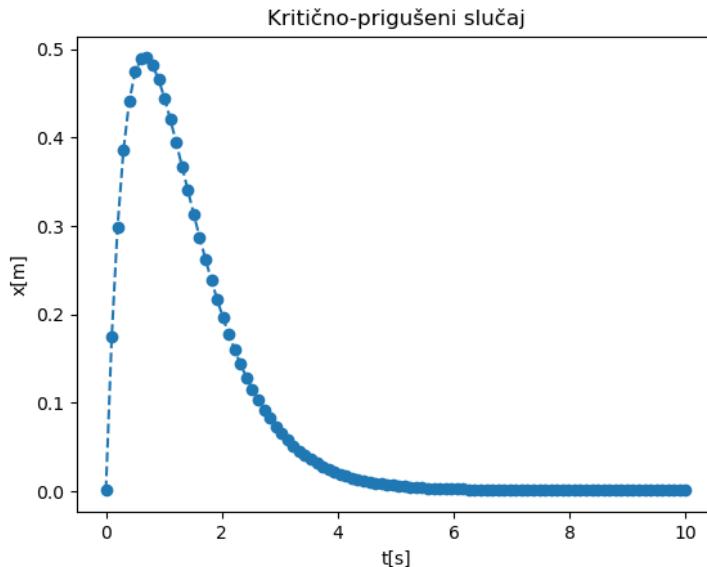
$$k = 2,25$$

Имамо да су:

$$\delta = \frac{\mu}{2m} = 1,5$$

$$\omega_0 = \sqrt{\frac{k}{m}} = 1,5$$

Како је $\delta^2 - \omega_0^2 = 0$, имаћемо два једнака реална решења, тј. критично-пригушени случај описан у секцији *Критично-пригушени случај* (страна 30). Решења добијена скриптом датом у *Имплементација* (страна 33) за почетне услове ($x_0 = 0, v_0 = 2$) приказана су на Сл.3.8.



Слика3.8: ФПНМ решење промене положаја тега у току времена за критично-пригушени случај осциловања.

3.4 Инверзни проблем

Како што смо већ у више наврата нагласили, права снага ФПНМ долази до изражaja код проблема у којима је потребно идентификовати непознате параметре. Разлог томе је што ФПНМ метода инверзне проблеме третира на исти начин као директне. Важно је само да се проблем постави не дозвољавајући вишезначност, што ћемо демонстрирати на следећем примеру. Замислимо експеримент постављен као на Сл.3.1 у коме нам је познат само коефицијент трења $\mu = 0,6$, а не зnamо ни масу куглице m ни коефицијент еластичности опруге k . Задатак нам је да идентификујемо ова два параметра тако што ћемо пустити куглицу да осцилује и на основу њеног кретања дати ФПНМ да одреди елементе који недостају.

Рецимо да смо визуелно утврдили да је у питању **подпригушени случај**. Прва стратегија мерења која би могла да буде реално изводљива је да куглицу отклонимо за $x_0 = 2$ и само пустимо ($v_0 = 0$), а онда штогерицом одредимо временске тренутке у којима куглица пролази кроз равнотежни положај и да те тренутке забележимо:

t	1,18	3,27	5,37	7,46	9,55
x	0	0	0	0	0

Дакле, поред граничних и почетних услова ($x_0 = 2, v_0 = 0$) увешћемо и граничне услове типа PointSet који дефинишу вредност моделоване функције у појединим тачкама. Програмски код који имплементира овај инверзни проблем дат је на следећем листингу.

Листинг 3.2: Инверзни проблем пригушених осцилација у 1Д. Непознати параметри су m и k .

```

1 import deepxde as dde
2 from deepxde.backend import tf
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 m = dde.Variable(0.5)
7 mu = 0.6
8 k = dde.Variable(2.)
9
10 x0, v0 = 2, 0
11
12 delta = mu / (2*m)
13 w0 = tf.sqrt(k/m)
14
15 # Da li je tacka blizu t=0 (provera pocetnog uslova)
16 def boundary_l(t, on_initial):
17     return on_initial and np.isclose(t[0], 0)
18
19 # Jednacina ODE
20 def Ode(t, x):
21     dxdt = dde.grad.jacobian(x, t)
22     dxdtt = dde.grad.hessian(x, t)
23     return m * dxdtt + mu * dxdt + k*x
24
25 # x(0)=x0
26 def bc_func1(inputs, outputs, X):
27     return outputs - x0
28
29 # x'(0)=v0
30 def bc_func2(inputs, outputs, X):
31     return dde.grad.jacobian(outputs, inputs, i=0, j=None) - v0
32
33 # Resava se na domenu t=(0, 10)
34 interval = dde.geometry.TimeDomain(0, 10)
35
36 # Pocetni uslovi
37 ic1 = dde.icbc.OperatorBC(interval, bc_func1, boundary_l)
38 ic2 = dde.icbc.OperatorBC(interval, bc_func2, boundary_l)
39
40 bc_x = np.array([1.18, 3.27, 5.37, 7.46, 9.55]).reshape(6,1)
41 bc_y = np.array([0, 0, 0, 0, 0]).reshape(6,1)
42 ic3 = dde.icbc.PointSetBC(bc_x, bc_y, component=0)
43
44 # Definsanje problema, granicnih uslova, broja kolokacionih tacaka
45 data = dde.data.TimePDE(interval, Ode, [ic1, ic2, ic3], 200, 20, solution=func,
46     ↪num_test=100)
47
48 layers = [1] + [30] * 3 + [1]
49 activation = "tanh"
50 init = "Glorot uniform"
```

(наставак на следећој страни)

(настављено са претходне стране)

```

50 net = dde.nn.FNN(layers, activation, init)
51
52 model = dde.Model(data, net)
53
54 # Callback funkcija koja stampa varijablu na svakih 1000 epoha
55 variable1 = dde.callbacks.VariableValue(k, period=1000)
56 variable2 = dde.callbacks.VariableValue(m, period=1000)
57
58 model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1, 1], metrics=["12 ↴relative error"], external_trainable_variables=[k,m])
59 losshistory, train_state = model.train(epochs=50000, callbacks=[variable1, ↴variable2])

```

Објаснићемо само делове који се разликују у односу на директни проблем решен у секцији *Имплементација* (страна 33). За почетак, ту су непознати параметри које декларишемо на следећи начин:

```

m = dde.Variable(0.5)
k = dde.Variable(2.)

```

У загради се дају почетне вредности параметра. Следеће линије дефинишу поменути додатни PointSet гранични услов (услове) који важе у појединим тачкама унутар домена:

```

bc_x = np.array([1.18, 3.27, 5.37, 7.46, 9.55]).reshape(6,1)
bc_y = np.array([0, 0, 0, 0, 0]).reshape(6,1)
ic3 = dde.icbc.PointSetBC(bc_x, bc_y, component=0)

```

Како бисмо обезбедили праћење вредности непознатих параметара током обуке, потребно је да поставимо тзв. callback функције, које ће се позивати на сваких 1000 епоха:

```

variable1 = dde.callbacks.VariableValue(k, period=1000)
variable2 = dde.callbacks.VariableValue(m, period=1000)

```

Приликом постављања модела постављамо екстерне варијабле за тренирање k и m, док се при позиву тренинга наводе callback функције:

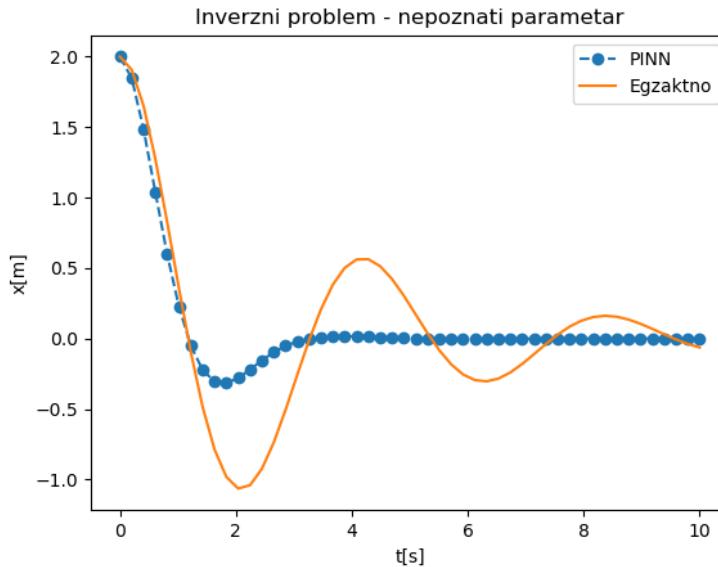
```

model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1, 1], metrics=["12 ↴relative error"], external_trainable_variables=[k,m])
losshistory, train_state = model.train(epochs=50000, callbacks=[variable1, ↴variable2])

```

Након завршеног обучавања, добија се, очигледно погрешно, решење које је приказано на Сл.3.9. У односу на аналитичко решење које је постављено користећи вредности параметара m=1 и k=2, 25, добијене вредности m=0, 287 и k=1, 22 се пуно разликују.

Зашто смо добили овако лоше решење? Одговор се крије у лоше постављеним граничним условима који доводе до неједнозначности инверзног проблема. Наиме, постоји више сценарија, тј. парова (m, k) који задовољавају граничне услове постављене само у тачкама проласка тега кроз равнотежни положај. Очигледно је да морамо да додамо још неку тачку ван равнотежног положаја, како бисмо обезбедили једнозначно решење. Замислимо да смо измерили и тренутак када је тег био на највећој негативној удаљености у односу на



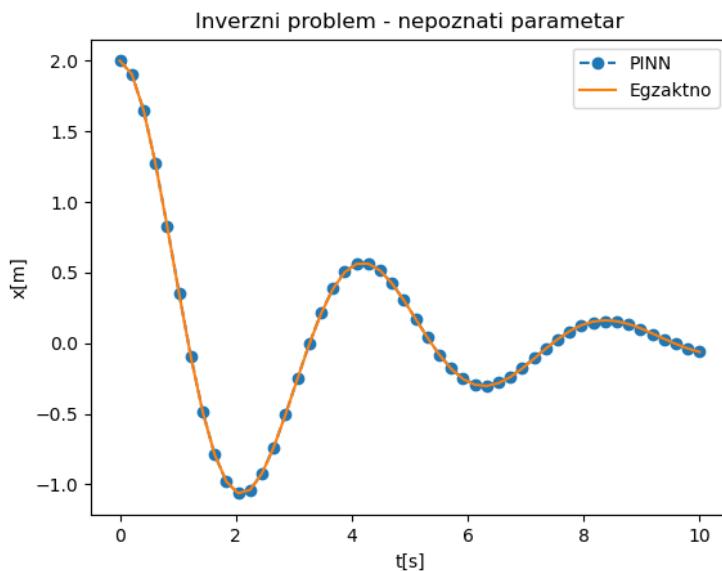
Слика3.9: ФПНМ решење инверзног проблема са непознатим параметрима. Пронађене вредности су $m=0, 287$ и $k=1, 22$.

равнотежни положај и колика је та удаљеност била. Додајмо сада и ту тачку у постављени PointSet, који сада изгледа овако:

t	1,18	3,27	5,37	7,46	9,55	2,12
x	0	0	0	0	0	-1,67

Погледом на Сл.3.10 одмах се види да је поклапање са аналитичким решењем у овако постављеном проблему скоро па идеално.

Исправност решења додатно потврђују параметри $m=0, 98$ и $k=2, 26$, чије су вредности веома близке онима које су дате у аналитичкој поставци $m=1$ и $k=2, 25$. На овај начин смо показали да приступ решавању инверзног проблема, иако методолошки сличан, има специфичности о којима треба водити рачуна. Код директних проблема решење је увек једнозначно, док код инверзних морају да се обезбеде одговарајући услови који у доволној мери детерминишу решење.



Слика3.10: ФПНМ решење инверзног проблема са непознатим параметрима. Пронађене вредности су $m=0, 98$ и $k=2, 26$.

4.1 Увод

У овом поглављу бавићемо се хидролошким проблемима које су аутор и сарадници имали прилике да решавају у пракси. Поглавље се састоји из два решена примера. Први пример се бави филтрацијом подземних вода, док се други бави проблематиком предвиђања понашања поплавног таласа. Оба проблема решаваћемо помоћу ФПНМ и успут наглашавати разлике у односу на третман методом коначних елемената, која је у овој области *de-facto* стандард.

4.2 Филтрација подземних вода

Основна величина од које се полази у теорији струјања течности кроз порозно тло је потенцијал ϕ дефинисан као

$$\phi = \frac{p}{\gamma} + h,$$

где је p притисак течности, γ специфична тежина, а h висина у вертикалном правцу мерења у односу на изабрану референтну раван. Брзина течности \mathbf{q} , позната и као Дарсијева брзина, представља запремину течности која прође у јединици времена кроз јединичну површину порозне средине. Она се може изразити помоћу потенцијала ϕ релацијом која се зове Дарсијев закон:

$$\mathbf{q} = -\mathbf{K}\nabla\phi,$$

где је \mathbf{K} матрица пермеабилности која за ортотропни материјал има облик

$$\mathbf{K} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix},$$

где су k_x , k_y и k_z коефицијенти пермеабилости у одговарајућим правцима. Компонентни облик једначине је према томе:

$$\begin{aligned} q_x &= -k_x \frac{\partial \phi}{\partial x}, \\ q_y &= -k_y \frac{\partial \phi}{\partial y}, \\ q_z &= -k_z \frac{\partial \phi}{\partial z}. \end{aligned}$$

Сада у систем уводимо једначину континуитета. У случају стационарног струјања нестишљивог флуида, каква је вода, једначина континуитета има облик

$$\nabla^T \mathbf{q} = 0,$$

или уз коришћење Дарсијевог закона

$$\frac{\partial}{\partial x} \left(k_x \frac{\partial \phi}{\partial x} \right) + \frac{\partial}{\partial y} \left(k_y \frac{\partial \phi}{\partial y} \right) + \frac{\partial}{\partial z} \left(k_z \frac{\partial \phi}{\partial z} \right) = 0$$

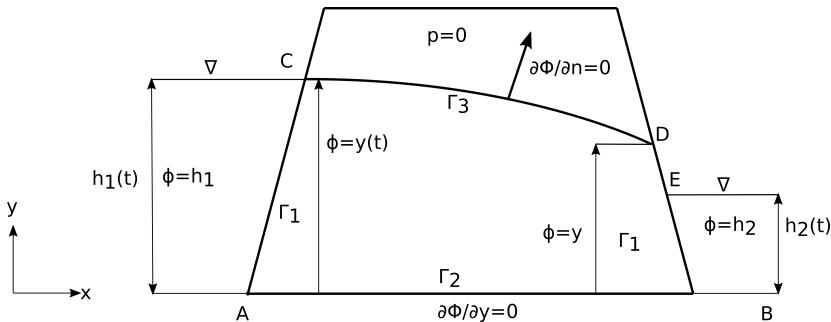
У случају када се промена коефицијената k_x, k_y, k_z са координатама може занемарити, што је најчешћи случај, једначина се своди на

$$k_x \frac{\partial^2 \phi}{\partial x^2} + k_y \frac{\partial^2 \phi}{\partial y^2} + k_z \frac{\partial^2 \phi}{\partial z^2} = 0$$

Конечно, ако постоји извор и/или понор, за стационарне услове, **хидродинамичка једначина** има следећи облик:

$$k_x \frac{\partial^2 \phi}{\partial x^2} + k_y \frac{\partial^2 \phi}{\partial y^2} + k_z \frac{\partial^2 \phi}{\partial z^2} + \bar{Q} = 0 \quad (4.1)$$

где је \bar{Q} запремински флукс (извор или понор као количина течности по јединици запремине порозне средине). Границни услови који се срећу у решавању проблема струјања кроз порозну средину описаног горњим једначинама приказани су на Сл.4.1.



Слика4.1: Различити гранични услови код проблема филтрације у две димензије.

Они могу бити:

1. задат потенцијал

$$\phi = \bar{\phi}, \quad | \Gamma_1$$

2. задат површински проток (флукс)

$$q_n = \bar{q} \quad | \Gamma_2$$

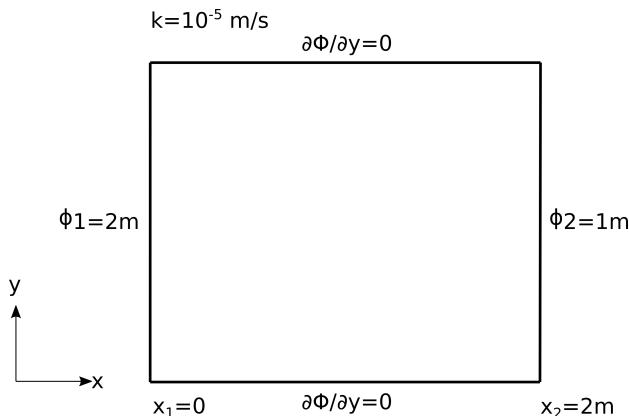
3. слободна површина

$$p = 0, \phi = y, \frac{\partial \phi}{\partial n} = 0 \quad | \Gamma_3 \quad (4.2)$$

Приметимо да је на слободној површини $\phi = y$. Пошто се облик слободне површине не зна, то је њено одређивање посебан задатак у овој области. И овај проблем ћемо покушати да покријемо ФПНМ методом.

4.2.1 Стационарно струјање кроз порозну средину

Дводимензионални стационарни ток кроз порозни медијум је регулисан константном разликом потенцијала на две површине. Проток се јавља између два непропусна слоја у правоугаоној геометрији димензија $2m \times 2m$, као на Сл.4.2.



Слика4.2: Поставка проблема стационарног струјања кроз порозну средину без слободне површине

Имплементација проблема је једноставна и њени најважнији делови се налазе на следећем листингу.

Листинг 4.1: Решење проблема филтрације без слободне површине у 2Д коришћењем SCIANN библиотеке

```

1 # Osnovni grid
2 x_data, y_data = np.meshgrid(
3     np.linspace(0, 2, 201),
4     np.linspace(0, 2, 201)
5 )
6
7 # Modeluje se  $\phi(x, y)$ 
8 x = sn.Variable('x')
9 y = sn.Variable('y')
10 phi = sn.Functional('phi', [x,y], 4*[30], 'sigmoid')
11
12 # %%
13 k = 1.e-5
14 TOL = 0.015
15
16 # Osnovna jednacina
17 fun1 = k * (diff(phi, x, order=2) + diff(phi, y, order=2))
18
19 # Dirihielovi granicni uslovi
20 C1 = (1-sign(x - (0+TOL))) * (phi-2)
21 C2 = (1+sign(x - (2-TOL))) * (phi-1)
22
23 # Nojumanovi granicni uslovi
24 N1 = (1-sign(y - (0+TOL))) * diff(phi,y)
25 N2 = (1+sign(y - (2-TOL))) * diff(phi,y)
26
27 # FZNN model
28 m2 = sn.SciModel([x,y], [fun1, C1, C2, N1, N2], optimizer='Adam')
29
30 # Trenin
31 pinn_model = m2.train([x_data, y_data], 5*['zero'], learning_rate=0.001, batch_
    ↪size=1024, epochs=100, stop_loss_value=1E-15)

```

Са свим овим поставкама смо се мање-више већ сретали, осим што до сада нисмо имали 2Д стационарни проблем. Постављамо равномерни грид колокационих тачака у димензијама домена ($2m \times 2m$), затим дефинишем функционал $\Phi(x, y)$ и диференцијалну једначину проблема. Приметимо да решење уопште не би требало да зависи од коефицијента k . Следећи корак је поставка Дирихелових граничних услова на левом ($\Phi_1 = 2m$) и на десном ($\Phi_1 = 1m$) крају домена, тј. на вертикалама $x_1 = 0m$ и $x = 2m$, респективно:

```

C1 = (1-sign(x - (0+TOL))) * (phi-2)
C2 = (1+sign(x - (2-TOL))) * (phi-1)

```

Недостају само још Нојманови гранични услови који јамче да су доња ($y = 0$) и горња ($y = 2m$) површина непропусне, тј. да је извод потенцијала по y једнак нули:

```

N1 = (1-sign(y - (0+TOL))) * diff(phi,y)
N2 = (1+sign(y - (2-TOL))) * diff(phi,y)

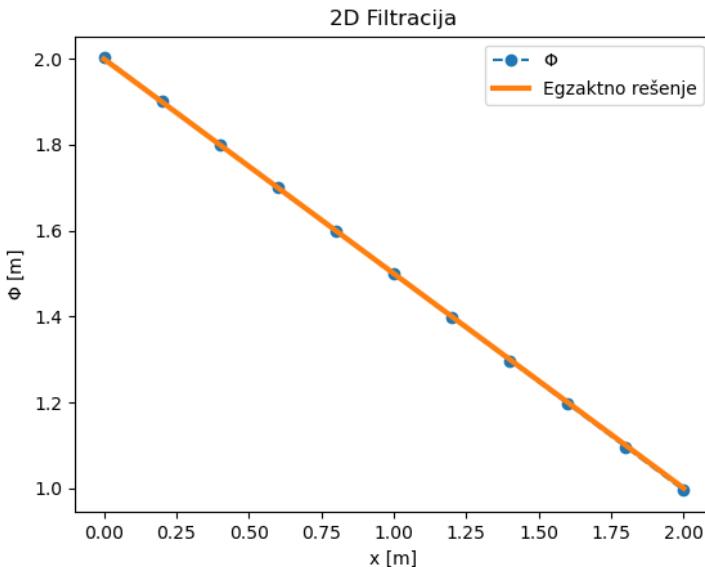
```

Када се постави проблем, решење се назире већ за неколико десетина епоха тренирања.

Аналитичко решење за потенцијал је, према Bear [Bea12]:

$$\Phi = \Phi_1 - \frac{\Phi_1 - \Phi_2}{L}(x - x_1)$$

где је $L = x_2 - x_1$. Дакле, потенцијално поље је једнолично у односу на Y координату, док је градијент потенцијала константан у правцу X осе.



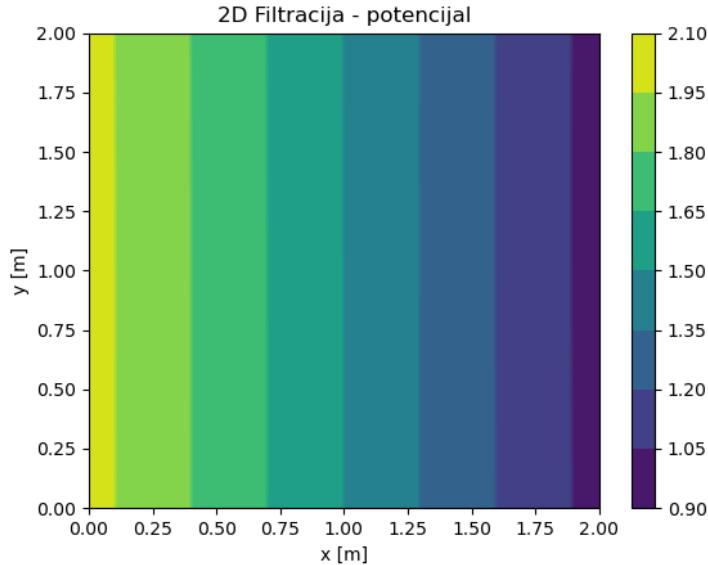
Слика4.3: ФПНМ решење потенцијала дуж X осе за 2Д случај филтрације без слободне површине.

Поређење аналитичког и ФПНМ решења је приказано на Сл.4.3, а потенцијално поље је приказано на Сл.4.4. Униформност потенцијалног поља у Y смеру, показује тачност 2Д ФПНМ решења за овај стационарни проблем.

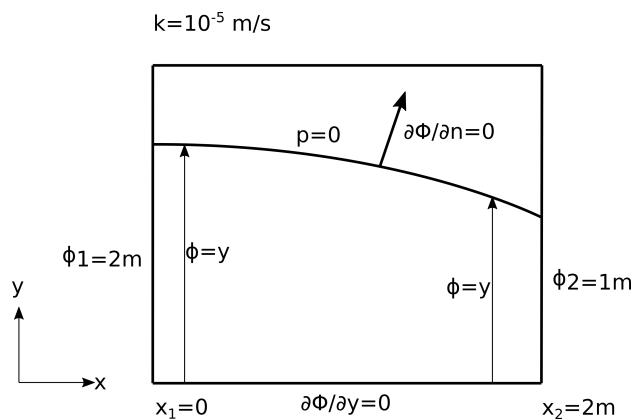
4.2.2 Стационарно струјање кроз порозну средину са слободном површином

Стационарни ток кроз порозни медијум, са слободном површином је регулисан константном разликом потенцијала на две супротне површине, као што је приказано на Сл.4.5. Доња површина је непропусна. Геометријски и материјални подаци, као и гранични услови, та које су дати на Сл.4.5.

Вредности потенцијала у колокацијоним тачкама на површини $x_1 = 0$ су $\Phi_1 = 2m$ док су у тачкама на линији $x_2 = 2m$ вредности $\Phi_2 = 1m$. Дакле, и Дирихлеови и Нојманови гранични услови су идентични као и претходни пример који није укључивао постојање слободне површине. Међутим, њено постојање је физички нужно и дефинисано условима (4.2).



Слика4.4: ФПНМ решење поља потенцијала за 2Д случај филтрације без слободне површине.



Слика4.5: Поставка проблема стационарног струјања кроз порозну средину са слободном површином

Како бисмо имплементирали овај гранични услов, морамо да израчунамо правац нормале:

```
k1 = diff(phi,x)
alpha = atan(k1)+np.pi/2
nx = cos(alpha)
ny = sin(alpha)
```

који ћемо добити тиме што додамо угао $\frac{\pi}{2}$ правцу тангенте на ϕ , коју израчунавамо захваљујући тривијалној доступности првог извода у ФПНМ методологији. Након тога лако израчунавамо компоненте нормале nx и ny . Гранични услов слободне површине постављамо на исти начин као и раније када смо користили библиотеку SCIAANN, тј. у виду којункције се наводи где услов важи и шта у том делу домена важи. Међутим, овог пута немамо строго дефинисане координате, јер положај слободне површине не знамо. Оно што знамо је да је на читавој слободној површини $\Phi = y$, па ово наводимо као област важења:

```
FS1 = (abs(y-phi)<0.009) * k * (diff(phi,x)*nx + diff(phi,y)*ny)
```

док услов непостојања протока кроз слободну површину $\frac{\partial \Phi}{\partial n} = \frac{\partial \Phi}{\partial x} n_x + \frac{\partial \Phi}{\partial y} n_y = 0$ наводимо као главну компоненту.

Потребно је обезбедити и довољан број колокационих тачака да би се исправно испратио облик слободне површине. То ћемо обезбедити тако што у делу домена у коме очекујемо појаву слободне површине концентрацију колокационих тачака повећамо (у нашој имплементацији) четири пута. Како је у питању чисто техничко решење, овде се тиме нећемо бавити, већ читаоца упућујемо на комплетан пример.

Аналитичко решење за потенцијал за овај једноставан проблем се по Bear [Bea12], може се написати у облику

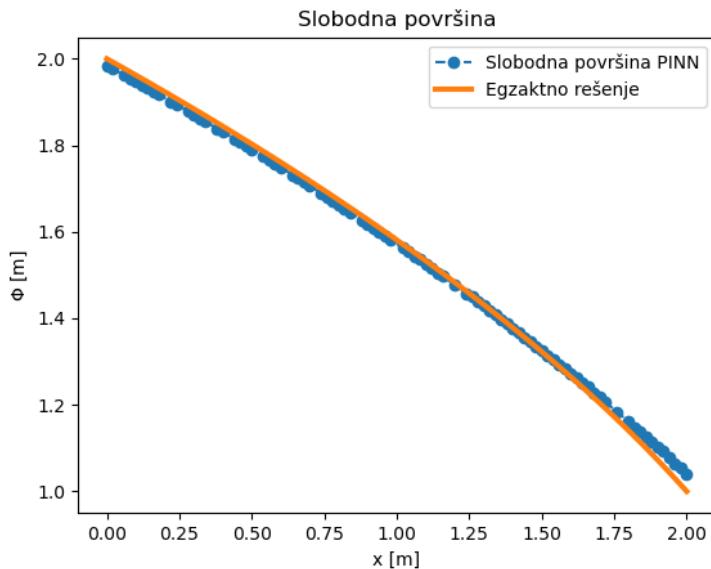
$$\Phi = \sqrt{\Phi_1^2 - 2B(x - x_1)},$$

где је

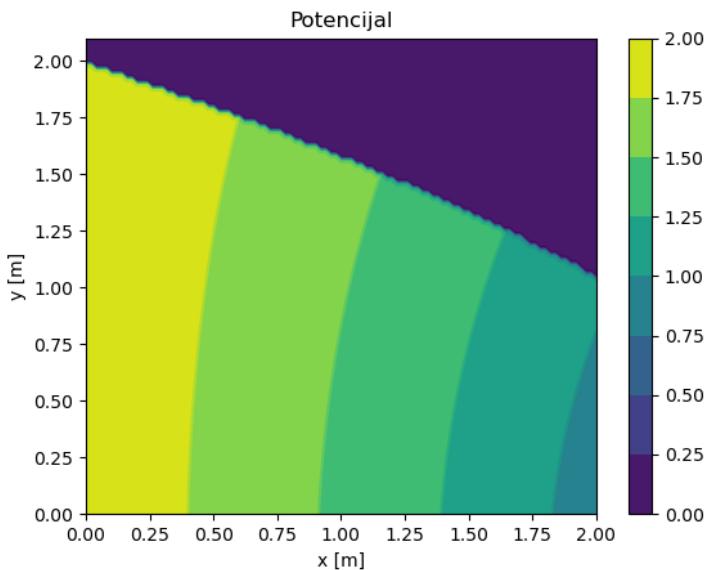
$$B = \frac{\Phi_1^2 - \Phi_2^2}{2L}$$

и $L = x_2 - x_1$. Дакле, као и код проблема без слободне површине, потенцијално поље је униформно у односу на Y координату, тј. не зависи од ње под условом да се налазимо испод слободне површине. Поређење ФПНМ решења са аналитичким решењем може се видети на Сл.4.6. Поље потенцијала је приказано на Сл.4.7.

Може се приметити релативно добро слагање ФПНМ решења са аналитичким решењем, као и очигледна разлика распореда поља потенцијала у односу на случај без слободне површине приказан на Сл.4.4. Ако пак упоредимо приступ решавању проблема слободне површине ФПНМ методом и класичном методом коначних елемената код Којић [Kojic98], можемо приметити да је ФПНМ приступ једноставнији. Разлог томе је што се код ФПНМ не захтева никакав посебан нумерички третман и употреба нумеричких претпоставки, већ се физика проблема директним путем преводи у ФПНМ гранични услов.



Слика4.6: ФПНМ решење потенцијала дуж X осе за 2Д случај филтрације са слободном површином.



Слика4.7: ФПНМ решење поља потенцијала за 2Д случај филтрације са слободном површином.

4.3 Пропагација поплавног таласа у отвореном каналу

Управљање водним ресурсима захтева алате за дугорочно и краткорочно предвиђање различитих хидролошких феномена. Предвиђање је значајно у областима управљања ризиком од поплава, управљања хидро-електрана, унутрашње пловидбе, водоснабдевања, итд. Бројни задаци у вези са предвиђањем хидролошких података успешно су обрађени коришћењем приступа предвиђања заснованог на класичном нумеричком моделовању. Иако овај приступ даје задовољавајуће резултате, још увек има много проблема које треба решити. У многим случајевима, време рачунања је параметар који може ограничити примену физички засnovаних хидролошких и хидрауличких модела у реалној примени. Поред тога, физички засновани модели су недовољно флексибилни у случајевима инверзних проблема, идентификације параметара, асимилације мерених података итд.

Модели предвиђања засновани на неуронским мрежама примењени су у моделовању падавина и отицаја Chadalawada *et al.* [CHB20], системима раног упозорења на поплаве Duncan *et al.* [DCK+13], као и моделовању урбаних водоводних мрежа Garzón *et al.* [GarzonKLT22]. Ови приступи захтевају велику количину података за обуку, што може створити проблеме уколико нема довољно података. Поред тога, може се приметити да ова врста модела није у стању да испоручи добре резултате када улазни подаци излазе изван опсега података који се користе за обуку. У овим случајевима, модели засновани на неуронским мрежама могу произвести физички немогуће резултате. У последњих неколико година, убрзано се уводи ФПНМ приступ и у овој области. У наредном примеру ћемо сагледати потенцијал употребе ФПНМ у једнодимензионом проблему пропагације поплавног таласа у отвореним каналима спајањем физичког закона са почетним и граничним условима описаним кинематичком једначином пропагације таласа.

4.3.1 Усмеравање поплава - физички закони

Ширење поплавног таласа у отвореним каналима је описано помоћу две једначине, и то **једначином континуитета** (4.3) и **законом одржања количине кретања** (4.4). Ова једначина садржи утицаје трења, гравитације, сile притиска, као и локалног и конвективног убрзања. У овом примеру, усмеравање поплава у правоугаоном каналу је представљено кинематичким таласом који поједностављује динамичку једначину узимајући само утицај трења и гравитације:

$$\frac{\partial h(x, t)}{\partial t} + c \frac{\partial h(x, t)}{\partial x} = 0 \quad (4.3)$$

$$Q(x, t) = \frac{1}{n} \cdot B \cdot h(x, t)^{\frac{5}{3}} \cdot \sqrt{I_d}, \quad (4.4)$$

где $h [m]$ представља дубину воде, $t [s]$ је време, $x [m]$ просторну координату, $c [m/s]$ брзину пропагације поремећаја, $Q [m^3/s]$ даје проток (пражњење), $n [m^{-\frac{1}{3}} s]$ представља Манингову храпавост, $B [m]$ ширину попречног пресека и I_d нагиб по уздужној оси.

Циљ моделовања усмеравања поплава је процена промене дубине воде дуж канала $h(x, t)$ која је изазвана поплавним таласом представљеним хидрографом тока $Q_{in}(t) = Q(0, t)$ на узводној граници.

4.3.2 Конструкција функције губитка

Ако се осврнемо на општи израз за функцију губитка (1.2), видимо три компоненте, и то губитак који потиче од резидуалне мреже, губитак који потиче од почетних услова и губитак који потиче од граничних услова. Кренимо редом. Компонента функције губитка која потиче од диференцијалне једначине (4.3) дефинисана је десном страном исте једначине. Компонента почетних услова се дефинише као:

$$h(x, t = 0) = h_0$$

Поред тога, потребан нам је и гранични услов који дефинише вредност висине таласа на почетку моделованог домена, тј. на $x = 0$ одакле талас долази. Тај гранични услов изводимо из динамичке једначине (4.4):

$$h(0, t) = \left(\frac{Q_{in}(t) \cdot n}{B \cdot \sqrt{I_d}} \right)^{\frac{3}{5}}.$$

Уколико као меру грешке усвојимо средњу квадратну грешку (*Mean Squared Error - MSE*), композитна функција губитка изгледаће овако:

$$MSE = MSE_r + MSE_0 + MSE_b,$$

где су:

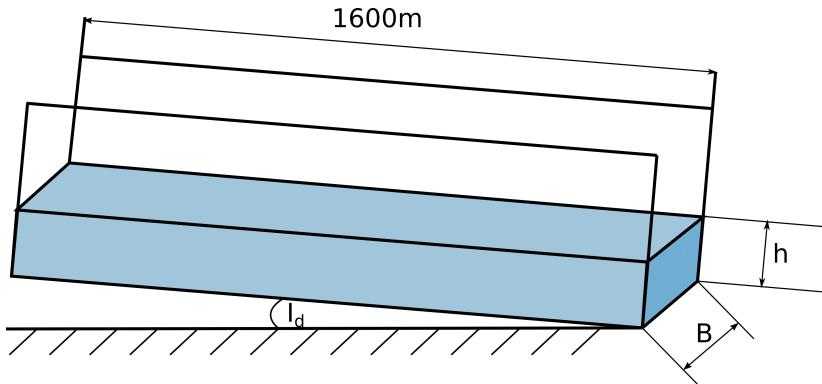
$$\begin{aligned} MSE_r &= \frac{1}{N_{x_f, t_f}} \sum |r(x_f, t_f)|^2, \\ MSE_0 &= \frac{1}{N_{x_0, t_0}} \sum |\tilde{h}(x_0, 0) - h(x_0, 0)|^2, \\ MSE_b &= \frac{1}{N_{x_b, t_b}} \sum |\tilde{h}(0, t_b) - h(0, t_b)|^2. \end{aligned}$$

Овде су N_{x_f, t_f} , N_{x_0, t_0} и N_{x_b, t_b} укупни бројеви колокационих тачака у унутрашњости моделованог домена, за почетне и за граничне услове, респективно.

4.3.3 Тест пример и имплементација

Пример на коме ћемо тестирати ваљаност нашег ФПНМ приступа за моделовање ширења поплавног таласа је пропагација таласа дуж канала дугог 1600 метара, облика призме и правоугаоног попречног пресека широког 15 метара, као на Сл.4.8.

Манингова храпавост има вредност од $n = 0,015m^{-\frac{1}{3}}s$, нагиб је постављен на $I_d = 0,005$, а брзина пропагације на $c = 15 m/s$. Циљ је израчунати промене дубине и протока воде дуж



Слика4.8: Поставка проблема пропагације поплавног таласа у времену.

канала, изазване поплавним таласом генерисаним као узводни гранични услов на $x = 0$:

$$Q_{in}(t) = Q(0, t) = 180 \cdot \left[1 + \left(-\frac{\operatorname{sgn}(t - 600)}{2} + \frac{1}{2} \right) \cdot \sin\left(\frac{\pi t}{600}\right) \right].$$

Поред овог услова, ту је и почетни услов Дирихлеовог типа, а то је да је висина воде у каналу $h(x, t = 0) = 1,751m$. Интересантни делови решења приказани су на [Листинг 4.2](#).

Листинг 4.2: Решење проблема пропагације поплавног таласа коришћењем DeepXDE оквира

```

1 import deepxde as dde
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from deepxde.backend import tf
6
7 c = 15 # brzina propagacije talasa
8 n = 0.015 # hraptavost kanala
9 Id = 0.005 # nagib dna kanala
10 B = 15 # poprecni presek
11 length = 1600
12 total_time = 1000.0
13
14 # Hiperparametri
15 layers = [2] + [30] * 4 + [1]
16 activation = 'tanh'
17 initializer = 'Glorot uniform'
18 optimizer = 'rmsprop'
19 batch_size = 128
20 num_of_epochs = 20000
21 learning_rate = 0.001
22 loss = 'mse'
23
24 # Jednacina kontinuiteta
25 def pde(x, h):
26     dh_t = dde.grad.jacobian(h, x, i = 0, j = 1)
27     dh_x = dde.grad.jacobian(h, x, i = 0, j = 0)

```

(наставак на следећој страни)

(настављено са претходне стране)

```

28     return dh_t + c * dh_x
29
30 # Da li je t=0?
31 def initial_h(x, on_boundary):
32     return on_boundary and np.isclose(x[1], 0)
33
34 # Da li je x=0?
35 def boundary_hx0(x, on_boundary):
36     return on_boundary and np.isclose(x[0], 0)
37
38 # Pocetni uslov za visinu vode x(t=0)
39 def func_init_h(x):
40     return 1.751
41
42 # Dirihirov granicni uslov - Profil poplavnog talasa u vremenu
43 def func_hx0(x):
44     t = x[:, 1:2]
45
46     Qin = 180 * (1 + (-np.sign(t - 600) / 2) + 0.5) * np.sin(t * np.pi / 600))
47     a = Qin * n
48     b = B * np.sqrt(Id)
49     c = a / b
50     return custom_pow(c, 3/5)
51
52 time_domain = dde.geometry.TimeDomain(0, total_time)
53 geom_domain = dde.geometry.Interval(0, length)
54 geotime = dde.geometry.GeometryXTime(geom_domain, time_domain)
55
56 # Realizacija granicnog i pocetnog uslova
57 bc = dde.icbc.DirichletBC(geotime, func = func_hx0, on_boundary = boundary_hx0)
58 ic = dde.icbc.IC(geotime, func = func_init_h, on_initial = initial_h)
59
60 # Konstrukcija modela i definisanje kolokacionih tacaka
61 data = dde.data.TimePDE(geotime, pde, [bc, ic], num_domain = 16000, num_
62     ↪boundary = 1000,
63     num_initial = 100, train_distribution = 'uniform')
64 net = dde.nn.FNN(layers, activation, initializer)
65 model = dde.Model(data, net)
66
67 # Treniranje RMSProp metodom
68 model.compile(optimizer = optimizer, loss = loss, lr = learning_rate)
69 loss_history, train_state = model.train(epochs = num_of_epochs, display_every =_
70     ↪1000, batch_size = batch_size)
71
72 # Dodatno treniranje L-BFGS-B metodom posle RMSprop optimizacije
73 model.compile("L-BFGS-B")
74 loss_history, train_state = model.train()

```

У овој скрипти одмах на почетку дефинишемо и физичке параметре проблема и хипер-параметре модела. Погледом на групу хипер-параметара одмах може да се примети значајно већи број епоха за тренинг, као и оптимизатор RMSProp уместо стандардног Adam. Adam приликом рачунања градијента користи и први и други извод (момент), док RMSProp користи само други извод. Током експериментисања са различитим хипер-параметрима, испоставило се да за овај конкретан пример RMSProp заиста нешто брже конвергира. Такође, показало се да је пример у неким сценаријима осетљив чак и на избор batch_size и иницијализатора тежина. Правило каже да је уз активационе функције као што су sigmoid или

\tanh боље користити Glorot иницијализатор, док уз активациону функцију relu боље иде He, по Katanforoosh and Kunin [KK19].

На жалост, око избора хипер-параметара не постоје строга правила. Све зависи од самог примера, па се избор оптималних хиперпараметара за неки конкретан проблем углавном своди на мануелну, временски захтевну процедуру. Помоћу алата као што је *Tensorflow/Keras* може се донекле умањити овај проблем једноставним алгоритмима као што је насумична претрага (*Random Grid Search*), која захтева огромне рачунарске ресурсе да би се добили иоле употребљиви резултати. С друге стране, постоји неколико алата који ову претрагу чине ефикаснијом паметнијим приступом оптимизацији. На пример, алат *Black-Fox*¹¹ користи дистрибуирани генетски алгоритам, а проблем хардверских ресурса решава дистрибуираном обуком на локалном *Kubernetes* кластеру или кластеру постављеном на неком клауд провайдеру.

Следи поставка почетног Дирихлеовог услова за ниво воде у каналу и нешто сложенијег граничног услова за висину воде h који се мења у времену по јединачини (4.4). Овде само треба нагласити да се код DeepXDE улази x и t воде као један двоколонски тензор, у коме је:

```
x[:, 0:1] # ulaz x
x[:, 1:2] # ulaz t
```

Како је у питању динамички проблем који покрива релативно велики просторни и временски домен, тј. прати се линија од 1,6 km током неких 17 минута, потребан је и већи број колокационих тачака него у неким проблемима које смо раније обрађивали. Модел се поставља као:

```
data = dde.data.TimePDE(geotime, pde, [bc, ic], num_domain = 16000, num_
boundary = 1000,
num_initial = 100, train_distribution = 'uniform')
```

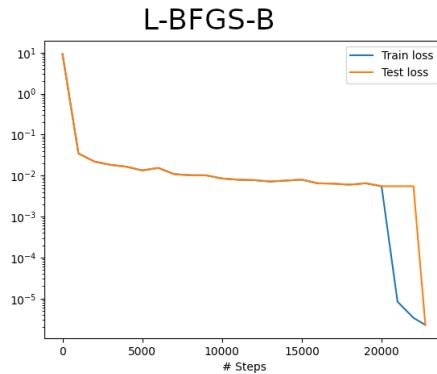
Бројност колокационих тачака за почетне и граничне услове прати бројност тачака унутар домена. Након стандардног тренирања методом RMSProp примећујемо још једну специфичност у односу на једноставније примере. Наиме, након што се обави „глобална” претрага, алгоритам *Limited Memory Broyden-Fletcher-Goldfarb-Shanno* има прилику да се додатно приближи оптималном решењу према Markidis [Mar21]:

```
model.compile("L-BFGS-B")
loss_history, train_state = model.train()
```

Примећујемо да се овде не наводи број епоха, већ се алгоритам ослања на аутоматску детекцију конвергенције. График функције губитка се може видети на Сл.4.9, где до 20.000. епохе, као што је већ речено, тече RMSProp, а онда се у локалној околини наставља са L-BFGS-B. Веома је уочљив раст перформанси тренирања, тј. пад вредности функције губитка у том делу.

Што се самог процеса тренирања ове ФПНМ тиче, треба нагласити да је за овогу количину података, тј. колокационих тачака, тај процес далеко брже ради на графичком процесору

¹¹ <https://blackfox.ai>



Слика4.9: Почетно тренирање методом *RMSProp* у 20.000 епоха и додатно тренирање методом *L-BFGS-B* до детектоване конвергенције

неко на стандардном процесору. Слободна процена је да је тренирање на *Tesla T4* графичком процесору више од 10 пута брже него на процесору *Intel Xeon Silver 4208 @ 2.10GHz*.

Конечно долазимо и до резултата. Висина воденог стуба у више контролних тачака (0, 400m, 800m, 1200m, 1600m) приказана је на [Сл.4.10](#). Ова решења се добро уклапају са решењима које даје метода коначних разлика, али то поређење овде нећемо приказивати.

4.3.4 Инверзни проблем

Пошто смо успешно решили директни проблем, хајде да замислимо ситуацију да нам није познат параметар n који репрезентује Манингову храпавост, али да смо посматрањем крећања таласа утврдили да је његов врх висине 2,65m прошао кроз контролне тачке (0, 400m, 800m, 1200m, 1600m) у следећим тренуцима:

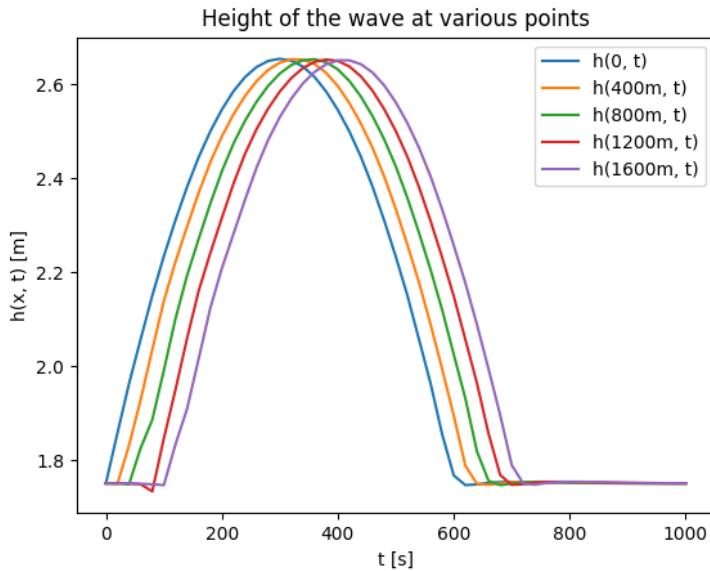
x [m]	0	400	800	1200	1600
t [s]	300	320	360	380	400

Ове опсервације чине могућим креирање *PointSet* граничног услова који смо већ користили, а то се код DeepXDE оквира ради на следећи начин:

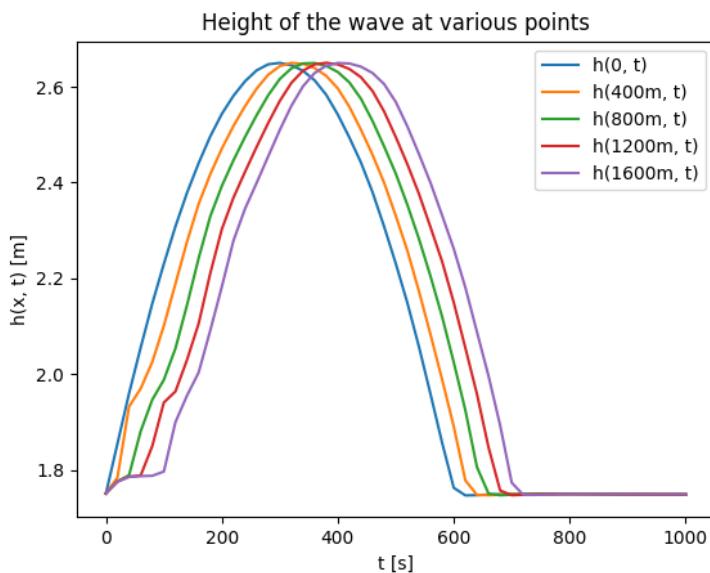
```
bc_x = np.array([[0, 300], [400, 320], [800, 360], [1200, 380], [1600, 400]]).reshape(5, ↴2)
bc_y = np.array([2.65, 2.65, 2.65, 2.65, 2.65]).reshape(5, 1)
ic3 = dde.icbc.PointSetBC(bc_x, bc_y, component=0)
```

Резултати висине воденог стуба су приказани на [Сл.4.11](#), док је вредност параметра n током обуке приказана на [Сл.4.12](#).

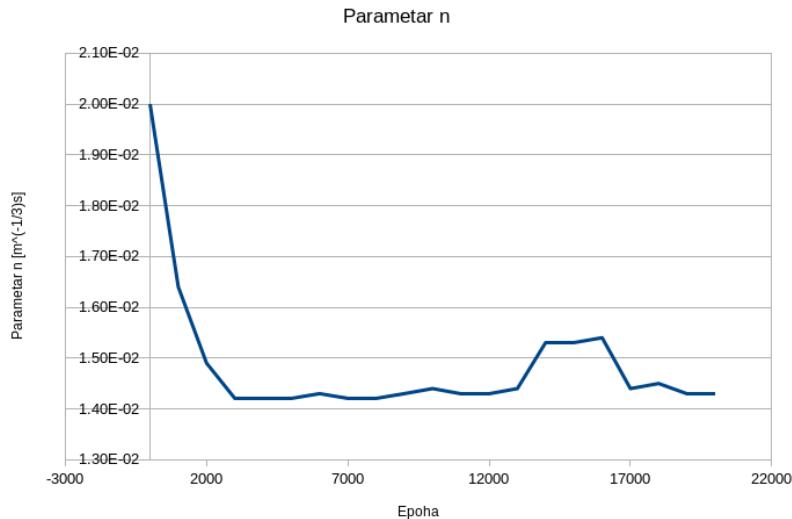
Видимо да се ФПНМ и у овом проблему доста добро сналази са инверзном поставком. Још једном ваља нагласити да се код ФПНМ директни и инверзни приступ методолошки уопште не разликују и да захтевају исту количину рачунарских ресурса. Насупрот томе,



Слика4.10: Висина воденог стуба у неколико тачака током времена.



Слика4.11: Висина воденог стуба код инверзног проблема



Слика4.12: Вредност непознатог параметра n током обуке

класичне нумеричке методе као МКЕ су у стању да реше искључиво директне проблеме. За индентификацију параметара код МКЕ морају да се користе методе за конвексну или чешће, неконвексну оптимизацију и које су у реалним применама рачунарски веома захтевне, а понекад и нерешиве.

Наравно, ни ФПНМ није идеalan. Успешност обуке и овом примеру много зависи од избора хипер-параметара, а често се дешава да услед стохастичког карактера саме обуке ни исти хипер-параметри не доводе до решења баш у сваком тренингу. Поред хипер-параметара, овде имамо и почетну вредност непознатог физичког параметра (или више параметара), па се неретко дешава да оптимизација за неке почетне вредности уопште не конвергира, задржавајући се у неком локалном минимуму.

5

Акустика

5.1 Увод

У овом поглављу бавићемо се могућностима за решавање Хелмхолцове једначине помоћу ФПНМ метода. Хелмхолцова једначина се користи у акустици за описивање вибрација у нехомогеним медијумима, као што су звучни таласи у ваздуху или води. Такође се користи за описивање различитих акустичких појава, као што су резонанција и дифракција звука. Описује како се звук шири, како се одбија и како рефлектује. Корисна је у пројектовању и анализи акустичких система, као што су звучници, микрофони и звучни изолатори. Поред тога, Хелмхолцова једначина се користи и за описивање других таласних појава, рецимо електромагнетних поља, у електромагнетној теорији, електротехнички и физички кондензованог стања, а посебно у новије време за пројектовање оптичких влакана за телекомуникације.

Овде ћемо се ограничити на акустику, и то из једноставног разлога. Да би се ФПНМ исправно обучила, потребна је густина колокационих тачака сразмерна таласној дужини, тј. најмање 10-30 тачака по једној λ . Моделовање електромагнетних појава чија је карактеристична таласна дужина λ за неколико редова величине мања, превазишла би доступне рачунарске ресурсе, и по питању процесорске снаге и по питању количине меморије. Први пример је елементарно простирање таласа у равни. У другом примеру научићемо како да поставимо мало сложенију геометрију и Нојманове граничне услове. У прва два примера користићемо оквир *DeepXDE*. Трећи пример укључује и имагинарни део функције, па ћемо решавати систем две парцијалне диференцијалне једначине са две непознате функције. За овај најкомплекснији пример биће употребљен оквир *SciANN*.

5.2 Решавање на домену облика квадрата

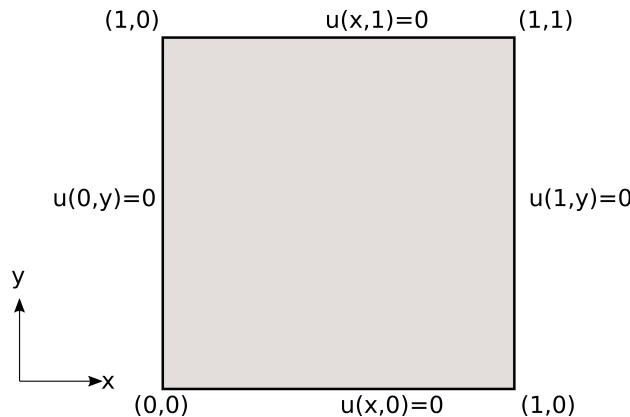
Кренућемо од најједноставнијег дводимензионог случаја стојећег таласа у акустици. За таласни број $k_0 = 2\pi n$ за $n = 2$, треба решити Хелмхолцову (*Helmholtz*) једначину облика:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - k_0^2 u = f, \quad | \Omega = [0, 1]^2,$$

уз Дирихлеове граничне услове

$$u(x, y) = 0, \quad (x, y) \in \partial\Omega$$

приказане на Сл.5.1 и члан који специфицира извор $f(x, y) = k_0^2 \sin(k_0 x) \sin(k_0 y)$.



Слика5.1: Поставка проблема и гранични услови.

Постоји аналитичко решење овог проблема и оно гласи:

$$u(x, y) = \sin(k_0 x) \sin(k_0 y).$$

За више детаља у погледу теоријске позадине диференцијалне једначине и граничних услова читалац може консултовати Ihlenburg [Ihl98]. Решење методом коначних елемената (МКЕ) је такође доступно у оквиру Dolfinx туторијала¹².

5.2.1 Имплементација

Решење директног проблема приказано је на следећем листингу.

¹² https://github.com/FEniCS/dolfinx/blob/main/python/demo/demo_helmholtz.py

Листинг 5.1: Решење проблема простирања стојећег таласа у 2Д ко-
ришћењем DeepXDE библиотеке

```

1 import deepxde as dde
2 import numpy as np
3
4 # Frekvencija
5 n = 2
6
7 precision_train = 10
8 precision_test = 30
9 weights = 100
10 iterations = 10000
11 learning_rate, num_dense_layers, num_dense_nodes, activation = 1e-3, 3, 150,
12     ↪"sin"
13
14 # Uvezi sinus
15 from deepxde.backend import tf
16 sin = tf.sin
17
18 # Osnovna PDE
19 def pde(x, u):
20     du_xx = dde.grad.hessian(u, x, i=0, j=0)
21     du_yy = dde.grad.hessian(u, x, i=1, j=1)
22
23     f = k0 ** 2 * sin(k0 * x[:, 0:1]) * sin(k0 * x[:, 1:2])
24     return -du_xx - du_yy - k0 ** 2 * u - f
25
26 # Egzaktno resenje
27 def func(x):
28     return np.sin(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2])
29
30 # Da li je kol. tacka na granici?
31 def boundary(_, on_boundary):
32     return on_boundary
33
34 # Geometrija jedinicnog kvadrata
35 geom = dde.geometry.Rectangle([0, 0], [1, 1])
36 # Talasni broj
37 k0 = 2 * np.pi * n
38 # Talasna duzina
39 wave_len = 1 / n
40
41 hx_train = wave_len / precision_train
42 nx_train = int(1 / hx_train)
43
44 hx_test = wave_len / precision_test
45 nx_test = int(1 / hx_test)
46
47 # Dirihleov granicni uslov y=0 na granicama
48 bc = dde.icbc.DirichletBC(geom, lambda x: 0, boundary)
49
50 data = dde.data.PDE(
51     geom,
52     pde,
53     bc,
54     num_domain=nx_train ** 2,
55     num_boundary=4 * nx_train,

```

(наставак на следећој страни)

(настављено са претходне стране)

```

55     solution=func,
56     num_test=nx_test ** 2,
57 )
58
59 # Mreza i model
60 net = dde.nn.FNN([2] + [num_dense_nodes] * num_dense_layers + [1], activation,
61   ↪"Glorot uniform")
62 model = dde.Model(data, net)
63
64 # Forsiraj vece tezine za granicne uslove nego za unutrasnjost domena
65 loss_weights = [1, weights]
66
67 model.compile("adam", lr=learning_rate, metrics=["l2 relative error"], loss_
68   ↪weights=loss_weights)
69
70 losshistory, train_state = model.train(iterations=iterations)
71 dde.saveplot(losshistory, train_state, issave=True, isplot=True)

```

Након стандардног импорта одговарајућих модула, почињемо спецификацијом општих параметара. Овај пример има пар специфичности у односу на остале. Наиме, да би се успешно моделовале таласне појаве помоћу ФПНМ, густина колокационих тачака мора да буде директно пропорционална фреквенцији. Што је виша фреквенција n , мања је таласна дужина $wave_len$, па је потребно више колокационих тачака да покрије домен. Овде смо узели 10 колокационих тачака по таласној дужини током тренинга и 30 тачака по таласној дужини у тест скупу.

```

# Frekvencija talasa
n = 2
precision_train = 10
precision_test = 30
weights = 100
learning_rate, num_dense_layers, num_dense_nodes, activation = 1e-3, 3, 150,
  ↪"sin"

```

Такође, видимо да користимо архитектуру са мањим бројем слојева, али са више неурона по слоју, као и активациону функцију $\sin(x)$ која би требало да буде погоднија за опонашање таласних феномена.

Следи спецификација саме парцијалне диференцијалне једначине у облику функције губитка како смо то већ навикли:

```

def pde(x, u):
    du_xx = dde.grad.hessian(u, x, i=0, j=0)
    du_yy = dde.grad.hessian(u, x, i=1, j=1)

    f = k0 ** 2 * sin(k0 * x[:, 0:1]) * sin(k0 * x[:, 1:2])
    return -du_xx - du_yy - k0 ** 2 * u - f

```

Овде користимо услужну функцију `dde.grad.hessian` одабиром координате која се диференцира и којом се диференцира. У овом примеру су гранични услови елементарни, па их овде нећемо посебно наводити.

Геометрија, таласни број $k_0 = 2\pi\nu$ и таласна дужина $\lambda = \frac{1}{\nu}$ дају се као:

```
geom = dde.geometry.Rectangle([0, 0], [1, 1])
k0 = 2 * np.pi * n
wave_len = 1 / n
```

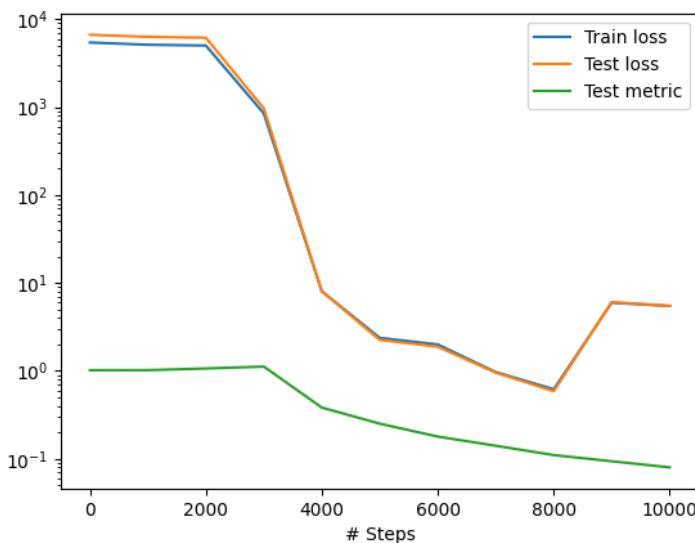
Једина специфичност коју додатно треба нагласити је да понекад треба форсирати поштовање граничних услова тиме што ћемо члану функције губитка који се односи на Дирихлеов гранични услов добити већу тежину у односу на члан који се односи на диференцијалну једначину.

```
weights = 100
loss_weights = [1, weights]
model.compile("adam", lr=learning_rate, metrics=["l2 relative error"], loss_
    ↴weights=loss_weights)
```

Да би се избегао овај корак који са собом носи експериментисање са различитим вредностима тежинског фактора, гранични услов се код *DeepXDE* може задавати и директном трансформацијом функције губитка, али овде се тиме нећемо бавити.

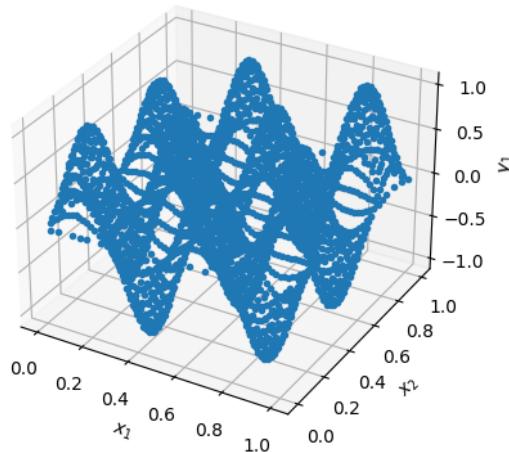
5.2.2 Резултати

Након 10.000 епоха обучавања оптимизационом меотодом Adam који је протекао као што је приказано на Сл.5.2, добијамо стојећи талас чији 3Д приказ можемо видети на Сл.5.3.



Слика5.2: Ток обука ФПНМ

Мера грешке модела RMSE (Root Mean Squared Error) износи $7,98 \cdot 10^{-2}$. Уз обраћање посебне пажње на форсирање граничних услова, затим архитектуру ФПНМ и најзад тип



Слика5.3: Тродимензиони приказ таласа у домену облика квадрата

активационе функције, успели смо да добијемо прилично добро решење. Читалац може самостално да проба како би промена фреквенције (а самим тим и таласне дужине), густине колокацијних тачака, архитектуре, утицала на процес обучавања модела.

Овде можемо дати и кратку препоруку **како приступити моделовању сложенијих појава**, са сложенијом геометријом и комплекснијим граничним условима. Пошто ФПНМ решавање зависи од већег броја хипер-параметара, препорука је да се прво реши до краја поједностављен проблем базиран на истој диференцијалној једначини, али са једноставнијом геометријом и граничним условима. Када се стекне слика о томе која комбинација хипер-параметара води до конвергенције решења, онда је лакше приступити главном, комплексном проблему. С друге стране, постоји неколико алата који претрагу хипер-параметара чине ефикаснијом, као већ поменути [BlackFox¹³](https://blackfox.ai) који користи дистрибуирани генетски алгоритам.

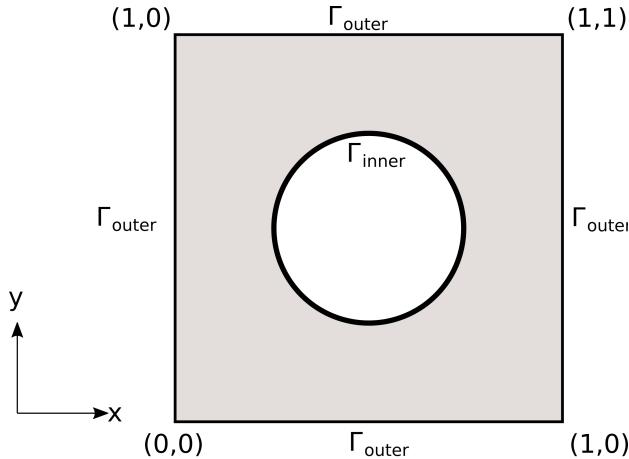
¹³ <https://blackfox.ai>

5.3 Решавање на домену облика квадрата са шупљином

У односу на претходни пример [Решавање на домену облика квадрата](#) (страна 62) дођајемо шупљину у средини квадратног домена и прописујемо одговарајуће Нојманове граничне услове на ободу шупљине. Да се подсетимо, домен проблема Ω је квадрат странице L , $L = 1$, из кога искључујемо круг полупречника $R = \frac{1}{4}$. За таласни број $k_0 = 2\pi n$ и $n = 1$, решавамо Хелмхолцово једначину:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - k_0^2 u = f, \quad | \Omega,$$

где је члан који специфицира извор $f(x, y) = k_0^2 \sin(k_0 x) \sin(k_0 y)$. Облик домена може се видети на Сл.5.4.



Слика5.4: Поставка проблема и гранични услови

Постоји аналитичко решење овог проблема и оно гласи:

$$u(x, y) = \sin(k_0 x) \sin(k_0 y).$$

Овде ћемо специфицирати Дирихлеове граничне услове према аналитичком решењу на спољној граници домена Ω коју означавамо са Γ_{outer} :

$$u(x, y) | \Gamma_{outer} = \sin(k_0 x) \sin(k_0 y), \quad (x, y) \in \Gamma_{outer}$$

На сличан начин, можемо да дефинишемо гранични услов на унутрашњој граници, овог пута Нојманов:

$$(\nabla u | \Gamma_{inner} \cdot n)(x, y) = [k_0 \cos(k_0 x) \sin(k_0 y), k_0 \sin(k_0 x) \cos(k_0 y)] \cdot n = g(x, y), \quad (5.1)$$

где је n вектор нормале. Концизније написано, Нојманов гранични услов на унутрашњој граници гласи:

$$\nabla u | \Gamma_{inner} \cdot n = g.$$

5.3.1 Имплементација

На следећем листингу су дати главни детаљи имплементације. Намерно су изостављени делови који су ирелевантни за само решавање, као што је цртање дијаграма. Целокупна скрипта се, као и остале, налази у репозиторијуму са примерима.

Листинг 5.2: Решење проблема простирања стојећег таласа у 2Д до-
мену са шупљином

```

1 import deepxde as dde
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from deepxde.backend import tf
5 sin = tf.sin
6
7 # Opsti parametri
8 n = 2
9 length = 1
10 R = 1 / 4
11
12 precision_train = 15
13 precision_test = 30
14
15 weight_inner = 10
16 weight_outer = 100
17 iterations = 5000
18 learning_rate = 1e-3
19 num_dense_layers = 3
20 num_dense_nodes = 350
21 activation = "sin"
22
23 k0 = 2 * np.pi * n
24 wave_len = 1 / n
25
26 # Parcijalna diferencijalna jednacina
27 def pde(x, y):
28     dy_xx = dde.grad.hessian(y, x, i=0, j=0)
29     dy_yy = dde.grad.hessian(y, x, i=1, j=1)
30     f = k0**2 * sin(k0 * x[:, 0:1]) * sin(k0 * x[:, 1:2])
31     return -dy_xx - dy_yy - k0**2 * y - f
32
33 # Egzaktno resenje
34 def func(x):
35     return np.sin(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2])
36
37 # Da li je tacka na granici?
38 def boundary(_, on_boundary):
39     return on_boundary
40
41 # Njumanovi granicni uslovi prema egzaktnom resenju
42 def neumann(x):
43     grad = np.array([
44         k0 * np.cos(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2]),
45         k0 * np.sin(k0 * x[:, 0:1]) * np.cos(k0 * x[:, 1:2]),])
46
47     normal = -inner.boundary_normal(x)
48     normal = np.array([normal]).T

```

(наставак на следећој страни)

(настављено са претходне стране)

```

49     result = np.sum(grad * normal, axis=0)
50     return result
51
52 # Geometrija
53 outer = dde.geometry.Rectangle([-length / 2, -length / 2], [length / 2, length /
54   ↪ 2])
55 inner = dde.geometry.Disk([0, 0], R)
56
57 # Da li je tacka na spoljnoj granici?
58 def boundary_outer(x, on_boundary):
59     return on_boundary and outer.on_boundary(x)
60
61 # Da li je tacka na unutrasnjoj granici?
62 def boundary_inner(x, on_boundary):
63     return on_boundary and inner.on_boundary(x)
64
65 # Iskljuci krug iz kvadrata
66 geom = outer - inner
67
68 hx_train = wave_len / precision_train
69 nx_train = int(1 / hx_train)
70
71 hx_test = wave_len / precision_test
72 nx_test = int(1 / hx_test)
73
74 # Na unutrasnjoj granici Njuman, na spoljnoj Dirihleovi
75 bc_inner = dde.icbc.NeumannBC(geom, neumann, boundary_inner)
76 bc_outer = dde.icbc.DirichletBC(geom, func, boundary_outer)
77
78 data = dde.data.PDE(
79     geom,
80     pde,
81     [bc_inner, bc_outer],
82     num_domain=nx_train**2,
83     num_boundary=16 * nx_train,
84     solution=func,
85     num_test=nx_test**2,
86 )
87
88 net = dde.nn.FNN(
89     [2] + [num_dense_nodes] * num_dense_layers + [1], activation, "Glorot-
90   ↪uniform"
91 )
92
93 model = dde.Model(data, net)
94
95 loss_weights = [1, weight_inner, weight_outer]
96 model.compile("adam", lr=learning_rate, metrics=["l2 relative error"], loss-
97   ↪weights=loss_weights)
98
99 losshistory, train_state = model.train(iterations=iterations)

```

Користићемо *Tensorflow* као *backend* у свим примерима, али треба имати у виду да оквир *DeepXDE* подржава и *PyTorch* и још неке. Након стандардне спецификације општих параметара и хипер-параметара, као у примеру *Решавање на домену облика квадрата* (страница 62), уз једину модификацију додавања нешто више неурона по слоју (350), дефинишемо

Нојманов гранични услов према једначини (5.1):

```
def neumann(x):
    grad = np.array([
        k0 * np.cos(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2]),
        k0 * np.sin(k0 * x[:, 0:1]) * np.cos(k0 * x[:, 1:2]),])

    normal = -inner.boundary_normal(x)
    normal = np.array([normal]).T
    result = np.sum(grad * normal, axis=0)
    return result
```

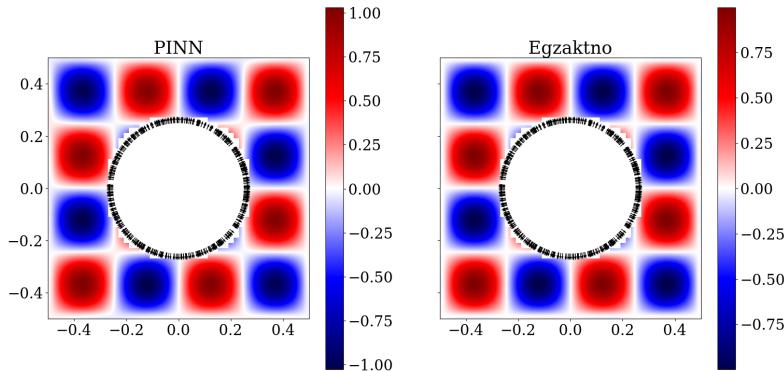
Као што се види из кода, постоје услужне функције које рачунају нормале на правилне границе у колокационим тачкама. Пондерске тежине граничних услова у обуци `weight_inner` и `weight_outer` такође спадају у неку врсту хипер-параметара, па и њима треба посветити пажњу уз неколико мануелних проба. Даље, следи спецификација геометрије проблема као разлике квадрата и диска:

```
outer = dde.geometry.Rectangle([-length / 2, -length / 2], [length / 2, length /
    ↪ 2])
inner = dde.geometry.Disk([0, 0], R)
geom = outer - inner
```

Остатак скрипте је сличан примеру без шупљине [Решавање на домену облика квадрата](#) (страна 62), па га нећемо додатно појашњавати. Довољно је рећи да пажњу треба обратити да буде довољно колокационих тачака на спољној и на унутрашњој граници.

5.3.2 Резултати

Добијени резултати су приказани у форми контурног графика на Сл.5.5. Око унутрашње границе приказани су правци вектора нормала.



Слика 5.5: Резултати примера квадратног домена са шупљином

Мера релативне грешке модела износи 0,048. Уз обраћање посебне пажње на форсирање граничних услова, затим архитектуру ФПНМ и најзад тип активационе функције, успели смо да добијемо прилично добро решење. Читалац може самостално да проба како би промена фреквенције (а самим тим и таласне дужине), густине колокационих тачака, архитектуре, утицала на процес обуке модела.

5.4 Дводимензиони проблем са сочивом

Овај одељак се бави нешто модификованим примером из туторијала¹⁴ за познати софтверски оквир за рад са методом коначних елемената **Deal.II**. Првобитна сврха овог примера је да симулира својства фокусирања ултразвучног таласа који генерише сочиво претварача са променљивом геометријом. Савремене примене у медицинском имицингу користе ултразвучне таласе не само за сврхе снимања, већ и за изазивање одређених локалних ефеката у материјалу, као што су промене у оптичким својствима, које се затим могу мерити другим техникама снимања. Витални састојак ових метода је способност фокусирања интензитета ултразвучног таласа у одређеном делу материјала, идеално у тачки, како би се могла испитати својства материјала на тој локацији.

Карakterистична таласна дужина ултразвука нешто мања од феномена које смо до сада моделовали ФПНМ методом, те би нам био потребан изузетно велики број колокационих тачака што продужава тренирање. Зато смо таласну дужину нешто повећали (смањили фреквенцију), док остале параметре нисмо мењали.

Како бисмо извели једначине за овај проблем, звук узимамо као талас којим се шири промена притиска:

$$\frac{\partial^2 U}{\partial t^2} - c^2 \Delta U = 0,$$

где је c брзина звука, која се због једноставности узима као константа, $U = U(x, t)$, $x \in \Omega$, $t \in \mathbb{R}$. Поставка проблема дата је на Сл.5.6.

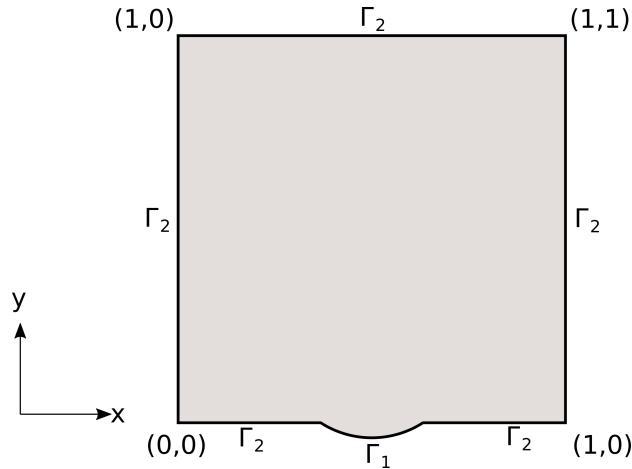
Граница $\Gamma = \partial\Omega$ подељена је на два дела, и то Γ_1 и $\Gamma_2 = \Gamma \setminus \Gamma_1$, где Γ_1 представља сочиво, а Γ_2 апсорбујућу границу. Заправо, желимо да направимо такав гранични услов на Γ_2 тако да се опонаша знатно већи домен. На Γ_1 , претварач генерише таласе константне фреквенције $\omega > 0$ и константне јединичне амплитуде:

$$U(x, t) = \cos \omega t, \quad x \in \Gamma_1$$

Пошто нема других (интерних или граничних) извора и пошто само извор еmitује таласе фреквенције ω , дозвољено је да извршимо раздавање променљивих $U(x, t) = \operatorname{Re}(u(x) e^{i\omega t})$. Комплексна функција $u(x)$ описује просторну зависност амплитуде и фазе (релативно у односу на извор) таласа фреквенције ω , док је амлитуда величина која нас интересује. Ако овако формулисану функцију уврстимо у таласну једначину, видимо да за u имамо

$$\begin{aligned} -\omega^2 u(x) - c^2 \Delta u(x) &= 0, & x \in \Omega, \\ u(x) &= 1, & x \in \Gamma_1. \end{aligned}$$

¹⁴ https://www.dealii.org/current/doxygen/deal.II/step_29.html



Слика5.6: Поставка проблема и гранични услови

Да бисмо нашли одговарајуће граничне услове на Γ_2 који опонашају апсорбујућу границу, размотримо талас облика $V(x, t) = e^{i(k \cdot x - \omega t)}$ фреквенције ω који се простире у правцу $k \in \mathbb{R}^2$. Да би V био решење таласне једначине, мора да важи $|k| = \frac{\omega}{c}$. Претпоставимо да талас долази до $x_0 \in \Gamma_2$ под правим углом, на пример $n = \frac{k}{|k|}$ где n означава нормалу на $\Omega \in x_0$. Онда у x_0 , овај талас задовољава једначину

$$c(n \cdot \nabla V) + \frac{\partial V}{\partial t} = (i c |k| - i \omega) V = 0.$$

Постављањем граничног условия

$$c(n \cdot \nabla U) + \frac{\partial U}{\partial t} = 0, \quad x \in \Gamma_2,$$

таласи који ударају у границу Γ_2 под правим углом биће савршено апсорбовани. Са друге стране, они делови таласног поља који не падају под правим углом не задовољавају овај услов, па ће долазити до парцијалних рефлексија. У основи, директни делови таласа ће проћи кроз границу као да она не постоји, док ће остали бити рефлектовани назад у домен.

Уколико смо спремни да прихватимо овако предложену апроксимацију, онда за u важи следеће:

$$\begin{aligned} -\omega^2 u - c^2 \Delta u &= 0, & x \in \Omega, \\ c(n \cdot \nabla u) + i \omega u &= 0, & x \in \Gamma_2, \\ u &= 1, & x \in \Gamma_1. \end{aligned} \tag{5.2}$$

препознајемо Хелмхолцову једначину са Дирихлеовим условом на Γ_1 и мешаним граничним условом на Γ_2 . Због услова на Γ_2 не можемо да третирамо реалане и имагинарне делове u посебно. Оно што можемо да урадимо је да формирамо систем од две парцијалне диференцијалне једначине у којима фигуришу реални и имагинарни део u , са граничним условима на Γ_2 које везују ове две компоненте. Ако означимо да је $v = \operatorname{Re} u$, $w = \operatorname{Im} u$,

систем (5.2) гласи:

$$\begin{aligned}
 -\omega^2 v - c^2 \Delta v &= 0 \\
 -\omega^2 w - c^2 \Delta w &= 0 \\
 x \in \Omega, \\
 c(n \cdot \nabla v) - \omega w &= 0 \\
 c(n \cdot \nabla w) + \omega v &= 0 \\
 x \in \Gamma_2, \\
 v &= 1 \\
 w &= 0 \\
 x \in \Gamma_1.
 \end{aligned} \tag{5.3}$$

Дакле, прве две једначине важе у целом домену Ω , друге две на граници Γ_2 , а последње две на Γ_1 . Овде први пут имамо систем диференцијалних једначина, али ни то не би требало да буде проблем за ФПНМ приступ, ако подразумевамо да је систем затворен, тј. једнозначен.

5.4.1 Имплементација

На основу система једначина (5.3) треба да формирамо композитну функцију губитка, да формирамо ФПНМ мрежу и да је истренирамо на довољном броју колокационих тачака. Ево кључних делова имплементације остварене помоћу оквира SCIANN:

Листинг 5.3: Решење проблема простирања таласа у 2Д домену са сочивом

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import sciann as sn
4 from numpy import pi
5 from sciann.utils.math import diff, sign, sin, sqrt
6
7 # Brzina talasa
8 c = 1
9 # Frekvencija
10 omega = 2*pi*4
11
12 x = sn.Variable('x')
13 y = sn.Variable('y')
14 v, w = sn.Functional (["v", "w"], [x, y], 3*[200] , 'sin')
15
16 # Diferencijalne jednacine za v i w
17 L1 = -omega**2 * v - c**2 * diff(v, x, order=2) - c**2 * diff(v, y, order=2)
18 L2 = -omega**2 * w - c**2 * diff(w, x, order=2) - c**2 * diff(w, y, order=2)
19
20 TOL = 0.015
21
22 # Dirihelev uslov na G1 (y=0 i 0.4<x<0.6)
23 a,b,c,d = 0.39762422, -1.57715550, -0.03696364, 1.60337246
24 C1 = (1 - sign(y - (a + b*x + c*sqrt(x) + d*x**2 + TOL))) * (1 + sign(x-0.4)) *_

```

(наставак на следећој страни)

(настављено са претходне стране)

```

25 ↪ (1 - sign(x-0.6)) * (1-v)
C2 = (1 - sign(y - (a + b*x + c*sqrt(x) + d*x**2 + TOL))) * (1 + sign(x-0.4)) * ↪
↪ (1 - sign(x-0.6)) * (w-0)

26
27 # Gornja granica G2 (gde je y=1)
C3 = (1+sign(y - (1-TOL))) * (c*diff(v,y) - omega*w )
C4 = (1+sign(y - (1-TOL))) * (c*diff(w,y) + omega*v )

30
31 # Desna granica G2 (gde je x=1)
C5 = (1+sign(x - (1-TOL))) * (c*diff(v,x) - omega*w )
C6 = (1+sign(x - (1-TOL))) * (c*diff(w,x) + omega*v )

34
35 # Leva granica G2 (gde je x=0)
C7 = (1-sign(x - (0+TOL))) * (-c*diff(v,x) - omega*w )
C8 = (1-sign(x - (0+TOL))) * (-c*diff(w,x) + omega*v )

38
39 # Donja granica G2 (gde je y=0) i (x<0.4 or x>0.6)
C9 = (1-sign(y - (0+TOL))) * ((1 - sign(x-0.4)) + (1 + sign(x-0.6))) * (- ↪
↪ c*diff(v,y) - omega*w )
C10 = (1-sign(y - (0+TOL))) * ((1 - sign(x-0.4)) + (1 + sign(x-0.6))) * (- ↪
↪ c*diff(w,y) + omega*v )

42 x_data, y_data = [], []
44
45 kolokacione_tacke = np.genfromtxt('kolokacione_tacke.txt', delimiter=" ")
46
47 for e in kolokacione_tacke:
    ind, x1, y1 = e
    x_data.append(x1)
    y_data.append(y1)
51
52 x_data, y_data = np.array(x_data), np.array(y_data)
53
54 # Model i obucavanje
m = sn.SciModel([x, y], [L1,L2,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10], 'mse', 'Adam')
h = m.train([x_data, y_data], 12*['zero'], learning_rate=0.001, batch_size=1024,
↪ epochs=8000, adaptive_weights={'method':'NTK', 'freq':200})
57
58 # Test
x_test, y_test = np.meshgrid(
    np.linspace(0, 1, 200),
    np.linspace(0, 1, 200)
)
62
63 v_pred = v.eval(m, [x_test, y_test])
w_pred = w.eval(m, [x_test, y_test])

```

Након убичајених импорта пакета, формирајмо ФПНМ мреже за реални део v и имагинарни део w непознате функције u . И овде ћемо ићи са активационом функцијом $\sin(x)$. Интересантан део кода је дефинисање граничног услова на Γ_1 према последње две једначине у систему (5.3):

```

a,b,c = 0.39762422, -1.57715550, 1.60337246
C1 = (1 - sign(y - (a + b*x + c*x**2 + TOL))) * (1 + sign(x-0.4)) * (1 - sign(x- ↪
↪ 0.6)) * (1-v)
C2 = (1 - sign(y - (a + b*x + c*x**2 + TOL))) * (1 + sign(x-0.4)) * (1 - sign(x- ↪
↪ 0.6)) * (w-0)

```

Једначина $y = a + b \cdot x + c \cdot x^2$ представља једначину спољне линије сочива Γ_1 , у којој су коефицијенти a, b, c добијени фитовањем. Дакле, прва заграда у граничним условима значи да узимамо колокационе тачке које припадају танком појасу изнад линије Γ_1 , док друга и трећа заграда имају ненулту вредност само ако је $0, 4 < x < 0, 6$.

Гранични услови C3, C4, C5, C6, C7, C8 се односе на мешану формулатура према друге две једначине у систему (5.3) и важе на Γ_2 . На пример:

```
C5 = (1+sign(x - (1-TOL))) * (c*diff(v,x) - omega*w)
C6 = (1+sign(x - (1-TOL))) * (c*diff(w,x) + omega*v)
```

се односи на десну границу где је $x=1$ и узима колокационе тачке које се налазе у танком појасу ширине TOL са леве стране те границе.

Компоненте функције губитка C9 и C10 односе се такође на границу Γ_2 , али на линији где је $y = 0$ и $x < 0, 4$ или $x > 0, 6$:

```
C9 = (1-sign(y - (0+TOL))) * ((1 - sign(x-0.4)) + (1 + sign(x-0.6))) * (-
    ↪ c*diff(v,y) - omega*w)
C10 = (1-sign(y - (0+TOL))) * ((1 - sign(x-0.4)) + (1 + sign(x-0.6))) * (-
    ↪ c*diff(w,y) + omega*v)
```

Овим смо комплетирали свих 12 компоненти функције губитка. Пошто их има толико, није једноставно извршити њихово пондерисање, односно доделу тежина свакој компоненти. У оваквим ситуацијама помажу методе за адаптивно одређивање тежина компоненти током обуке. У нашем решењу:

```
h = m.train([x_data, y_data], 12*['zero'], learning_rate=0.001, batch_size=1024,
    ↪ epochs=8000, adaptive_weights={'method':'NTK', 'freq':200})
```

употребили смо иновативну методу *Neural Tangent Kernel* (NTK) према Wang *et al.* [WYP22]. Објашњење методе излази из оквира овог практикума, па је нећемо детаљно разрађивати. Такође, ваља напоменути да смо колокационе тачке учитали из посебног фајла kolokacione_tacke.txt, који је добијен тако што смо исписали чворове коначних елемената који се добијају из генератора мреже пакета Deal.II¹⁵.

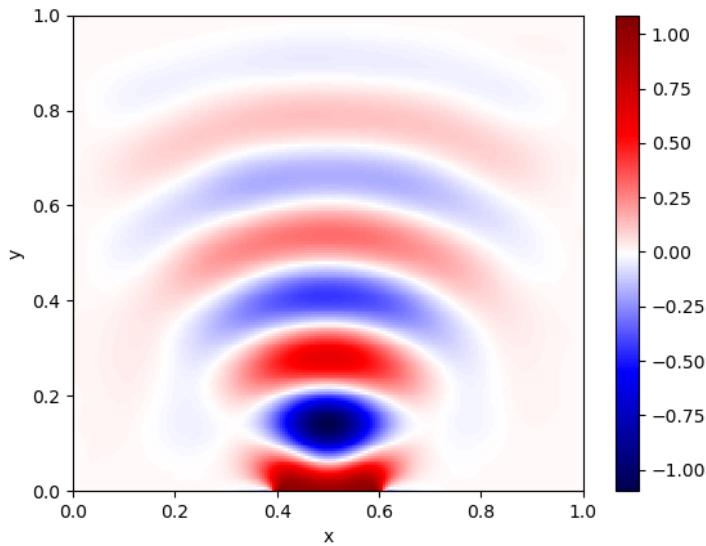
5.4.2 Резултати

Како што је на почетку одељка већ речено, пример је преузет из документације за пакет који се бави анализом методом коначних елемената Deal.II¹⁶, тако да можемо да упоредимо решење за $v = \text{Re } u$ које смо добили помоћу ФПНМ (Сл.5.7) и решење које се добија класичном методом коначних елемената (Сл.5.8). Решење добијено МКЕ методом можемо сматрати референтним, јер је коришћена веома густа мрежа и показана је конвергенција.

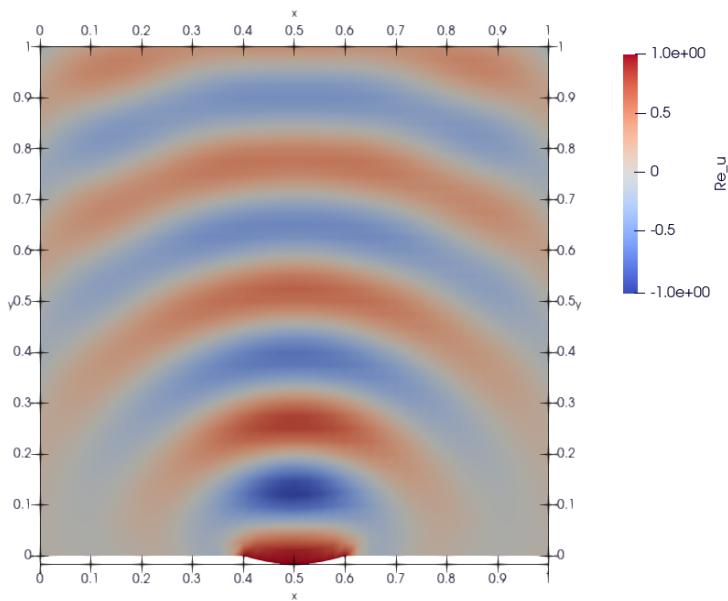
Квалитативно гледано, ФПНМ решење има исте карактеристике као МКЕ решење. Међутим, око сочива и прима границама Γ_2 очигледан је пад квалитета решења. Узрок можемо тражити на неколико места:

¹⁵ <https://www.dealii.org>

¹⁶ https://www.dealii.org/current/doxygen/deal.II/step_29.html



Слика5.7: Решење за $v = \text{Re } u$ добијено помоћу ФПНМ



Слика5.8: Решење за $v = \text{Re } u$ добијено методом коначних елемената

- Прво, код МКЕ је извор на линији сочива Γ_1 могуће прецизније специфицирати по самој линији, а не у појединачним тачкама као код ФПНМ.
- Могуће је да 50.000 колокационих тачака није довољно за обучавање.
- Примећено је да обучавање оптимизационим алгоритам Adam не може да спусти вредност губитка испод неке границе. Овде вероватно треба експериментисати са варијабилном стопом учења, или додати неки други оптимизатор као у примеру *Пропагација поплавног таласа у отвореном каналу* (страна 53).

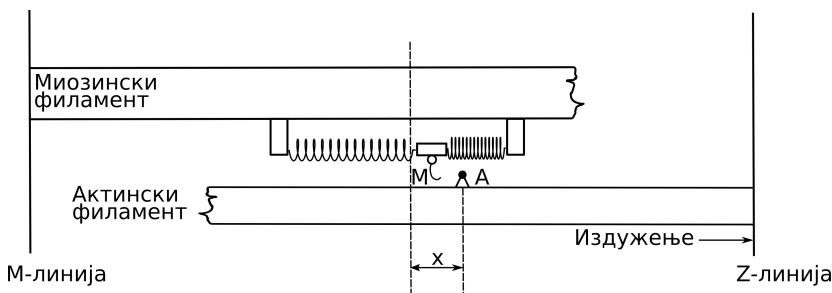
Даље експериментисање на овом примеру остављамо читаоцу.

Моделовање мишића

6.1 Увод

Хаксли и Хоцкин су 1957. године развили биофизички модел који је био основа за сва наредна истраживања у овој области Huxley [Hux57]. Природа Хакслијевог модела се ослања на динамику попречних мостова, због чега је сама теорија и назvana теорија попречних мостова Huxley [Hux57]. Детаље везане за анатомију и физиологију мишића не можемо излагати у овом типу литературе, већ се читалац упућује на Svićević [Svivecic20].

Хаксли је разматрао половину саркомере и сматрао је да су главе миозина за миозински филамент причвршћене помоћу еластичних веза. На Сл.6.1 имамо шематску репрезентацију модела, на којој се може уочити да је дебели миозински филамент фиксиран у простору за M-линију. Приликом стимулације мишића очекивано је да се миозинске главе вежу на најближе слободно место на актину, услед чега долази до формирања еластичних веза у виду попречних мостова. У том тренутку долази до генерисања активне мишићне сile.



Слика6.1: Хакслијев модела клизајућих филамената

Сила се тада преноси на актински филамент који се том приликом креће ка Z-линији. Процес формирања попречних мостова се понавља услед сталног релативног клизања актинског филамента дуж миозинског и зависи од положаја самих миозинских глава. Том приликом,

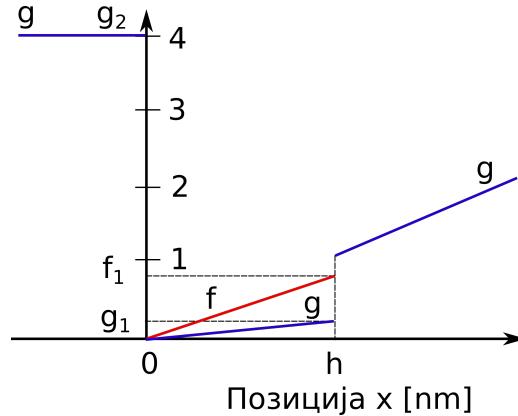
попречни мостови трпе истезања и скраћивања. Због сталног релативног клизања филамената, Хакслијева теорија се назива и Теоријом клизајућих филамената Huxley [Hux57].

У случају издужења мишића, актински филамент клизи удесно, дуж миозинског филамента, који је фиксиран за М-линију. Померање главе миозина од њене усправне позиције, у правцу М-линије или Z-линије до активног места A на актинском филаменту, означенено је са x и тумачи се као дужина попречног моста. У једном тренутку, миозинска глава може бити везана само за једно активно место актина, при чему је померање ограничено максималним померањем главе миозина, изазваним термичким флуктуацијама, h , тако да важи $0 < x < h$. Уколико дужина попречног моста постане већа од h , долази до раскидања ове везе између филамената. У сваком тренутку је могуће идентификовати да ли је нека миозинска глава закачена или не, и да ли том приликом формира попречни мост дужине x . Вероватноћа да случајно изабрана миозинска глава у тренутку t формира попречни мост дужине x из домена Ω , означена је са $n(x, t)$.

Вероватноћа $n(x, t)$ се може тумачити и као удео броја миозинских глава које су у тренутку t закачене на растојању x у односу на укупан број миозинских глава. Овај број зависи од брзине успостављања и раскидања попречних мостова, тако да се континуиран процес стварања и раскидања попречних мостова може формулисати једначином:

$$\frac{dn(x, t)}{dt} = [1 - n(x, t)] f(x) - n(x, t) g(x), \quad (6.1)$$

где су $f(x)$ и $g(x)$ редом стопе успостављања и раскидања везе између миозина и актина у јединици времена, које зависе од растојања x , као на Сл.6.2. Вероватноћа успостављања везе је представљена производом удела оних миозинских глава које још увек нису закачене, $1 - n(x, t)$, и стопе успостављања везе, $f(x)$. С друге стране, вероватноћа да се успостављена веза између актина и миозина прекине је дата као $n(x, t)g(x)$.



Слика6.2: Стопе успостављања везе, f , (наранџаста линија) и раскидања везе, g , (плава линија) између миозина и актина

Хакслијева теорија кинетике попречних мостова се може изразити коришћењем парцијалне диференцијалне једначине над доменом Ω :

$$\frac{\partial n}{\partial t}(x, t) - v \frac{\partial n}{\partial x} = \mathcal{N}(n(x, t), x), \quad \forall x \in \Omega,$$

где је $v = -dx/dt$ брзина клизања филамента актина у односу на филамент миозина (позитивна при контракцији), а

$$\mathcal{N}(n(x, t), x) = [1 - n(x, t)] f(x) - n(x, t) g(x)$$

представља брзину промене стања попречних мостова.

У циљу што реалистичнијег описа понашања мишића током издужења, Захалак је увео одређене модификације оригиналног Хакслијевог модела. Увео је минималне промене у дефиницији стопа успостављања и раскидања веза између миозина и актина, тако да је при скраћивању мишића све остало непромењено, док је при издужењу омогућио већу стопу откачињања. Ово је практично реализовано тако што је уведен Захалаков фактор, f_{Zah} , у случају када је дужина попречног моста x већа од h :

$$\begin{aligned} x < 0 & : f(x) = 0; g(x) = g_2; \\ 0 \leq x \leq h & : f(x) = f_1 x/h; g(x) = g_1 x/h; \\ x > h & : f(x) = 0; g(x) = f_{Zah} g_1 x/h \end{aligned} \quad (6.2)$$

Може се уочити да је у области $x < 0$ дефинисана висока вредност стопе откачињања, $g(x) = g_2$, како би се попречни мостови који су доспели у ову зону брзо прекинули. Постоји извесна стопа откачињања и у области $0 < x < h$, али је ниска у поређењу са негативном облашћу.

Након завршене обуке неуронске мреже, она се може користити као замена за **метод карактеристика** објашњен у Svićević [Svivcevic20]. Овде ћемо се зауставити са објашњавањем математичког модела, а читаоца који се интересује за изучавање ове области упутити на Svićević [Svivcevic20].

6.2 Изометријски случај

Ради тестирања способности ФПНМ да решавају ову врсту парцијалних диференцијалних једначина, најпре је од интереса изометријски случај, јер је он најједноставнији, будући да је **брзина клизања филамената једнака нули**. Код изометријског случаја, вишеслојни перцептрон као улаз узима позицију x доступног актинског сајта у односу на равнотежни положај миозинске главе и време t , а предвиђа вероватноће закачињања миозинских глава за актинске сајтове $n(x, t)$.

6.2.1 Имплементација

Комплетан код је дат на следећем листингу.

Листинг 6.1: Решење Хакслијеве једначине за изометријски случај

```
1 import deepxde as dde
2 import numpy as np
3 import tensorflow as tf
```

(наставак на следећој страни)

(настављено са претходне стране)

```

4      ''' fixed parameters '''
5      f1_0 = 43.3
6      h = 15.6
7      g1 = 10.0
8      g2 = 209.0
9      fzah = 4.0
10     a = 1.
11
12
13 # Verovatnoca kacenja
14 def f(x,a):
15     return (1+tf.sign(x)) * (1-tf.sign(x-h)) * (f1_0*a*x/h) * 0.25
16
17 # Verovatnoca raskacivanja
18 def g(x):
19     return 0.5 * (1-tf.sign(x)) * g2 + \
20             0.25 * (1+tf.sign(x)) * (1-tf.sign(x-h)) * (g1*x/h) + \
21             0.5 * (1+tf.sign(x-h)) * (fzah*g1*x/h)
22
23 # n = n(x,t)
24 def pde(x, n):
25     dn_dt = dde.grad.jacobian(n, x, i=0, j=1)
26     loss = dn_dt - (1.0-n) * f(x[:,0:1],a) + n*g(x[:,0:1])
27     # Obezbedi pozitivna resenja
28     return loss + n*(1-tf.sign(n))
29
30 # Computational geometry
31 geom = dde.geometry.Interval(-20.8, 63)
32 timedomain = dde.geometry.TimeDomain(0, 0.4)
33 geomtime = dde.geometry.GeometryXTime(geom, timedomain)
34
35 # Pocetni uslovi
36 ic1 = dde.icbc.IC(geomtime, lambda x: 0.0, lambda _: on_initial: on_initial)
37
38 data = dde.data.TimePDE(geomtime, pde, [ic1], num_domain=10000, num_
39 ↪boundary=100, num_initial=500)
40 net = dde.nn.FNN([2] + [40] * 3 + [1], "tanh", "Glorot normal")
41 model = dde.Model(data, net)
42
43 model.compile("adam", lr=1e-3)
44 model.train(100000)
45 model.compile("L-BFGS", loss_weights=[1.e-1, 1])
46 losshistory, train_state = model.train()
47 dde.saveplot(losshistory, train_state, issave=True, isplot=True)

```

Конструисана је мрежа са 3 слоја, са по 40 неурона и активационом функцијом \tanh . Генерирана је мрежа података за 10.000 еквидистантних вредности за x у опсегу $-20.8[nm] \leq x \leq 63 [nm]$ и са 500 насумичних вредности за t у опсегу $0[s] \leq t \leq 0.4 [s]$. Ове генериране тачке су коришћене као улазни подаци за обуку мреже. У току обуке, од 100.000 епоха, минимизује се резидуал Хакслијеве једначине за мишићну контракцију и резидуал почетног услова, коришћењем Адам оптимизације са стопом учења 10^{-3} .

Што се саме имплементације тиче, треба скренути пажњу на неколико битних момената. Као прво, да бисмо формулисали једначине (6.2) не саветује се коришћење условних израза (`if` клаузула), већ се иста функционалност интерпретира помоћу *TensorFlow* израза за знак, ако се он користи као бекенд. На пример:

```
def g(x):
    return 0.5 * (1-tf.sign(x)) * g2 + \
        0.25 * (1+tf.sign(x)) * (1-tf.sign(x-h)) * (g1*x/h) + \
        0.5 * (1+tf.sign(x-h)) * (fzah*g1*x/h)
```

У функцији парцијалне диференцијалне једначине имамо нешто другачији проблем. Наиме, морамо да обезбедимо да функција $n(x, t)$ не почне да узима вредности мање од нуле. Иако сама математичка поставка то дозвољава, подсетимо се да $n(x, t)$ означава вероватноћу закачињања, па негативна вредност нема никаквог физичког смисла. То ћemo урадити на следећи начин:

```
def pde(x, n):
    dn_dt = dde.grad.jacobian(n, x, i=0, j=1)
    loss = dn_dt - (1.0-n) * f(x[:,0:1], a) + n*g(x[:,0:1])
    # Obezbedi pozitivnu resenja
    return loss + n*(1-tf.sign(n))
```

У последњој линији се вредности функције губитка додаје члан који има ненулту вредност у случају да је n негативно. На први поглед, било би довољно уместо $n * (1-tf.sign(n))$ поставити само $1-tf.sign(n)$. Међутим, испоставља се да тако формирана функција губитка не доводи до конвергенције, јер није диференцијабилна, па оптимизациони алгоритми као што је Адам не конвергирају. Када се пак функција знака помножи са n , добијамо диференцијабилнију функцију губитка, а самим тим и већу вероватноћу конвергенције.

Као и у примеру *Пропагација поплавног таласа у отвореном каналу* (страна 53), и овде ћемо покушати да додатно смањимо вредност функције губитка методом L-BFGS:

```
model.compile("L-BFGS", loss_weights=[1.e-1, 1])
losshistory, train_state = model.train()
```

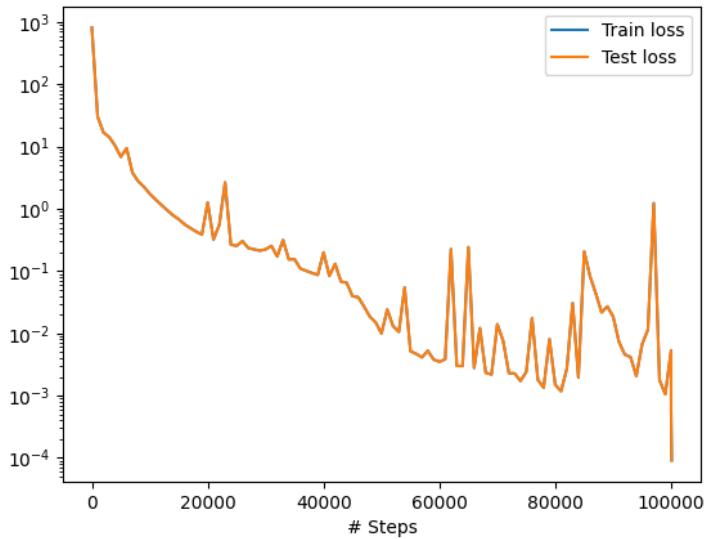
Веома је важно испоштовати постављене почетне услове, па ћемо им помоћу `loss_weights=[1.e-1, 1]` дати за ред величине већу тежину од компоненте која следи из саме парцијалне диференцијалне једначине.

6.2.2 Резултати

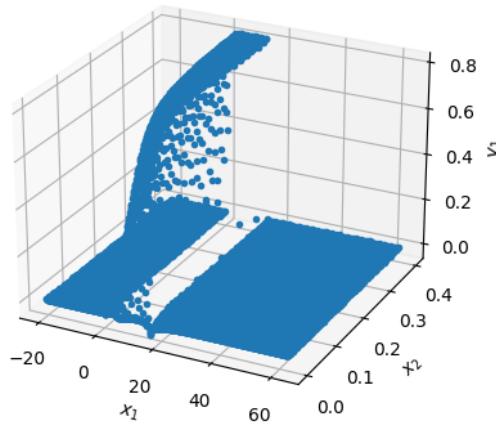
И овај пример ће услед великог броја колокационих тачака (података) најбоље перформансе имати уколико се извршава на неком графичком процесору који ову масовност уме да паралелизује. На пример, на нашем графичком процесору NVidia Tesla T4 обука траје око 170 секунди, док на осмојезгarnом процесору Intel Xeon Silver 4208 на 2.10GHz траје 2150 секунди. То је убрзање од готово 13 пута!

На Сл.6.3 се види како је текао процес тренинга ФПНМ. Уочљиво је да додатно обучавање помоћу L-BFGS дефинитивно има ефекта и да смо помоћу њега спустили вредност губитка за додатни ред величине.

Тродимензионални приказ резултата дат је на Сл.6.4. Из оваквог приказа није погодно утврђивати било какву тачност решења, али је довољан за квалитативни увид. Уочљиво је да близу $t = 0$ (x_2 на слици) има неколико колокационих тачака где је $n(x, t) < 0$, али свеукупно решење делује логично.

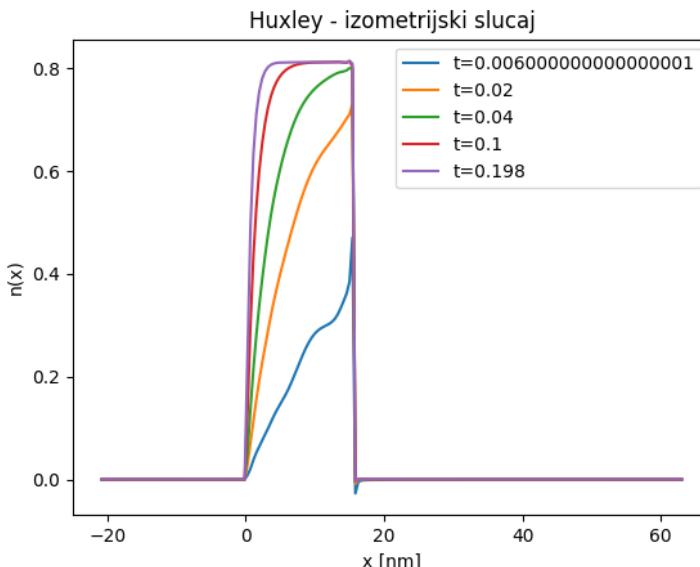


Слика6.3: Функција губитка током обучавања



Слика6.4: Тродимензионални приказ добијених резултата. x_1 је просторна координата, а x_2 временска.

Прецизнију визуелну анализу резултата можемо обавити тек графичким представљањем вероватноће закачињања мостова $n(x, t)$ дуж x осе у неколико различитих временских тренутака, као што је дато на Сл.6.5. Као што смо очекивали, уочљива је мала нестабилност око тачке $x = h$ у неколико првих временских корака, али се она губи како процес закачињања напредује.



Слика6.5: Величина $n(x, t)$ у неколико различитих временских тренутака

Упоређивање резултата са онима који су добијени методом карактеристика излази из оквира овог практикума, па ћemo тај део прескочити. На крају је важно напоменути да једну потенцијалну примену ФПНМ на коју до сада нисмо обраћали пажњу, а може бити од велике користи. Наиме, док код класичних нумеричких метода за решавање парцијалних диференцијалних једначина, решење у временском кораку $t + 1$ зависи од решења које смо имали у временском кораку t . Код свих примера које смо обрадили помоћу ФПНМ то није случај, јер се време узима као било која друга променљива по којој се врши парцијална диференцијација. Ако погледамо Сл.6.5 очигледно је да у тренутку $t = 0, 006$ имамо грешку. Међутим, **та грешка се не пропагира на касније временске тренутке** баш из наведеног разлога.

Овај другачији третман времена као улазне променљиве има импликације и на перформансе. Наиме, када се ФПНМ модел користи у продукцији, и потребно нам је да знамо нпр. $n(x, t = t_1)$ није потребно да прођемо кроз све временске кораке $t \leq t_1$, већ одмах можемо да избацимо резултат, једним проласком кроз обучену ФПНМ. Тиме се време значајно штеди и за неколико редова величине. Тиме се отварају неке нове примене, нарочито у области моделовања на више скала, као на пример у Svićević [Svivcevic20]. Таквим скоком у перформансама било би могуће овакве сложене моделе изводити скоро у реалном времену.

Моделовање производње соларних електрана

7.1 Увод

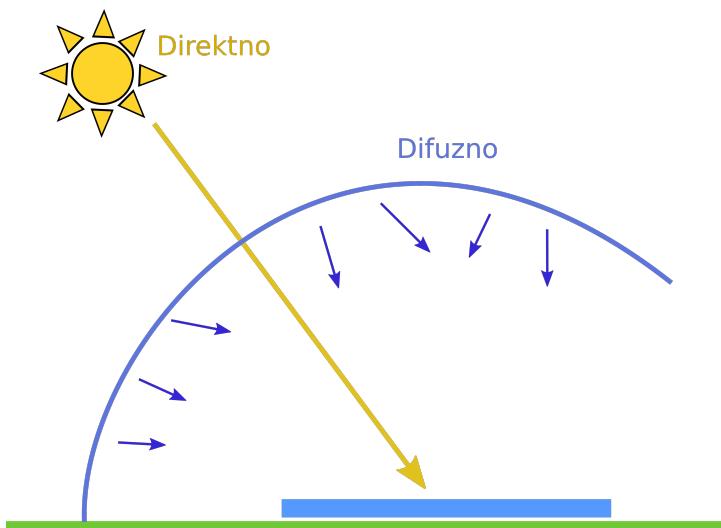
У овом поглављу бавићемо се једним врло практичним проблемом, који је аутору и сарадницима послужио као један од основних мотива за рад са ФПНМ. Наиме, ради се о „вечитом јазу” између резултата које даје модел и стварних мерења. Како овај јаз смањити, тј. како помирити модел и стварне податке? Примера ради, соларне електране су прилично добро покривене различитим физичким моделима који дају одличне резултате, али **само ако су сви улазни параметри и параметри самог система познати**, што углавном није случај. Прво, подаци о времену као што су температура, компоненте сунчевог зрачења и брзина ветра нису сасвим прецизни. Ту је и фактор старења самих панела који умањује њихову ефикасност, могућа прекривеност снегом и прашином и разни други фактори који реално утичу на производњу.

Са друге стране, било који соларни систем новије производње нуди инстант мерење излазне снаге наизменичне струје на излазу из инвертера. Покушаћемо да одговоримо на питање да ли је могуће ове мерење податке „вратити” у модел, тј. њиховим коришћењем побољшати предвиђање модела у будућности. Додатно треба нагласити да модел соларног система који ћемо демонстрирати Dobos [Dob14] и није заснован на обичној или парцијалној диференцијалној једначини, већ на једноставном аналитичком изразу. Строго гледано, могуће је искористити и обичну дубоку неуронску мрежу да опише понашање такве једначине и мањуелно формирати и функцију губитка која ће испоштовати и једначину и мерење податке. Међутим, ФПНМ библиотека као што је DeepXDE пружа добар софтверски оквир који у великој мери олакшава рад са оваквим проблемима.

У наставку ћемо објаснити основе на којима функционишу соларне електране, укључујући значајне улазне варијабле, начин функционисања и карактеристике самих уређаја.

7.1.1 Компоненте зрачења

Количина сунчеве енергије прикупљена од стране соларног панела између осталог зависи од **његове оријентације**. На пример, соларни панел окренут ка западу ће прикупљати малу количину сунчеве енергије током јутра јер је оријентисан ка тамнијем делу неба. Са помеђујем сунца, соларни панел се све више обасјава, па сакупља веће количине зрачења. Ово је кључ *Plane of Array* (POA) концепта који подразумева количину сунчеве светлости која се може прикупити за задати положај панела. Локација, нагнутост и оријентација панела су од кључног значаја за процену количине енергије која на панел пада. Осунчаност се изражава у ватима по квадратном метру $\frac{W}{m^2}$.



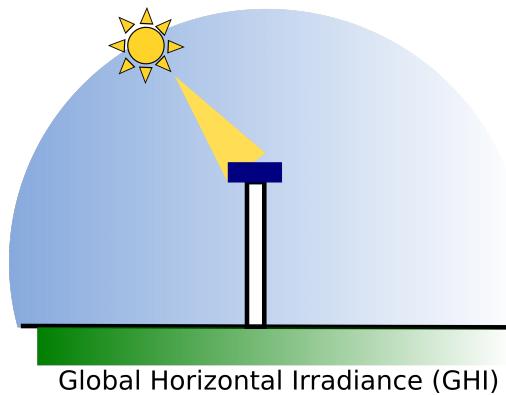
Слика 7.1: Илустрација директног и дифузног сунчевог зрачења

Када се моделује озраченсот равни панела, из практичних разлога, посматрају се следеће три компоненте:

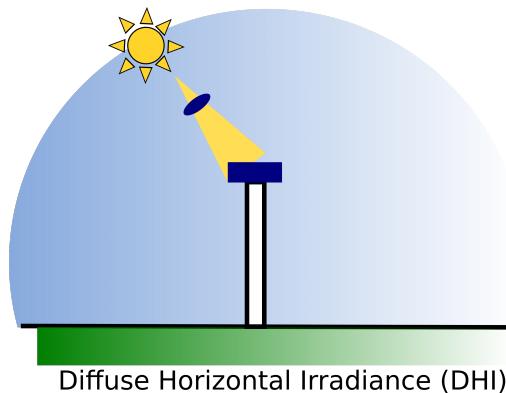
- GHI – **Глобална хоризонтална озраченост**, укупна јачина сунчеве светлости која пада на хоризонталну раван, [Сл.7.2.](#)
- DHI – **Дифузна хоризонтална озраченост**, део сунчеве светлости која пада на хоризонталну раван, али која не долази директно од сунца, [Сл.7.3.](#)
- DNI – **Директна нормална озраченост**, део сунчеве светлости која долази директно од сунца, [Сл.7.4.](#)

Свака од наведених компоненти се одговарајућим мерним поступком и инструментом посебно мери. На пример, израчунавање компоненте директног зрачења DNI која пада на панел решава се једноставно на основу упадног угла. Проналажење компоненте дифузне озрачености DHI је сложеније и може варирати у зависности од атмосферских услова. Употреби су различити приступи за конверзију DHI у дифузну компоненту. Трећа компонента зрачења GHI је светлост која се одбија од тла пре него што је сакупи фотонска плоча. На Интернету се могу пронаћи бесплатне базе историјских података. На појединим

плаћеним сервисима могу се наћи и прогнозе све три компоненте. Серије на овим сервисима су углавном сатне учестаности.



Слика7.2: GHI- Глобална хоризонтална озраченост

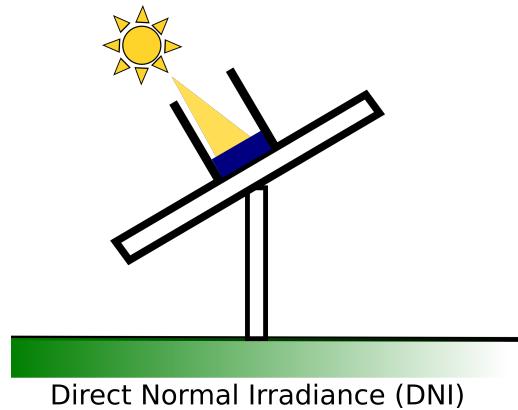


Слика7.3: DHI – Дифузна хоризонтална озраченост

7.1.2 Оријентација и локација панела

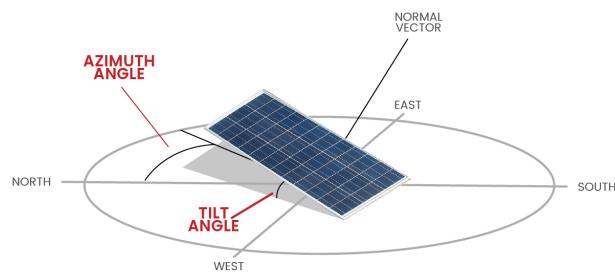
Два су кључна угла који дефинишу оријентацију соларног панела. Један одређује правац постављања панела (север, исток, југ, запад), а други одређује нагнутост соларног панела у односу на хоризонталну раван. *Azimuth* одређује правац постављања панела, при чему договорно важи да је север 0, исток 90, југ 180, а запад 270 степени. *Tilt* који одређује нагнутост соларног панела има вредност 0 ако је панел постављен хоризонтално, а вредност 90 уколико је постављен потпуно вертикално. У зависности од начина постављања система, и један и други угао могу бити фиксне вредности или временске серије. На таквим системима се обично *tilt* мења, пративши кретање сунца на небу и обезбеђујући већу озраченост нормалном компонентом.

Одређивање тачне **локације панела** је веома важно приликом рачунања угла који Сунце



Direct Normal Irradiance (DNI)

Слика7.4: DNI – Директна нормална озраченост



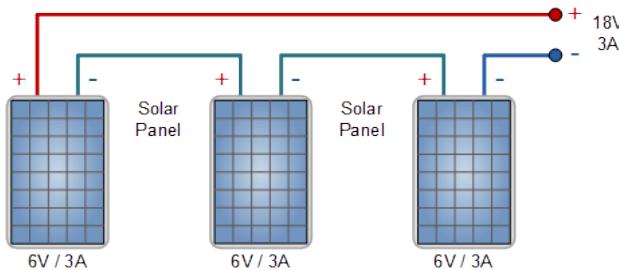
Слика7.5: Оријентација соларног панела

заклапа са панелом у различито доба године. Параметри за рачунање угла који сунце заклапа са панелом су:

- географска ширина,
- географска дужина,
- надморска висина и
- временска зона.

7.1.3 Начини повезивања панела

Соларни панели се у електрично коло могу редно (серијски) и паралелно. Редна веза сумира напон у електричном колу, док паралелна веза повећава јачину струје. Могуће је направити и комбинацију редне и паралелне везе.



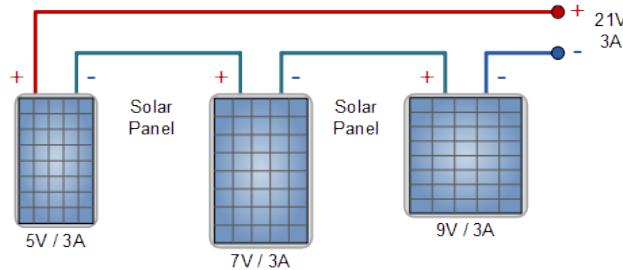
Слика 7.6: Серијска веза соларних панела истих карактеристика. Сви соларни панели су истог типа и имају исту излазну снагу. Укупан напон електричног кола је збир напона на сваком панелу. У овом примеру имамо 3 панела који производе напон од 6V и струју јачине 3A, односно електрично коло има напон од 18V и струју јачине 3A. Укупна снага везе је $18V \cdot 3A = 54W$ при максималној осунчаности¹⁷.

7.1.4 Номинална снага панела (Peak Power - kW_p)

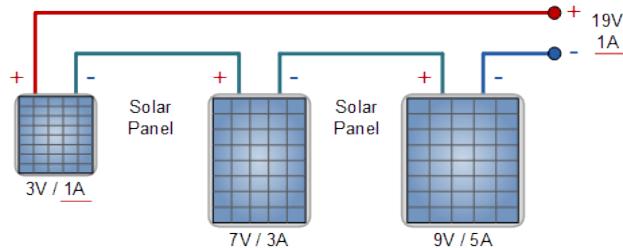
Мерења снаге соларних панела у лабораторији или фабрици врши се под стандардизованим условима. Ти услови дефинисани су интеграционим стандардном IEC-60904-1 и то су:

- Интензитет осунчаности износи $1000W/m^2$ на целој површини соларног панела. У реалним условима ова вредност је неретко већа.
- Температура панела износи $25^\circ C$. Спектар светlostи мора бити исти као глобални спектар светlostи дефинисан у IEC 60904-3. Одговара спектру светlostи по сунчаном дану са положајем сунца око 40° изнад хоризонта и панелом који је окренут према сунцу, а са хоризонтом заклапа угао од 40° .

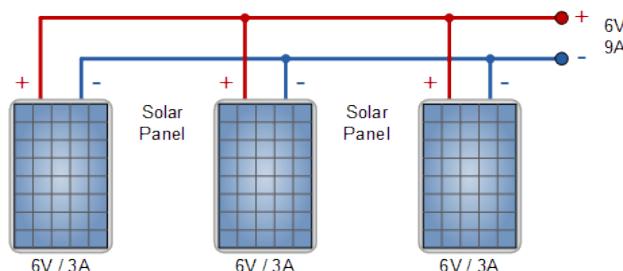
¹⁷ <https://www.alternative-energy-tutorials.com/solar-power/connecting-solar-panels-together.html>



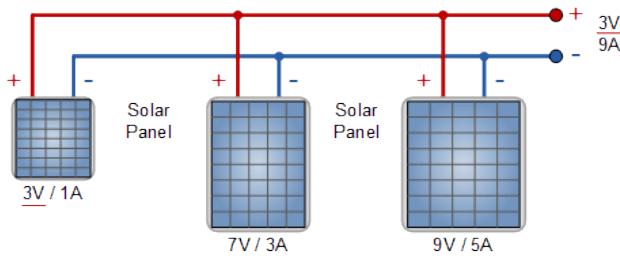
Слика7.7: Серијска веза соларних панела различитих волтажа. У овом примеру сви соларни панели су различитих типова, имају различиту снагу, али им је заједничка максимална јачина струје. Када су везани серијски, заједно производе електрични напон од 21V и струју јачине 3A, односно снага је 63W. Јачина струје је иста као и у претходном примеру, али је промењен напон ($5V+7V+9V$).



Слика7.8: Серијска веза панела различитих напона и јачина електричне струје. У овој методи соларни панели су различитих типова, сваки панел има различит напон, јачину струје и снагу. Укупни напон електричног кола поново се рачуна као збир напона на сваком соларном панелу ($3V+7V+9V$), док је јачина струја у колу ограничена панелом с најмањом јачином струје - 1A. Тиме је снага лимитирана на само 19W од могућих 69W. Употреба соларних панела различитих струја није ефикасна у редној вези.



Слика7.9: Паралелна веза соларних панела истих карактеристика. Сви соларни панели на слици имају исте карактеристике, напон, јачину електричне струје и снагу. Напон на сваком панелу је 6V па је и укупан напон кола 6V. Јачина струје на излазу представља збир свих јачина електричне струје на панелима $3A+3A+3A=9A$. Остварена снага при потпуној осунчаности панела износи 54W.



Слика 7.10: **Паралелна веза соларних панела различитих напона и јачина струје.** Да би радили у паралелиној вези, сви соларни панели морају имати исти напон, односно напон на свим панелима биће једнак најмањем напону на једном од панела. Дакле, укупан напон кола износи 3V, док је укупна јачина електричне струје одређена збиром струја $1A + 3A + 5A = 9A$. Снага износи само 27W. Због ових губитака не препоручује се паралелна веза соларних панела различитих напона.

Измерена снага при овим условима назива се номинална снага или снага у пику – *Peak Power*. Номинална снага изражава се у киловат-пику kW_p . Ако није позната укупна декларисана номинална снага соларних панела, а познати су површина соларних панела m^2 и декларисана ефикасност у %, може се израчунати по формулама:

$$NominalnaSnaga = 1 \frac{kW}{m^2} \cdot Povrsina \cdot \frac{Efikasnost}{100}$$

У већини случајева номинална снага је позната и дата је у спецификацији производа од стране производа. Номинална снага још се назива и максимална снага и означава са P_{max} .

7.1.5 Инвертер

Соларним панелима производи се једносмерна струја (DC). Да би се употребила произведена електрична енергија, потребно је извршити DC/AC конверзију. У употреби су различити типови инвертера:

- *Grid-tie inverter* – Прикључени су на дистрибутивну електричну мрежу, односно произведена електрична енергија прослеђује се дистрибутивној електричној мрежи. За њихов рад није потребна батерија.
- *Off-grid inverter* – Познати су и као независни инвертери. Конвертују једносмерну струју из батерија у наизменичну. Углавном се користе за више домаћинства, или стамбену зграду.
- *Hybrid inverter* – Конвертују DC у AC и могу се користити и као off-grid и као grid-tie системи.
- *String inverter* – Најчешће се користе у домаћинствима. Називају се „стринг“ јер се на њих прикључује низ соларних панела. Може се прикључити и више низова одједном.

Сваки инвертер има дефинисану максималну снагу. То је важно из два разлога: (1) електронске компоненте инвертера дизајниране су за рад са одређеним опсегом напона и (2) и

сами соларни панели су дизајнирани за рад до одређене снаге.

Инвертери, наравно, не могу пружити већу излазну снагу од прописане. Када се на улаз инвертера доводи једносмерна струја снаге веће од прописане улазне снаге појављује се одсецање инвертера, тј. инвертер има исту излазну снагу и поред повећања снаге на улазу. Инвертер може мењати напон на улазу како би смањио снагу на улазу. Повећава оперативни напон соларних панела преко њиховог дефинисаног максимума, чиме се смањује јачина произведене струје, односно умањује снага на улазу инвертера.

Размотримо ситуацију где су упарени соларни панели снаге 6 kW са DC/AC инвертером снаге 5 kW. На први поглед изгледа да ће дosta снаге бити изгубљено, али велики број производића препоручују управо овакав однос. Овде уводимо појам односа снага DC и AC познат и као *Inverter Load Ratio - ILR*. У овом примеру тај однос износи 1,2 (6kW/5kW). Пројектанти оваквих система су генерално конзервативни по питању DC/AC односа. Већина сматра да је однос 1,1 идеалан, а 1,2 прејак. Ипак, однос 1,2 доводи до најмањих губитака, док 1,25 или 1,3 могу остварити одређене економске бенефите при куповини, јер се користе јефтинији инвертери¹⁸.

7.1.6 I-V карактеристика соларне ћелије

Крива I-V карактеристике приказује однос струје и напона соларне ћелије, соларног панела или низа панела. Детаљно описује ефикасност конверзије соларне енергије у електричну. Познавање електричне I-V карактеристике и номиналне снаге P_{max} панела је кључно за одређивање ефикасности.

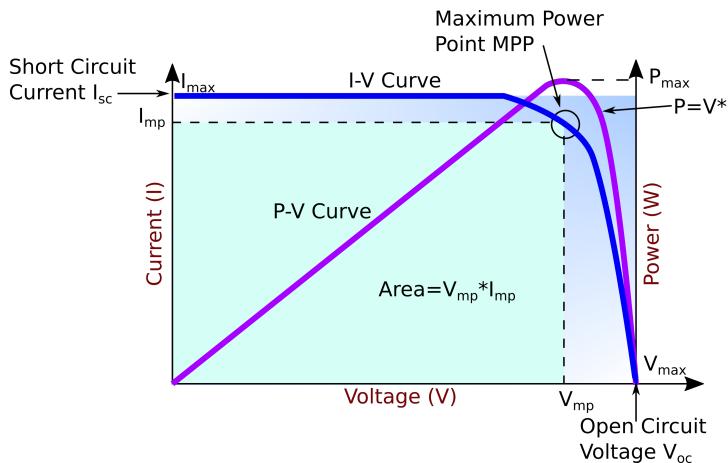
Интензитет зрачења којим се обасјава соларна ћелија одређује интензитет струје, док **повећање температуре соларне ћелије смањује напон**. Крива I-V карактеристике је графичка репрезентација операција у соларној ћелији или панелу сумирајући однос између струје и напона, односно осунчаности и температуре. Крива пружа потребне информације за конфигурацију соларног панела, како би се конфигурисао за рад близу своје оптималне снаге (*Peak Power*).

На Сл.7.11 види се I-V карактеристика (плава линија) типичне силиконске соларне ћелије при нормалним условима. Снага испоручена од стране једне соларне ћелије или панела је производ излазне струје и напона. Крива снаге у зависности од напона означена је љубичастом бојом. Соларна ћелија је **пасивни уређај у електричном колу**. I-V крива приказује све могућности за рад соларне ћелије, али стварни однос струје и напона зависиће од додатног оптерећења у електричном колу.

На пример, када је на ћелију прикључена батерија, напон је 12V, а струја је висока. С друге стране, када је прикључен потрошач, мења се однос струје и напона. Размотримо два крајња случаја:

- **Отворено коло** није повезано на оптерећење. Тада је струја на нули, а напон има максималну вредност. Такав напон назива се напоном отвореног кола, *Open Circuit Voltage*, односно V_{OC} .

¹⁸ <https://help.helioscope.com/article/248-understanding-dc-ac-ratio>



Слика 7.11: Карактеристична I-V крива

- **Коло кратког споја**, када су позитивни и негативни крај соларне ћелије у кратком споју. Напон на соларној ћелији је једнак нули, док је струја максимална. Таква струја назива се струјом кратког споја - *Short Circuit Current, I_{SC}* .

За нас је најзанимљивији случај у којем комбинација струје и напона даје највећу вредност снаге. Означимо те вредности са I_{MP} и V_{MP} . То је тачка у којој соларна ћелија генерише максимум снаге и означена је у горњем десном углу зеленог правоугаоника на слици означеном *MPP (Maximum Power Point)*. Дакле, идеална производња соларне ћелије дефинисана је тачком *MPP* која се налази на превоју I-V карактеристичне криве. Одговарајуће вредности за I_{MP} и V_{MP} могу се проценити на основу напона отвореног кола $V_{MP} \approx (0,85 - 0,9) \cdot V_{OC}$ и струје кратког споја $V_{MP} \approx (0,85 - 0,95) \cdot I_{SC}$.

С обзиром да на напон соларног панела **значајно утиче његова температура**, стварне вредности излазне снаге могу варирати. До сада смо разматрали I-V карактеристичну криву једне соларне ћелије или једног панела. Када имамо више уvezаних соларних панела крива I-V карактеристике има исти облик, само су вредности скалиране. Као што смо већ поменули, панели могу бити уvezани серијски или паралелно, односно могу произвести већи напон или већу струју. У сваком случају, горњи десни угао осенченог правоугаоника означаваје *MPP*. Постоје још два значајна параметара који описују рад панела:

- **FF (Fill Factor)** - Преставља однос максималне снаге коју низ соларних панела може да произведе под нормалним условима и производи струје кратког споја и напона отвореног кола, $FF = P_{max} / (I_{SC} \cdot U_{OP})$. Што је вредност ближа јединици, може се произвести више снаге. Уобичајене вредности су између 0,7 и 0,8.
- **%Eff (Percent Efficiency)** - Ефикасност низа соларних панела је однос између максималне електричне снаге коју могу произвести и осунчаности. Данас је ефикасност панела углавном око 10% до 12%, у зависности од типа технологије.

У наредном одељку *Основне једначине модела* (страница 96) изложићемо основне једначине које ћемо користити за формирање модела базираног на ФПНМ.

7.2 Основне једначине модела

Како што је већ најављено у претходном одељку *Увод* (страна 87), за моделовање ФПНМ методом користићемо једноставан *PVWatts* модел Dobos [Dob14]. Вертикална моделовања састоји се из три корака:

- прорачун температуре соларног панела,
- прорачун излазне снаге једносмерне струје (*PVWatts* у ужем смислу) и
- прорачун излазне наизменичне струје на излазу из инвертера.

7.2.1 Прорачун температуре соларног панела

Загревање соларног панела утиче на промену напона, па је прорачун температуре панела значајна ставка у моделовању перформанси. У широкoj употреби су два модела, и то SAPM (*Sandia Array Performance Model*) и PVSyst. Ми ћемо користити овај први. SAPM методологија моделирања температуре ћелије T_{cell} дата је следећим паром једначина:

$$\begin{aligned} T_m &= E \cdot e^{a+bW_s} + T_a \\ T_{cell} &= T_m + \frac{E}{E_0} \Delta T, \end{aligned} \tag{7.1}$$

где E представља интензитет *POA* зрачења, W_c представља брзину ветра на висини од 10 метара, T_a температуру ваздуха, док је E_0 референтна вредности интензитета *POA* зрачења, за коју се обично узима вредност од $1000 W/m^2$.

Вредности параметара $a, b, \Delta T$ су специфичне за различите моделе соларних панела у зависности од начина њихове израде (*glass/glass* или *glass/polymer*) и начина на који се постављају (*open rack, close roof, open rack, insulated back*).

7.2.2 Излазна снага једносмерне струје

Модел *PVWatts* први пут је објављен од стране *National Renewable Energy Laboratory (NREL)*. Једначине на којима је базиран су релативно једноставне. Улазне варијабле су:

- Ефективна осунчаност G_{poaef} - Укупна озраченост у јединицама W/m^2 . Подразумева се да смо већ узели у обзир губитке на основу угла под којим је панел постављен, али не и друге губитке, нпр. због прашине, спектралне губитке, итд.
- Температура панела T_c која се добија из једначине (7.1).

Израз гласи:

$$P_{dc} = \frac{G_{poaef}}{1000} P_{dc0} (1 + \gamma_{pdc}(T_{cell} - T_{ref})), \tag{7.2}$$

где су P_{dc0} и T_{ref} номинална снага панела и референтна температура (подразумевано $25^\circ C$), респективно. γ_{pdc} представља температурни коефицијент у јединицама $1/C$ са типичним вредностима у интервалу од -0,002 до -0,005 по степену Целзијуса. Што је овај коефицијент виши по апсолутној вредности, то производња више опада загревањем панела.

7.2.3 PVWatts модел инвертера

PVWatts модел инвертера Dobos [Dob14] рачуна његову ефикасност η као функцију улазне снаге једносмерне струје:

$$\eta = \frac{\eta_{nom}}{\eta_{ref}} \left(-0,0162\zeta - \frac{0,0059}{\zeta} + 0,9858 \right), \quad (7.3)$$

где је $\zeta = P_{dc}/P_{dc0}$ и $P_{dc0} = P_{ac0}/\eta_{nom}$.

Тада је снага излазне наизменичне струје:

$$P_{ac} = \min(\eta P_{dc}, P_{ac0}).$$

Величина P_{dc} представља улазну снагу једносмерне струје и дата је у истим јединицама као P_{dc0} . P_{dc0} је горња граница улазне снаге. Коефицијенти η_{nom} и η_{ref} имају подразумеване вредности од 0,96 и 0,9637 респективно.

7.3 Пример прорачуна производње

7.3.1 Аналитички модел

У овом одељку ћемо употребити методологију приказану у [Увод](#) (страна 87) и једноставни модел PVWatts објашњен у [Основне једначине модела](#) (страна 96) употребити на једном реалном примеру. У питању је соларна електрана у Централној Србији са 146 инсталисаних панела на крову једне зграде. Све карактеристике инсталације су нам познате (угао нагиба, азимут, врста панела, карактеристике инвертора и слично). Од података имамо и серије температуре ваздуха и компоненти осунчаности, тако да лако можемо аналитички, по једначинама (7.1), (7.2) и (7.3) да израчунамо производњу. У случају да поседујемо временску прогнозу, можемо приближно и да предвидимо будућу производњу користећи стандардни PVWatts модел.

Иако математика није компликована, за програмску имплементацију овог аналитичког модела користићемо помоћ библиотеке PVLIB Holmgren *et al.* [[HHM18](#)]. На наредном листингу приказан је део кода који се тиче учитавања, припреме података и извођења PVWatts модела.

Листинг 7.1: Имплементација PVWatts (аналитичког) модела производње

```

1 import numpy as np
2 import pandas as pd
3 import pvlib
4 from pvlib.location import Location
5
6 # Ucitaj podatke o vremenu i proizvodnji
7 solcast_data = pd.read_csv("data_21.08.16_22.10.14.csv", index_col="Time")
8 solcast_data.index = pd.to_datetime(solcast_data.index, dayfirst=True)

```

(наставак на следећој страни)

(настављено са претходне стране)

```

9
10 # Nagib, azimut i lokacija panela
11 surface_tilt = 7
12 surface_azimuth = 290
13 location = Location(latitude=43.905410, longitude=20.341986, altitude=243, tz=
14     ↪"Europe/Belgrade", name="Pons Cacak")
15
16 # Pomeri vreme za pola sata, izracunaj poziciju sunca u svakom trenutku
17 times = solcast_data.index - pd.Timedelta('30min')
18 solar_position = location.get_solarposition(times)
19 solar_position.index += pd.Timedelta('30min')
20
21 # Novi dataframe sa vrednostima za vrednosti osuncanosti DNI, GHI, DHI
22 df_poa = pvlib.irradiance.get_total_irradiance(
23     surface_tilt=surface_tilt,
24     surface_azimuth=surface_azimuth,
25     dni=solcast_data['DNI'],
26     ghi=solcast_data['GHI'],
27     dhi=solcast_data['DHI'],
28     solar zenith=solar_position['apparent_zenith'],
29     solar azimuth=solar_position['azimuth'],
30     model='isotropic')
31
32 # Izvuci ukupnu POA vrednost iz df_poa
33 E_data = df_poa["poa_global"]
34 # Izvuci temperaturu vazduha
35 T_data = solcast_data["Tamb"]
36 # Izvuci proizvodnju P
37 P_data = solcast_data["P"]
38
39 # Nedelju dana za treniranje
40 E_data_train = E_data.loc["2021-10-31":"2021-11-06"]
41 T_data_train = T_data.loc["2021-10-31":"2021-11-06"]
42 P_data_train = P_data.loc["2021-10-31":"2021-11-06"]
43
44 # Sledecih nedelju dana za testiranje
45 E_data_test = E_data.loc["2021-11-07":"2021-11-13"]
46 T_data_test = T_data.loc["2021-11-07":"2021-11-13"]
47 P_data_test = P_data.loc["2021-11-07":"2021-11-13"]
48
49 #
50 # Parametri modela
51 pdc0 = 0.375 # nominal power [kWh]
52 Tref = 25.0 # cell reference temperature
53 gamma_pdc = -0.005 # influence of the cell temperature on PV system
54 pdc0_inv = 50
55 eta_inv_nom = 0.96
56 eta_inv_ref = 0.9637
57 pac0_inv = eta_inv_nom * pdc0_inv # maximum inverter capacity
58 a = -2.98 # cell temperature parameter
59 b = -0.0471 # Wind coefficient
60 E0 = 1000 # reference irradiance
61 deltaT = 1 # cell temperature parameter
62 num_of_panels = 146 # Broj panela u instalaciji
63
64 #

```

(наставак на следећој страни)

(настављено са претходне стране)

```

65 # Originalni PVWatts model
66 #
67 def orig_pvwatts_model(x):
68     Ta = x[:,0:1] # Temperatura vazduha
69     E = x[:,1:2] # Ukupna POA osuncanost
70
71     Tm = E * np.exp(a+b*T) + Ta # Brzina vетra uzeta kao prosecna od 2 m/s
72     Tc = Tm + E/E0*deltaT
73     P_dc_temp = ((Tc-Tref) * gamma_pdc + 1.)
74     P_dc = (E * 1.e-03 * pdc0 * P_dc_temp) * num_of_panels
75     zeta = (P_dc+1.e-2)/pdc0_inv
76
77     eta = eta_inv_nom/eta_inv_ref * (-0.0162*zeta - 0.0059/zeta + 0.9858)
78     eta[eta<0] = 0.
79     ac = np.minimum(eta*P_dc, pac0_inv)
80
81     return ac

```

Након стандардних импорта библиотека, учитавамо сатне серије података о времену:

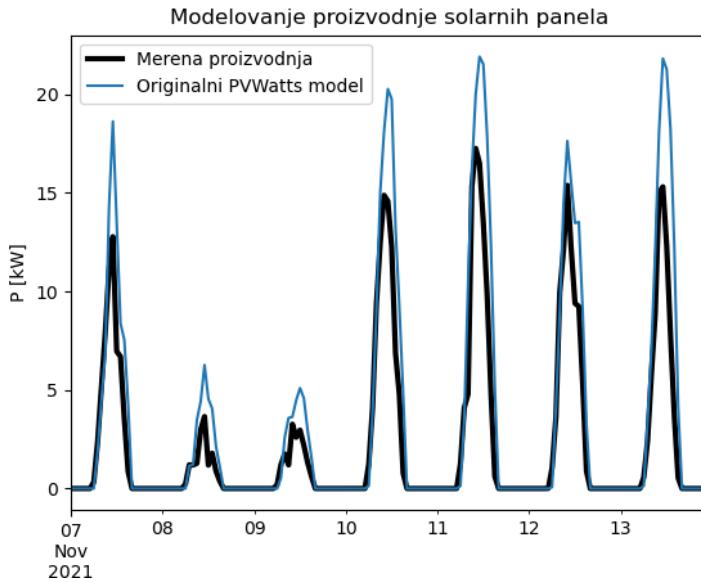
- T_a - температура ваздуха,
- DNI, GHI, DHI - компоненте осунчаности

и податке о производњи P у *Pandas* оквир. На жалост, немамо доступне податке о брзини ветра на локацији, па ћемо ту компоненту узети као просечну на том поднебљу, тј. вредност од 2 m/s . Функција `pvlib.irradiance.get_total_irradiance()` библиотеке *PVLIB* нам израчунава положај сунца за било који временски тренутак и било коју локацију на Земљи, а на основу њега и укупну осунчаност панела чији је положај дат угловима `surface_tilt` и `surface_azimuth`. Као улаз, ова метода узима компоненте DNI, GHI и DHI дефинисане у одељку *Компоненте зрачења* (страна 88).

Након екстракције периода од недељу дана од 31. октобра до 6. новембра и другог периода од 7. до 13. новембра, постављамо све параметре који се користе у једначинама (7.1), (7.2) и (7.3), онако како најбоље одговара самој инсталацији. Сви параметри осим три су константе прихваћене у литератури. Та три параметра чија вредност може да се подешава су a и b из израза (7.1) и γ_{pdc} из израза (7.2). Њихове вредности (-2,98, -0,0471 и -0,005 респективно) задајемо према врсти панела и начину постављања инсталације.

Поређење резултата добијених чистим моделом и мерених вредности може се видети на Сл.7.12. Очигледно је да постоји значајно одступање, тј. модел даје више вредности од мерења. Корен средње квадратне грешке (*RMSE*) износи чак $2,46 \text{ kW}$. Одређени допринос овако високој вредности грешке сигурно је последица чињенице да хлађење панела услед утицаја ветра нисмо узели у обзир услед недостатка података о ветру.

Ниједан модел није савршена слика стварности и не може да обухвати све факторе који утичу на производњу. Чак иако је модел савршен (а не може бити), могуће су појаве грешака при мерењу температуре, осунчаности, брзине ветра. Даље, код самог модела имамо више параметара чије вредности узимамо из литературе и декларације произвођача. Реалне вредности тих параметара сигурно одступају од тих вредности и мењају се током животног века уређаја.



Слика 7.12: Поређење чистог PVWatts модела са мереном производњом

7.3.2 Подешавање параметра помоћу ФПНМ

Поставља се питање да ли вредности које даје модел могу бити приближније реалним вредностима само подешавањем параметара модела. Пробаћемо да искористимо чињеницу да податке о производњи за недељу од 31. октобра имамо и да помоћу ФПНМ пробамо да „помиримо” излаз аналитичког модела и мерења производње тако што ћемо параметар a једначине (7.2) прогласити непознатим. Основна идеја је да се ФПНМ тренира и једначином и подацима и да као излаз испоручи и модел са мањом грешком и нову, бољу вредност параметра a .

На следећем листингу могу се видети интересантни делови имплементације ове идеје:

Листинг 7.2: Инверзни проблем подешавања параметара PVWatts/SAPM модела

```

1 # Parametar "a" pustamo da se trenira
2 a_var = dde.Variable(-4.0)
3
4 #
5 # Jednacina koju koristi PINN
6 #
7 def pvwatts_eq(x, y):
8     Ta = x[:, 0:1] # Temperatura vazduha
9     E = x[:, 1:2] # Ukupna POA osuncanost
10
11     Tm = E * tf.exp(a_var+b*x**2) + Ta # Brzina vetra uzeta kao prosecna od 2 m/s
12     Tc = Tm + E/E0*deltaT

```

(наставак на следећој страни)

(настављено са претходне стране)

```

13 P_dc_temp = ((Tc-Tref) * gamma_pdc + 1)
14 P_dc = (E * 1.e-03 * pdc0 * P_dc_temp) * num_of_panels
15 zeta = (P_dc+1.e-2)/pdc0_inv
16
17 eta = eta_inv_nom/eta_inv_ref * (-0.0162*zeta - 0.0059/zeta + 0.9858)
18 eta = tf.maximum(0., tf.sign(eta)) * eta
19 ac = tf.minimum(eta*p_dc, pac0_inv)
20
21 return y - ac
22
23 # Imamo 168 tacaka sa merenjima prozivodnje. Pripremi strukturu za PointSet_ ↵ granicni uslov
24 train_points = np.zeros((168,2))
25 train_measured_production = np.zeros((168,1))
26 train_points[:,0] = T_data_train.to_numpy().T
27 train_points[:,1] = E_data_train.to_numpy().T
28 train_measured_production[:,0] = P_data_train.to_numpy().T
29
30 # Imamo 168 tacaka sa merenjima za narednu nedelju za test
31 test_points = np.zeros((168,2))
32 test_measured_production = np.zeros((168,1))
33 test_points[:,0] = T_data_test.to_numpy().T
34 test_points[:,1] = E_data_test.to_numpy().T
35 test_measured_production[:,0] = P_data_test.to_numpy().T
36
37 # Minimumi i maksimumi T i E za kreiranje geometrije problema
38 minT, maxT = min(train_points[:,0]), max(train_points[:,0])
39 minE, maxE = min(train_points[:,1]), max(train_points[:,1])
40
41 geom = dde.geometry.Rectangle([minT, minE], [maxT, maxE])
42 bc_y = dde.icbc.PointSetBC(train_points, train_measured_production, component=0)
43
44 # Isti broj kolokacionih tacaka za jednacinu i za granicne uslove. Moze i ↵ drugacije.
45 data = dde.data.PDE(geom, pwatts_eq, [bc_y], 168, 168, solution = orig_pwatts_ ↵ model, num_test=100)
46
47 layer_size = [2] + [30] * 5 + [1]
48 activation = "tanh"
49 initializer = "Glorot uniform"
50 net = dde.nn.FNN(layer_size, activation, initializer)
51
52 variable_a = dde.callbacks.VariableValue(a_var, period=1000)
53 model = dde.Model(data, net)
54
55 model.compile(optimizer="adam", lr=0.001, metrics=["l2 relative error"], ↵ external_trainable_variables=[a_var])
56 losshistory, train_state = model.train(iterations=20000, callbacks=[variable_a])
57 predicted_test = model.predict(test_points)
58
59 # Predikcije manje od nule nemaju smisla. Nuluji ih.
60 predicted_test[predicted_test<0]=0

```

Као и код раније обрађених инверзних проблема који користе библиотеку *DeepXDE*, постављамо параметар као варијаблу чија се вредност добија процесом обучавања:

```
a_var = dde.Variable(-4.0)
```

Ако погледамо функцију `pvwatts_eq(x, y)`, она је готово идентична функцији `orig_pvwatts_model(x)` са [Листинг 7.1](#). Разлика је у томе што `pvwatts_eq(x, y)` не враћа вредност снаге, већ функцију губитка, као што смо већ навикли. Још једна разлика огледа се у коришћењу *TensorFlow* логике уместо условног израза:

```
eta = tf.maximum(0., tf.sign(eta)) * eta
```

Овај израз није ништа друго него услов да ако имамо нефизичку вредност $\text{eta} < 0$ поставимо да је $\text{eta}=0$. Овом формулатијом избегавамо инструкцију условног скока, која се на графичком процесору изводи дosta спорије од чистог рачуна у покретном зарезу.

У даљем току програма треба да поставимо структуру за посебан гранични услов `PointSet`, који смо већ користили у одељку [Инверзни проблем](#) (страна 39) и [Пропагација поплавног таласа у отвореном каналу](#) (страна 53):

```
train_points = np.zeros((168,2))
train_measured_production = np.zeros((168,1))
train_points[:,0] = T_data_train.to_numpy().T
train_points[:,1] = E_data_train.to_numpy().T
train_measured_production[:,0] = P_data_train.to_numpy().T

bc_y = dde.icbc.PointSetBC(train_points, train_measured_production, component=0)
```

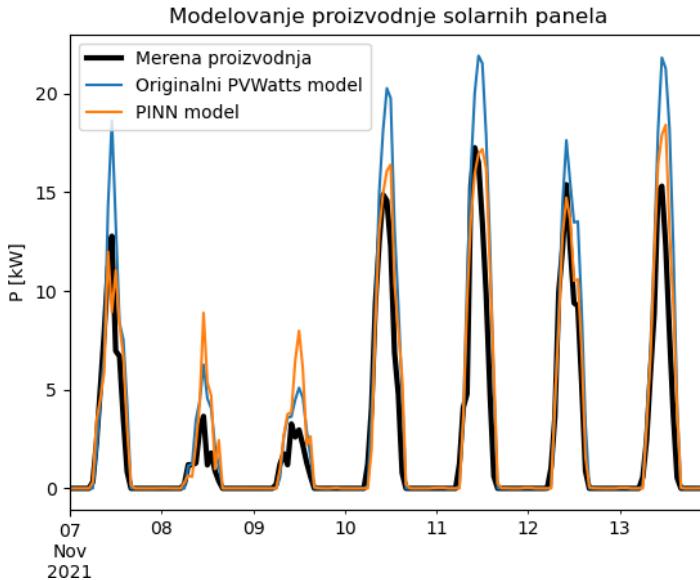
Имамо укупно 168 тачака (7×24) мерења производње које ћemo искористити у покушају да „спустимо“ вредности које даје оригинални модел. Геометрију проблема који решавамо дефинишићемо опсегом улазних варијабли температуре и укупне осунчаности, које се узимају из табеле података. Онда можемо да поставимо и колокационе тачке на домену наредбом:

```
data = dde.data.PDE(geom, pvwatts_eq, [bc_y], 168, 168, solution = orig_pvwatts_
→model, num_test=100)
```

За број случајних колокационих тачака унутар домена узели смо исти број тачака колико имамо у `PointSet` граничном услову. Није нужно да број тачака буде једнак, па остављамо читаоцу да експериментише различитим вредностима. Остатак кода је мање-више исти као код свих других примера који користе DeepXDE за решавање инверзних проблема. Ту је постављање архитектуре ФПНМ и хипер-параметара, алгоритма оптимизације, стопе обуке, `callback` функције за штампу тренутне вредности `a_var` током обуке и слично. На крају се анулирају негативне вредности предвиђене производње јер немају физичког смисла.

7.4 Резултати

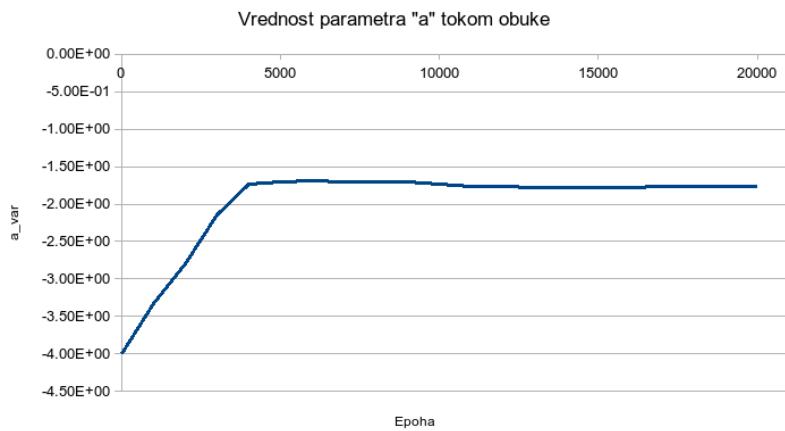
На дијаграму Сл.7.13 може видети резултат приступа описаног у одељку [Пример прорачуна производње](#) (страница 97). Ради поређења, на слици је остао приказан и график са Сл.7.12.



Слика7.13: Поређење ФПНМ модела и чистог PVWatts модела са мереном производњом

Очигледно је да ФПНМ модел даје боље резултате од чистог PVWatts модела. То потврђује и грешка која је спуштена са 2,51kW на 1,92kW. Илустрације ради, дајемо и вредност параметра `a_var` током обуке на Сл.7.14. Уместо вредности из литературе -2,98, испоставља се да подацима више одговара вредност од око -1,78.

Сигурно је да модел и даље може да се подешава, рецимо укључивањем варијације још неког параметра, као што је `gamma_pdc`. Међутим, овде се тиме нећемо бавити јер смо у доволној мери постигли циљ, тј. показали начин рада са овом врстом проблема. Читалац може да проба и неки други приступ јер су сви потребни подаци доступни у репозиторијуму практикума.



Слика 7.14: Вредност параметра a током обуке

Литература

- [Bea12] Jacob Bear. *Hydraulics of groundwater*. Courier Corporation, 2012.
- [CHB20] Jayashree Chadalawada, HMVV Herath, and Vladan Babovic. Hydrologically informed machine learning for rainfall-runoff modeling: a genetic programming-based toolkit for automatic model induction. *Water Resources Research*, 56(4):e2019WR026933, 2020.
- [Dob14] Aron P Dobos. Pvwatts version 5 manual. Technical Report, National Renewable Energy Lab.(NREL), Golden, CO (United States), 2014.
- [DCK+13] Andrew Duncan, Albert S Chen, Edward Keedwell, Slobodan Djordjevic, and Dragan Savic. Rapids: early warning system for urban flooding and water quality hazards. 2013.
- [GarzonKLT22] Alexander Garzón, Zoran Kapelan, J Langeveld, and Riccardo Taormina. Machine learning-based surrogate modelling for urban water networks: review and future research directions. *Water Resources Research*, pages e2021WR031808, 2022.
- [HJ21] Ehsan Haghighat and Ruben Juanes. Sciann: a keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, 373:113552, 2021.
- [HHM18] William F Holmgren, Clifford W Hansen, and Mark A Mikofski. Pvlib python: a python package for modeling solar energy systems. *Journal of Open Source Software*, 3(29):884, 2018.
- [Hux57] Andrew F Huxley. Muscle structure and theories of contraction. *Prog. Biophys. Biophys. Chem*, 7:255–318, 1957.
- [Ihl98] Frank Ihlenburg. *Finite element analysis of acoustic scattering*. Springer, 1998.

- [IS22] Milos Ivanovic and Visnja Simic. Efficient evolutionary optimization using predictive auto-scaling in containerized environment. *Applied Soft Computing*, 129:109610, 2022.
- [ISS+15] Milos Ivanovic, Visnja Simic, Boban Stojanovic, Ana Kaplarevic-Malisic, and Branislav Marovic. Elastic grid resource provisioning with wobingo: a parallel framework for genetic algorithm based optimization. *Future Generation Computer Systems*, 42:44–54, 2015.
- [ISS17] Milos Ivanovic, Marina Svicevic, and Svetislav Savovic. Numerical solution of stefan problem with variable space grid method based on mixed finite element/finite difference approach. *International Journal of Numerical Methods for Heat and Fluid Flow*, 2017.
- [KK19] Kian Katanforoosh and Daniel Kunin. Initializing neural networks. *DeepLearning.ai*, 2019.
- [Kojic98] Miloš Kojić. *Metod konačnih elemenata: Linearna analiza. I.* Mašinski fakultet, 1998.
- [LLF98] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9:987–1000, 1998.
- [LMMK21] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: a deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.
- [Mar21] Stefano Markidis. The old and the new: can physics-informed deep-learning replace traditional linear solvers? *Frontiers in big Data*, pages 92, 2021.
- [PU92] Dimitris C. Psichogios and Lyle H. Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38:1499–1511, 1992.
- [RPK19] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [Rec04] Gerald W Recktenwald. Finite-difference approximations to the heat equation. *Mechanical Engineering*, 2004.
- [SC09] Svetislav Savovic and James Caldwell. Numerical solution of stefan problem with time-dependent boundary conditions by variable space grid method. *Thermal Science*, 13:165–174, 2009.
- [SSI19] Visnja Simic, Boban Stojanovic, and Milos Ivanovic. Optimizing the performance of optimization in the cloud environment—an intelligent auto-scaling approach. *Future Generation Computer Systems*, 101:909–920, 2019.
- [Svivcevic20] Marina Svičević. *Višeskalni računarski model mišića zasnovan na makromodelu konačnih elemenata i Hakslijevom mikromodelu*. PhD thesis, Универзитет у Крагујевцу, Природно-математички факултет, 2020.

- [WYP22] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: a neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.