



УНИВЕРЗИТЕТ У НОВОМ  
САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ  
НАУКА У НОВОМ САДУ

---




Жарко Благојевић

# **Визуализација градијентних оптимизационих алгоритама**

Дипломски рад  
- Основне академске студије -

Нови Сад, 2022.



|   |   |              |
|---|---|--------------|
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ<br><b>ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА</b><br>21000 НОВИ САД, Трг Доситеја Обрадовића<br>6 | Датум:       |
|   | <b>ЗАДАТАК ЗА ИЗРАДУ<br/>ДИПЛОМСКОГ (BACHELOR) РАДА</b>   | Лист:<br>1/1 |

(Податке уноси предметни наставник - ментор)

|                                  |                           |
|----------------------------------|---------------------------|
| Врста студија:                   | Основне академске студије |
| Студијски програм:               | Рачунарство и аутоматика  |
| Руководилац студијског програма: | проф. др Милан Рапаић     |

|  |  |               |            |
|--|--|---------------|------------|
| Студент:   | Жарко Благојевић                         | Број индекса: | RA 44/2018 |
| Област:  | Електротехничко и рачунарско инжењерство |               |            |
| Ментор:  | Др Александар Ковачевић, проф.           |               |            |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И<br>ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ<br>РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:<br>- проблем – тема рада;<br>- начин решавања проблема и начин практичне провере резултата рада, ако<br>је таква провера неопходна;<br>- литература |  |               |            |

#### НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

|  |
|--|
| Визуализација градијентних оптимизационих алгоритама |
|--|

#### ТЕКСТ ЗАДАТКА:

|  |
|--|
| Имплементирати систем за визуализацију градијентних оптимизационих алгоритама, са нагласком на тестирање понашања алгоритама над различитим критеријумима оптималности, под утицајем различитих вредности хиперпараметара. Описати имплементиране оптимизационе алгоритме и архитектуру система. |
|--|

|                                  |              |
|----------------------------------|--------------|
| Руководилац студијског програма: | Ментор рада: |
|                                  |              |

|  |
|--|
| Примерак за: <input type="checkbox"/> - Студента; <input type="checkbox"/> - Ментора |
|--|



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|   |  |
|---|--|
| Редни број, <b>РБР</b> :  |  |
| Идентификациони број, <b>ИБР</b> :  |  |
| Тип документације, <b>ТД</b> :  | монографска публикација  |
| Тип записа, <b>ТЗ</b> :   | текстуални штампани документ   |
| Врста рада, <b>ВР</b> :   | дипломски рад  |
| Аутор, <b>АУ</b> :  | Жарко Благојевић   |
| Ментор, <b>МН</b> :   | др Александар Ковачевић, проф.   |
| Наслов рада, <b>НР</b> :  | Визуализација градијентних оптимизационих алгоритама                         |
| Језик публикације, <b>ЈП</b> :  | српски   |
| Језик извода, <b>ЈИ</b> :   | српски / енглески  |
| Земља публиковања, <b>ЗП</b> :  | Србија   |
| Уже географско подручје, <b>УГП</b> :   | Војводина  |
| Година, <b>ГО</b> :   | 2022   |
| Издавач, <b>ИЗ</b> :  | ауторски репринт   |
| Место и адреса, <b>МА</b> :   | Нови Сад, Факултет техничких наука,<br>Трг Доситеја Обрадовића 6             |
| Физички опис рада, <b>ФО</b> :<br><small>(поглавља/страна/ цитата/табела/слика/графика/прилога)</small> | 7 / 69 / 17 / 0 / 43 / 0 / 0   |
| Научна област, <b>НО</b> :  | Софтверске и информационе технологије  |
| Научна дисциплина, <b>НД</b> :  | Рачунарска интелигенција   |
| Предметна одредница/Кључне речи, <b>ПО</b> :  | Оптимизација, визуализација, градијентни оптимизациони алгоритми             |
| <b>УДК</b>  |  |
| Чува се, <b>ЧУ</b> :  | Библиотека Факултета техничких наука, Трг Доситеја<br>Обрадовића 6, Нови Сад |
| Важна напомена, <b>ВН</b> :   |  |

|                                   |   |
|-----------------------------------|---|
| Извод, <b>ИЗ:</b>                 | Рад описује систем за визуализацију градијентних оптимизационих алгоритама битних за обучавање неуронских мрежа. Описани систем пружа лак, брз и интуитиван начин за тестирање и поређење горепоменутих алгоритама над различитим критеријумима оптималности. |
| Датум прихватања теме, <b>ДП:</b> |   |
| Датум одбране, <b>ДО:</b>         |   |
| Чланови комисије,<br><b>КО:</b>   | Председник: др Јелена Сливка, ванр. проф.   |
|                                   | Члан: др Никола Лубурић, доцент   |
|                                   | Члан, ментор: др Александар Ковачевић, ред. проф.   |
|                                   | Потпис ментора  |

## KEY WORDS DOCUMENTATION

|  |  |
|--|--|
| Accession number, <b>ANO</b> :   |  |
| Identification number, <b>INO</b> :  |  |
| Document type, <b>DT</b> :   | monographic publication  |
| Type of record, <b>TR</b> :  | textual material   |
| Contents code, <b>CC</b> :   | bachelor thesis  |
| Author, <b>AU</b> :  | Žarko Blagojević   |
| Mentor, <b>MN</b> :  | Aleksandar Kovačević, full professor, PhD  |
| Title, <b>TI</b> :   | Visualization of gradient optimization algorithms                                    |
| Language of text, <b>LT</b> :  | Serbian  |
| Language of abstract, <b>LA</b> :  | Serbian / English  |
| Country of publication, <b>CP</b> :  | Serbia   |
| Locality of publication, <b>LP</b> :   | Vojvodina  |
| Publication year, <b>PY</b> :  | 2022   |
| Publisher, <b>PB</b> :   | author's reprint   |
| Publication place, <b>PP</b> :   | Novi Sad, Faculty of Technical Sciences,<br>Trg Dositeja Obradovića 6                |
| Physical description, <b>PD</b> :<br>(chapters/pages/ref./tables/pictures/graphs/appendixes) | 7 / 69 / 17 / 0 / 43 / 0 / 0   |
| Scientific field, <b>SF</b> :  | Software and Information Technologies  |
| Scientific discipline, <b>SD</b> :   | Artificial intelligence  |
| Subject/Key words, <b>S/KW</b> :   | Optimization, visualization, gradient optimization algorithms                        |
| <b>UC</b>  |  |
| Holding data, <b>HD</b> :  | Library of the Faculty of Technical Sciences, Trg<br>Dositeja Obradovića 6, Novi Sad |
| Note, <b>N</b> :   |  |

|   |  |
|---|--|
| Abstract, <b>AB</b> :                             | This bachelor thesis describes a system for visualization of gradient optimization algorithms important for training neural networks. Described system provides easy, fast and intuitive way to test and compare aforementioned algorithms with different optimization problems. |
| Accepted by the Scientific Board on, <b>ASB</b> : |  |
| Defended on, <b>DE</b> :                          |  |
| Defended Board,<br><b>DB</b> :                    | President: Jelena Slivka, associate professor, PhD   |
|   | Member: Nikola Luburić, assistant professor, PhD   |
|   | Member, Mentor: Aleksandar Kovačević, full professor, PhD  |
|   | Mentor's signature   |



## САДРЖАЈ

|   |    |
|---|----|
| КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА.....                   | 5  |
| KEY WORDS DOCUMENTATION.....                              | 7  |
| 1. УВОД .....   | 11 |
| 2. ПРЕГЛЕД АЛГОРИТАМА И СЛИЧНИХ СИСТЕМА .....             | 13 |
| 2.1 Теоријски преглед алгоритама.....                     | 13 |
| 2.1.1 Градијент .....                                     | 13 |
| 2.1.2 Метод најбржег спуста.....                          | 13 |
| 2.1.3 Метод најбржег спуста са моментом .....             | 14 |
| 2.1.4 Метод најбржег спуста Нестерова .....               | 15 |
| 2.1.5 Метод адаптивног градијента (AdaGrad).....          | 16 |
| 2.1.6 Метод пропагирања корена средње грешке (RMSProp)... | 17 |
| 2.1.7 Метод адаптивне брзине обуке (ADADELTA) .....       | 18 |
| 2.1.8 Метод адаптивне процене момента (Adam) .....        | 19 |
| 2.2 Преглед сличних система .....                         | 20 |
| 3. КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ.....                  | 23 |
| 3.1 Vue.js радни оквир.....                               | 23 |
| 3.2 Plotly.js.....  | 23 |
| 3.3 D3.js .....   | 24 |
| 3.4 Math expression evaluator.....                        | 24 |
| 3.5 MathJax .....   | 24 |
| 4. СПЕЦИФИКАЦИЈА .....                                    | 25 |
| 4.1 Спецификација захтева .....                           | 25 |
| 4.1.1 Функционални захтеви.....                           | 25 |
| 4.1.2 Нефункционални захтеви.....                         | 29 |
| 5. ИМПЛЕМЕНТАЦИЈА.....                                    | 31 |
| 5.1 Компонента ModalDialog.vue .....                      | 31 |
| 5.1.1 Параметри компоненте.....                           | 32 |
| 5.1.2 Шаблон компоненте .....                             | 34 |
| 5.2 Компонента OptimizerPicker.vue.....                   | 35 |
| 5.2.1 Параметри компоненте.....                           | 36 |
| 5.2.2 Шаблон компоненте .....                             | 38 |
| 5.2.3 Имплементација оптимизационих алгоритама .....      | 40 |
| 5.3 Компонента FunctionPicker.vue .....                   | 42 |
| 5.3.1 Параметри компоненте.....                           | 42 |
| 5.3.2 Компонента SurfacePlot.vue .....                    | 44 |
| 5.3.3 Шаблон компоненте .....                             | 47 |
| 5.3.4 Модални дијалози критеријума оптималности.....      | 49 |
| 5.4 Компонента ContourPlot.vue.....                       | 51 |
| 5.4.1 Параметри компоненте.....                           | 51 |

|       |   |    |
|-------|---|----|
| 5.4.2 | Шаблон компоненте.....                        | 51 |
| 5.4.3 | Цртање контурног дијаграма и алгоритама ..... | 52 |
| 5.5   | Интеракција главних компоненти .....          | 57 |
| 6.    | ДЕМОНСТРАЦИЈА .....                           | 59 |
| 6.1   | Сценарио 1.....                               | 59 |
| 6.2   | Сценарио 2.....                               | 63 |
| 7.    | ЗАКЉУЧАК.....                                 | 65 |
|       | ЛИТЕРАТУРА.....                               | 67 |
|       | БИОГРАФИЈА.....                               | 69 |

## 1. УВОД

Машинско учење, као једна од тренутно најпопуларнијих и најзанимљивијих области истраживања вештачке интелигенције са интензивном применом у разним индустријама, представља саставни део живота све већег броја људи. Аутономна возила, системи за препоруку производа или садржаја на друштвеним мрежама, софтверски агенти који одговарају на постављена питања (енгл. *chatbot*) представљају само неке од бројних примена алгоритама машинског учења. Посебно успешне у решавању овако комплексних проблема су неуронске мреже, специјална класа алгоритама машинског учења. Пре него што се неуронска мрежа може користити, неопходно ју је прво обучити за решавање конкретног проблема. Обучавање неуронске мреже није нимало лак процес и своди се на оптимизацију специјално дефинисане функције грешке (критеријума оптималности) у вишедимензионалном простору (и до неколико милијарди димензија). Стога, одабир адекватног оптимизационог алгорита је често од пресудног значаја за успешно обучавање неуронске мреже.

Одговор на питање: „Како одабрати адекватан оптимизациони алгоритам за конкретан проблем?“ крије се у разумевању начина на који дати алгоритми функционишу, односно познавању њихових предности и мана. Визуализација често служи за приказ интуиције иза концепта који се визуализује и у многome доприноси бољем поимању самог концепта. Визуални приказ рада различитих оптимизационих алгоритама над истим критеријумом оптималности олакшао би њихово међусобно поређење и у први план истакао специфичности сваког алгорита.

Визуализација градијентних оптимизационих алгоритама, посебне класе оптимизационих алгоритама који се најчешће користе при обучавању неуронских мрежа, као и њихово поређење над разним критеријумима оптималности у тродимензионалном простору представља специфичан проблем чије решење је описано у овом раду.

Жеља да визуализатор што лакше буде доступан свима који су заинтересовани за градијентне оптимизационе алгоритме допринела је

да се решење имплементира у програмском језику Јаваскрипт (енгл. *JavaScript*), како би било доступно на интернету. Имплементирано је 7 градијентних оптимизационих алгоритама, где корисник може да бира који алгоритми ће се визуализовати и подешава хиперпараметре сваког од појединачних алгоритама. Централни део апликације представља контурни дијаграм критеријума оптималности, где кликом на њега корисник бира почетну тачку и започиње извршавање одабраних алгоритама. Кориснику је приказан критеријум оптималности и у облику површинског тродимензионалног графика и омогућен избор предефинисаних критеријума оптималности, као и дефинисање нових критеријума.

Специфичност решења огледа се у жељи да се корисник што мање ограничава. Корисник сам бира који се алгоритми извршавају, може да подешава хиперпараметре и бира или дефинише критеријум оптималности. Посебна пажња посвећена је прегледности дијаграма и анимацији цртања оптимизационе путање где су кружићима у разним бојама наглашени кораци одговарајућих алгоритама.

Рад је организован по следећим поглављима: Увод, Преглед алгоритама и сличних система, Коришћене софтверске технологије, Спецификација, Имплементација, Демонстрација и Закључак.

## 2. ПРЕГЛЕД АЛГОРИТАМА И СЛИЧНИХ СИСТЕМА

У овом поглављу описан је теоријски преглед градијентних оптимизационих алгоритама и система сличних апликацији описаној у раду.

### 2.1 Теоријски преглед алгоритама

У овом поглављу биће описане теоријске основе алгоритама који су имплементирани у самом систему. Како се сви имплементирани алгоритми заснивају на математичком концепту градијента, прво ће бити описано шта он представља, а затим како је искоришћен у овим алгоритмима.

#### 2.1.1 Градијент

Градијент [1] скаларне диференцијабилне функције више променљивих  $f$  представља вектор колоне парцијалних извода те функције.

$$\nabla f = \left[ \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n} \right]^T \quad (1)$$

Градијентни вектор  $\nabla f$  описује правац и стопу најбржег раста функције  $f$ . Правац најбржег раста огледа се у правцу градијентног вектора, док се стопа најбржег раста огледа у његовом интензитету.

#### 2.1.2 Метод најбржег спуста

Метод најбржег спуста (енгл. gradient/steepest descent) [2] представља најједноставнију градијентну методу оптимизације. Аналогно томе да градијент описује правац и стопу најбржег раста функције, негативни градијент описује правац и стопу најбржег пада функције. Ово може да се искористи у налажењу минимума функције, тако што ће се алгоритам стално кретати у правцу негативног градијента. Општи корак методе најбржег спуста приказан је једначином (2).

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \quad (2)$$

Нова позиција алгоритма рачуна се тако што се вектор тренутне позиције сабере са негативним градијентом кога множи хиперпараметар  $\alpha$ . Хиперпараметри представљају предефинисане (не рачунате) параметре оптимизационог алгоритма, који су овакав назив добили како би се разликовали од осталих параметара који се ажурирају у процесу оптимизације. Хиперпараметар  $\alpha$  назива се још и брзина или стопа обуке (енгл. *learning rate*) и бира се из опсега (0, 1). Мала брзина обуке утиче на то да алгоритам уситни кораке и тако успори процес оптимизације. Велика брзина обуке утиче на то да алгоритам прави велике скокове и потенцијално може да промаши оптимум (дивергира). Зато је одабир одговарајуће вредности за хиперпараметар  $\alpha$  од суштинске важности за правилан рад овог алгоритма.

Основни проблем овог алгоритма је што не поседује никакав механизам који ће спречити „заглављивање“ у регијама превоја и локалних оптимума. Наиме, око превојних тачака и локалних оптимума функције интензитет градијента је веома мали, што утиче на то да се корак знатно уситни. Стога, алгоритам често може да се заустави пре него што је нашао глобални оптимум.

### 2.1.3 Метод најбржег спуста са моментом

Метод најбржег спуста са моментом покушава да превазиђе поменути проблем прераног заустављања, тако што моделује инерцију кретања алгоритма, односно тенденцију алгоритма да неко време (у неколико наредних итерација) настави да се креће у правцу у коме се у претходним корацима кретао. Овај приступ први је у свом раду предложио Борис Пољак [3]. Интуицију иза овог приступа Пољак објашњава метафором пуштања мале тешке лопте низ брдо. Наиме, лопта што је тежа, има већу тенденцију да настави да се котрља у истом правцу низ нагиб брда, тако прелазећи преко неравнина и удубљења у брду. У овој метафори кретање мале тешке лопте представља понашање алгоритма, брдо представља функцију која се оптимизује (критеријум оптималности), док неравнине и удубљења представљају регије превоја и локалних оптимума. Општи корак ове методе приказује једначина (3).

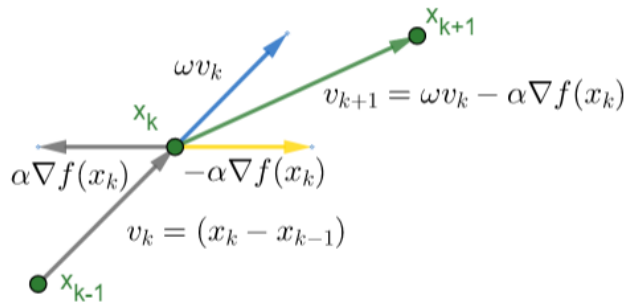
$$x_{k+1} = x_k - \alpha \nabla f(x_k) + \omega(x_k - x_{k-1}) \quad (3)$$

Новина овог алгоритма у односу на основну верзију методе најбржег спуста јесте додаток члана са коефицијентом  $\omega$  (хиперпараметар моменат). Вектор  $x_k - x_{k-1}$  представља искорак алгоритма, чији утицај се, са уделом  $\omega$  ( $0 < \omega < 1$ ), преноси у наредну итерацију алгоритма.

У пракси, једначина (3) често се дели у два корака

$$\begin{aligned} v_{k+1} &= \omega v_k - \alpha \nabla f(x_k) \\ x_{k+1} &= x_k + v_{k+1} \end{aligned} \quad (4)$$

где променљива  $v$  памти искорак алгоритма (Слика 2.1).



Слика 2.1 Визуализација методе најбржег спуста са моментом у равни

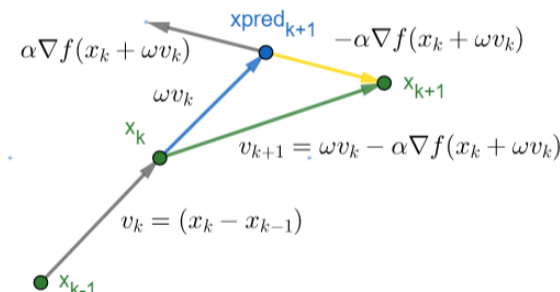
Иако увођење момента убрзава конвергенцију алгоритма и пружа механизам превазилажења регија превоја и локалних оптимума, недостатак овог алгоритма јесте што постоји велика шанса да због момента алгоритам промаши глобални оптимум и дивергира.

#### 2.1.4 Метод најбржег спуста Нестерова

Метод најбржег спуста Нестерова, познатији под називом „Убрзани градијент Нестерова“ (енгл. *NAG – Nesterov accelerated gradient*) [4], представља директну модификацију претходног приступа. Суштинска разлика јесте та да се градијент не рачуна у тренутној тачки, већ у претпостављеној будућој тачки (енгл. *lookahead*). С тим у виду, (4) се модификује на следећи начин

$$\begin{aligned} v_{k+1} &= \omega v_k - \alpha \nabla f(x_k + \omega v_k) \\ x_{k+1} &= x_k + v_{k+1} \end{aligned} \quad (5)$$

Вектор  $x_k + \omega v_k$  представља претпостављену будућу тачку до које би алгоритам стигао да је наставио да се креће у правцу  $v_k$  под утицајем момента  $\omega$  (Слика 2.2).



Слика 2.2 Визуализација убрзаног градијента Нестерова

Предиктивни карактер овог алгоритма доприноси стабилизацији алгоритма са моментом и побољшава његову конвергенцију.

### 2.1.5 Метод адаптивног градијента (AdaGrad)

Чест случај са критеријумом оптималности јесте да му се вредност не мења једнаким интензитетом по различитим осама (променљивама).

У произвољном региону датог критеријума мале промене једне променљиве могу да утичу на велике промене вредности критеријума (парцијални извод је по тој променљивој велик). Ту променљиву, у том региону, треба мењати полако, односно смањити искорак алгоритма по тој оси, како наглим скоковима алгоритам не би промашио оптимум.

У том истом региону, мале промене неке друге променљиве могу незнатно да утичу на промену вредности критеријума (парцијални извод је по тој променљивој мали). Ту променљиву, у том региону, треба мењати нагло, односно повећати искорак алгоритма по тој оси, како би се наглим скоковима убрзала његова конвергенција.

Суштински проблем свих претходно описаних алгоритама јесте да не поседују флексибилност подешавања искорака по појединачним осама. Ово се огледа у томе што се при ажурирању позиције градијент, односно сваки **парцијални извод**, множи истом брзином обуке  $\alpha$ .



Решење овог проблема било би да се сваки парцијални извод множи посебно дефинисаним коефицијентом. Овај коефицијент треба да буде мали када је одговарајући парцијални извод велик (решен проблем промашаја оптимума), а велик када је одговарајући парцијални извод мали (решен проблем споре конвергенције).

Метод адаптивног градијента (енгл. *AdaGrad – Adaptive gradient*) [5], имплементира описано решење, дефинишући посебне коефицијенте који се прилагођавају променама одговарајућих парцијалних извода.

Прво се дефинише  $g_{k,i}$  као парцијални извод по  $i$ -тој променљивој у  $k$ -тој итерацији

$$g_{k,i} = \nabla f(x_k)_i = \frac{\partial f(x_k)}{\partial x_i} \quad (6)$$

Затим се дефинише  $G_{k,i}$  као сума квадрата парцијалних извода по  $i$ -тој променљивој у првих  $k$  итерација.

$$G_{k,i} = \sum_{m=0}^k g_{m,i}^2 \quad (7)$$

Коначно, општи корак алгоритма дефинисан је са

$$x_{k+1,i} = x_{k,i} - \frac{\alpha}{\sqrt{G_{k,i}} + \varepsilon} g_{k,i} \quad (8)$$

где је  $\varepsilon$  јако мали број ( $10^{-6}$ ,  $10^{-8}$ ) који спречава дељење са нулом.

Сума (7) ће расти већом брзином ако је (6) из итерације у итерацију велик број, док ће у супротном случају стагнирати. Како је у једначини (8) коефицијент који множи (6) реципрочан корену ове суме, то значи да ће коефицијент расти како одговарајући парцијални извод опада, а опадати како одговарајући парцијални извод расте.

### 2.1.6 Метод пропагирања корена средње грешке (RMSProp)

Проблем са методом адаптивног градијента огледа се у суми (7). Како су сабирци квадрирани, самим тим и позитивни, ова сума непрекидно

расте, што значи да током времена коефицијент опада и смањују се кораци које алгоритам извршава (енгл. *vanishing gradient problem*).

Овај проблем покушава да превазиђе метод пропагирања корена средње грешке, (*RMSprop – root mean squared propagation*) предложен у предавању Џефрија Хинтона. Уместо да се градијенти акумулирају неограничено, акумулирају се помоћу експоненцијалног покретног просека (енгл. *EMA – exponentially weighted moving average*). Циљ ове модификације јесте да се при рачунању суме (7) не узимају сви градијенти са истом тежином, него да се веће тежине (самим тим и утицај) придружују градијентима који су били у скоријим итерацијама.

Ова модификација утиче само на једначину (7) која постаје

$$G_{k,i} = \omega G_{k-1,i} + (1 - \omega) g_{k,i}^2 \quad (9)$$

где се  $\omega = 0.9$  показала као најбоља вредност у пракси.

### 2.1.7 Метод адаптивне брзине обуке (ADADELTA)

Метод адаптивне брзине обуке (енгл. *ADADELTA - Adaptive Learning Rate Method*) [6], представља још један приступ решавању проблема присутног код метода адаптивног градијента. Велика предност овог алгоритма у односу на *RMSProp* јесте та да није неопходно експлицитно задати хиперпараметар  $\alpha$ , већ се он рачуна за сваку итерацију алгоритма.

Прво се израчуна  $G_{k,i}$  као у једначини (9). Затим се дефинише искорак тренутне итерације

$$\Delta x_{k,i} = - \frac{\sqrt{D_{k-1,i} + \varepsilon}}{\sqrt{G_{k,i} + \varepsilon}} g_{k,i} \quad (10)$$

а затим ажурира

$$\begin{aligned} D_{k,i} &= \omega D_{k-1,i} + (1 - \omega) \Delta x_{k,i}^2 \\ x_{k+1,i} &= x_{k,i} + \Delta x_{k,i} \end{aligned} \quad (11)$$

Битно је приметити да променљива  $D$  садржи експоненцијални покретни просек квадрата искорака, на основу кога се и дефинише прилагодљива брзина обуке.

### 2.1.8 Метод адаптивне процене момента (Adam)

Метод адаптивне процене момента (енгл. *Adam – adaptive moment estimation*) [7] је у пракси најзаступљенији од свих до сада описаних алгоритама. Настао је под утицајем алгоритама *AdaGrad* и *RMSProp*, комбинујући добро понашање са проређеним градијентима (енгл. *sparse gradient*) које испољава *AdaGrad*, са идејом употребе експоненцијалног покретног просека квадрата градијента који уводи *RMSProp*.

Алгоритам прво ажурира  $m_k$  и  $v_k$  – експоненцијални покретни просек градијента и квадрата градијента, респективно

$$\begin{aligned} m_k &= \omega_1 m_{k-1} + (1 - \omega_1) g_k \\ v_k &= \omega_2 v_{k-1} + (1 - \omega_2) g_k^2 \end{aligned} \quad (12)$$

Променљива  $m_k$  заправо представља апроксимацију просека градијента, док променљива  $v_k$  представља апроксимацију нецентриране варијансе. Како се вектори ових променљивих иницијализују 0 вредностима, апроксимација просека градијента вуче ка 0, посебно у почетним итерацијама. Како би се превазишао овај проблем, врши се корекција променљивих

$$\begin{aligned} \hat{m}_k &= \frac{m_k}{1 - \omega_1} \\ \hat{v}_k &= \frac{v_k}{1 - \omega_2} \end{aligned} \quad (13)$$

а затим се помоћу њих ажурира тренутна позиција

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{\hat{v}_k} + \epsilon} \hat{m}_k \quad (14)$$

Алгоритам уводи додатне параметре  $\omega_1$  и  $\omega_2$ , који у пракси узимају вредности 0.9 и 0.999, респективно.

## 2.2 Преглед сличних система

Системи описани у овом поглављу су својим занимљивим изгледом, функционалностима, едукативним карактером, али и ограничењима у великој мери допринели коначном изгледу описаног визуализатора.

Предавање др. Милана Рапаића из предмета Методе оптимизације под називом „Вишедимензиона нумерика – градијентни алгоритми“ представља основну инспирацију за овај рад. Редослед приказивања алгоритама пружа јасан увид како сваки алгоритам представља природан искорак у односу на претходни, иако су неки од њих настајали паралелно и независно. Сви алгоритми описани у предавању су имплементирани у самој апликацији. Иако овде није реч о систему, већ о предавању одржаном уз помоћ Јупитер окружења (енгл. *Jupyter notebook*), настала је жеља да се извршавање алгоритама учини више интерактивним и поједностави тестирање оптимизационих алгоритама над произвољним критеријумом оптималности.

Апликација Клемента Михаилеска (енгл. Clément Mihăilescu) под називом Визуализатор алгоритама претраге [8] на интерактиван, једноставан и визуално привлачан начин приближава интуицију иза најпознатијих алгоритама претраге. Апликација пружа кориснику могућност избора конкретног алгоритма претраге који се визуализује, избора почетне и крајње позиције претраге, исцртавање произвољних зидова који блокирају простор претраге и тиме је отежавају, избор предефинисаних лавирината и подешавање брзине извршавања претраге. Интерактивност и лепота графичког интерфејса ове апликације утицало је да слично решење настане и за градијентне оптимизационе алгоритме.

Апликација и блог написан од стране Емилијана Дупона (франц. Emilien Dupont) под називом Интерактивна визуализација оптимизационих алгоритама у дубоком учењу (енгл. *deep learning*) [9] помогла је у обликовању апликације описане у овом раду. Идеја избора почетне тачке и извршавања алгоритама оптимизације кликом на контурни дијаграм инспирисана је Емилијановом апликацијом. Недостаци ове апликације испољавају се кроз:

- Мање прегледан контурни дијаграм
  - Нису видљиве осе независних променљивих  $x$  и  $y$
  - Скала боја примењена на контуре дијаграма, где боје иду од тамније ка светлијим нијансама како вредност критеријума оптималности расте, интуитивније би дочарала осећај дубине
  - Оптимизациона путања не наглашава конкретне кораке које оптимизациони алгоритми врше већ само путању коју прелазе
- Ограничење оптимизационих алгоритама
  - Кориснику није омогућено подешавање хиперпараметара алгоритама
  - Кориснику је понуђена визуализација само 4 градијентна оптимизациона алгоритама
- Немогућност корисника да бира или дефинише критеријум оптималности



## 3. КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ

У овом поглављу биће описане технологије које су коришћене за имплементацију система. У поглављу 3.1 описан је Јаваскрипт радни оквир Vue.js. У поглављу 3.2 описана је Јаваскрипт библиотека *plotly.js*. У поглављу 3.3 описана је Јаваскрипт библиотека *D3.js*. У поглављу 3.4 описана је Јаваскрипт библиотека *math-expression-evaluator*. У поглављу 3.5 описана је Јаваскрипт библиотека MathJax.

### 3.1 Vue.js радни оквир

Vue.js [10] представља модеран, једноставан и лако прилагодљив Јаваскрипт радни оквир за изградњу веб апликација. Заснива се на коришћењу основних веб стандарда, као што је HTML, CSS и Јаваскрипт, како би на интуитиван начин решио бројне проблеме присутне у свим веб апликацијама. Понос овог радног оквира представља реактиван, високо оптимизован систем за исцртавање веб страница (енгл. *rendering engine*), који омогућава моментално усклађивање стања графичког корисничког интерфејса (енгл. *GUI – grafical user interface*), са стањем модела података који се налази у његовој основи. Лакоћа креирања поново употребљивих компоненти интерфејса пружа слободу корисницима овог радног оквира да изглед апликације изделе у мале, лако одрживе делове које је лако прилагодити новим захтевима крајњих корисника.

### 3.2 Plotly.js

Plotly.js [11] представља Јаваскрипт библиотеку за цртање графика. Изграђена помоћу библиотека D3.js и stack.gl, ова библиотека имплементира око 40 врста графика у које спадају тродимензионални, статистички, финансијски графици као и скалабилне (енгл. *SVG – Scalable Vector Graphics*) мапе. Цртање графика је декларативно и своди се на одабир типа графика, прослеђивања одговарајућих података за цртање и подешавању великог броја опција изгледа графика. Ова библиотека на брз и ефикасан начин исцртава графике, како при самом дефинисању, тако и при измени података везаних за сам график. Библиотека пружа могућност анимације графика и додавање сопствених елемената на графике, како би их учинила што више интерактивним.

### 3.3 D3.js

D3.js (D3 - data driven documents) [12] представља Јаваскрипт библиотеку за манипулацију документима на основу података. Ова библиотека омогућава везивање произвољних података за објектни модел документа (енгл. *DOM – document object model*) и примену различитих трансформација над документом, вођеним овим подацима. Библиотека користи пун потенцијал постојећих веб стандарда (HTML, CSS, SVG) како би ефикасно манипулисала изменама садржаја и изгледа документа. Функционални приступ који користи ова библиотека природно подстиче писање декларативног, сажетог и поново употребљивог кода. D3 библиотека имплементирана је кроз скуп од 53 модула, с циљем да корисници у своје пројекте укључе само онај подскуп модула који им је неопходан за решавање конкретног проблема.

### 3.4 Math expression evaluator

Math expression evaluator [13] представља малу, уско специјализовану Јаваскрипт библиотеку за ефикасну евалуацију математичких израза. Библиотека парсира математички израз у облику низа карактера (енгл. *string*) и преводи га у извршиви Јаваскрипт израз. Дати низ карактера чине предефинисани или кориснички дефинисани токени различитог типа (функције, променљиве, константе), које библиотека препознаје и мења одговарајућим Јаваскрипт изразом. Уколико је парсирање неуспешно, евалуација израза није могућа, те се пријављује одговарајућа грешка.

### 3.5 MathJax

MathJax [14] представља Јаваскрипт библиотеку за приказ математичких формула са подршком за велики број претраживача. Ова библиотека подржава стандардизован начин за приказивање математичких формула задатих у синтаксама као што су TeX, MathML или ASCIImath. Формуле су доступне великом броју корисника због добре интеграције са читачима екрана (енгл. *screenreader*) и софтвером попут Office пакета и LaTeX алата.



## 4. СПЕЦИФИКАЦИЈА

### 4.1 Спецификација захтева

Ово поглавље садржи опис свих функционалних и нефункционалних захтева који систем треба да омогући.

#### 4.1.1 Функционални захтеви

Систем корисницима треба да пружи лак, брз и интуитиван начин за тестирање и поређење градијентних оптимизационих алгоритама над различитим критеријумима оптималности. Анализом циља система издвојени су следећи случајеви коришћења:

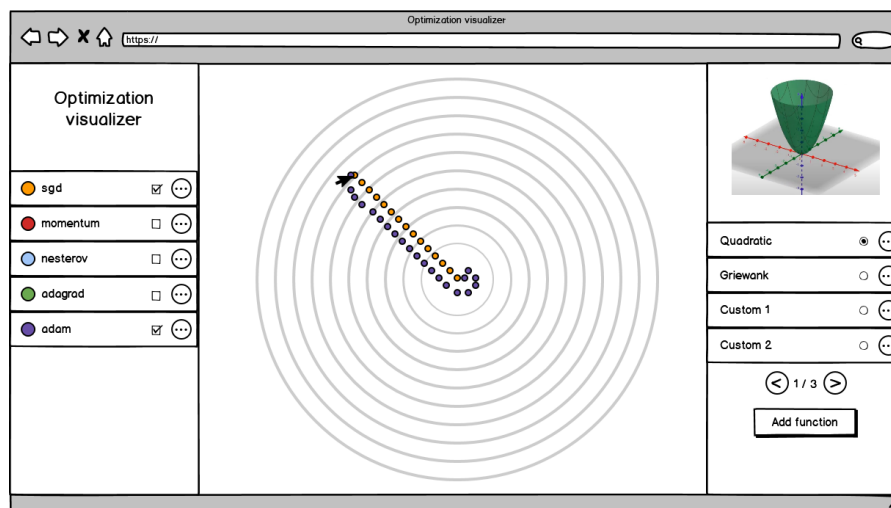
- Визуализација оптимизационих алгоритама
- Подешавање хиперпараметара оптимизационих алгоритама
- Подешавање параметара предефинисаних критеријума оптималности
- Дефинисање и ажурирање нових критеријума оптималности

#### Визуализација оптимизационих алгоритама

*Случај коришћења:* Корисник може да бира један или више оптимизационих алгоритама, дефинише заједничку почетну тачку алгоритама и започне њихову визуализацију над одабраним критеријумом оптималности.

*Сценарио:* Корисник прегледа избор оптимизационих алгоритама. Када је одабрао алгоритме за визуализацију, корисник прегледа избор критеријума оптималности и бира један над којим жели да тестира одабране алгоритме. Претходна два корака могу бити извршена у произвољном редоследу. График изабраног критеријума оптималности приказује се кориснику. Кликом на било коју тачку графика, корисник задаје заједничку почетну тачку за све оптимизационе алгоритме и започиње њихову визуализацију. На графику се исцртавају путање које су алгоритми у процесу оптимизације израчунали. Слика 4.1 приказује овај случај коришћења.

*Изузеци:* Уколико корисник није одабрао ниједан алгоритам, обавештава се да мора одабрати бар један за успешну визуализацију.

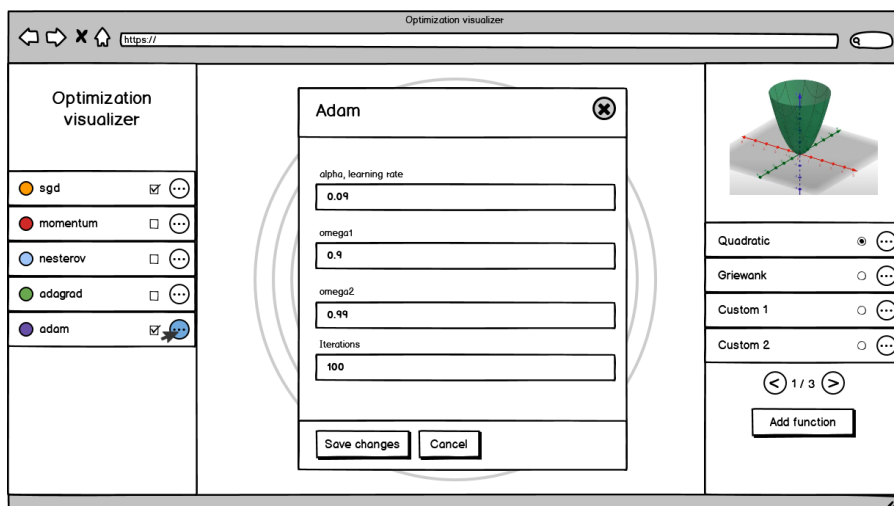


Слика 4.1 Скица система након визуализације оптимизационих алгоритама

### Подешавање хиперпараметара оптимизационих алгоритама

*Случај коришћења:* Корисник може да подешава хиперпараметре било ког од имплементираних оптимизационих алгоритама. Овај случај коришћења омогућава кориснику да тестира како хиперпараметри утичу на понашање оптимизационих алгоритама.

*Сценарио:* Корисник прегледа избор оптимизационих алгоритама. Када је одлучио ком алгоритму жели да мења хиперпараметре, корисник активира панел са додатним информацијама о конкретном алгоритму. У оквиру панела приказана је форма попуњена тренутним вредностима хиперпараметара алгоритма. Корисник може да мења вредности хиперпараметара и да те измене сачува или одустане од њих. Слика 4.2. приказује овај случај коришћења.

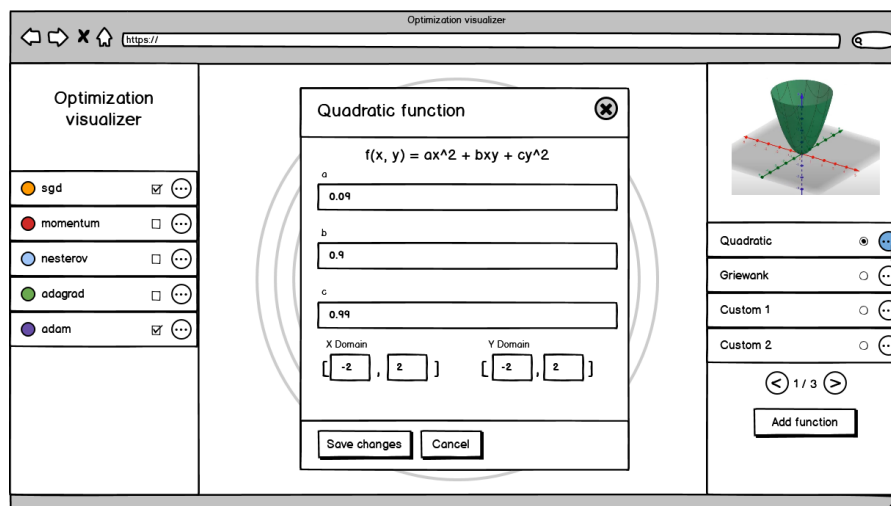


Слика 4.2 Скица система током измене хиперпараметара оптимизационих алгоритама

### Подешавање параметара предефинисаних критеријума оптималности

*Случај коришћења:* Корисник може да подешава параметре предефинисаних критеријума оптималности.

*Сценарио:* Корисник прегледа избор предефинисаних критеријума оптималности. Када је одлучио ком критеријум жели да мења параметре, корисник активира панел са додатним информацијама о конкретном критеријуму оптималности. У оквиру панела приказана је математичка формула критеријума оптималности и форма попуњена тренутним вредностима параметара критеријума, као и вредностима граница домена независних променљивих  $x$  и  $y$ . Корисник може да мења вредности наведених параметара и да те измене сачува или одустане од њих. Уколико дође до измене параметара, ажурира се изглед графика како би се новонастале измене правилно одразиле. Слика 4.3 приказује овај случај коришћења.



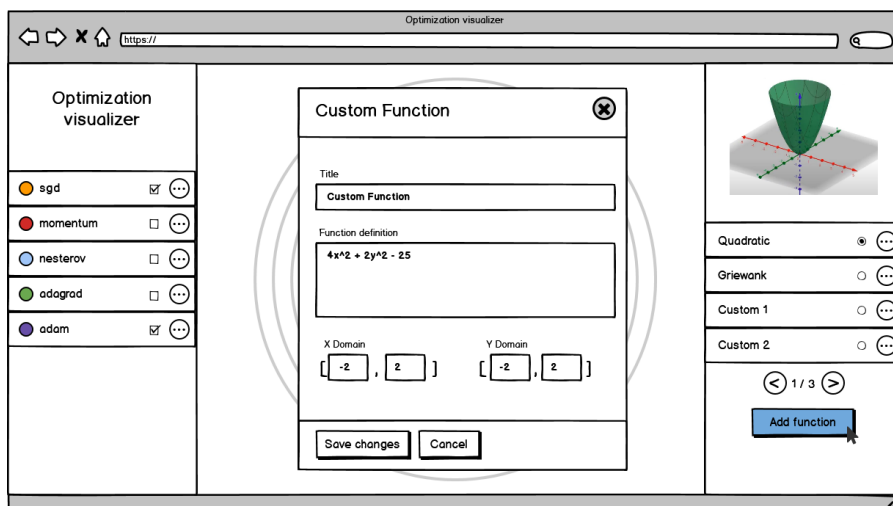
Слика 4.3 Скица система током измене параметара предефинисане функције

## Дефинисање и ажурирање нових критеријума оптималности

*Случај коришћења:* Корисник може да дефинише и ажурира сопствене критеријуме оптималности.

*Сценарио:* Корисник жели да тестира понашање оптимизационих алгоритама над произвољним критеријумом оптималности. Активацијом панела за додавање новог критеријума оптималности, кориснику се приказује форма (попуњена подразумеваним подацима) која омогућава унос назива новог критеријума оптималности, одговарајуће математичке формуле и границе домена независних променљивих  $x$  и  $y$ . Корисник може да мења вредности подразумеваних параметара и да те измене сачува или одустане од њих. Након потврде измена, у систем је додат нови критеријум оптималности који је могуће накнадно ажурирати. Слика 4.4 приказује овај случај коришћења.

*Изузеци:* Уколико корисник унесе неодговарајуће параметре (не попуни назив или унесе математички неисправну функцију), снимање измена биће онемогућено, док ће одустајање од измена вратити дати критеријум у најскорије валидно стање.



Слика 4.4 Скица система током дефинисања новог критеријума оптималности

#### 4.1.2 Нефункционални захтеви

Нефункционални захтеви свде се на корисничко искуство при коришћењу апликације (енгл. *UX - user experience*).

Кориснички интерфејс мора експлицитно да наглашава који алгоритми су одабрани за визуализацију, и којом бојом ће бити исцртане путање које су алгоритми срачунали. Путању оптимизационог алгоритма неопходно је приказати као комбинацију линије и тачака. Тачке наглашавају кораке које је алгоритам извршио, док линија пролази кроз тачке, повезујући их. Анимација којом се исцртавају ове путање, иако процесно захтевна, поготово када су одабрани сви алгоритми, мора бити течна.

Ради боље прегледности и лакшег поимања критеријума оптималности, кориснику је неопходно приказати критеријум оптималности помоћу контурног дијаграма као и помоћу тродимензионалног површинског графика.

Измена параметара оптимизационих алгоритама или критеријума оптималности мора бити испраћена обавештењем које потврђује да је измена прихваћена од стране система.



## 5. ИМПЛЕМЕНТАЦИЈА

Ово поглавље садржи опис имплементације система специфицираног у претходном поглављу. Објашњено је од којих компоненти се састоји кориснички интерфејс апликације, како оне међусобно интерагују и на који начин су искоришћене технологије описане у поглављу 3.

Као и многи модерни радни оквири за Јаваскрипт, Vue.js у први план ставља организацију корисничког интерфејса у компоненте. Решење је имплементирано као класична једнострана апликација (енгл. *SPA – single page application*), где главну страницу `HomeView.vue` чине четири основне компоненте:

- `ModalDialog.vue` – Базна компонента која моделује уопштено понашање и изглед модалног дијалога
- `OptimizerPicker.vue` - Компонента за преглед, одабир и измену хиперпараметара оптимизационих алгоритама
- `FunctionPicker.vue` - Компонента за преглед, одабир и измену параметара критеријума оптималности
- `ContourPlot.vue` - Компонента за приказ интерактивног контурног дијаграма одабраног критеријума, зависна од података из претходне две компоненте

Како би интеракција ових компоненти била јаснија, неопходно је прво описати сваку од њих, шта од података очекују и како интерно функционишу.

### 5.1 Компонента `ModalDialog.vue`

Ова компонента присутна је у компонентама `OptimizerPicker.vue` и `FunctionPicker.vue`, те ће прва бити објашњена. У систему се модални дијалози користе у две сврхе:

- Приказ додатних информација и измена хиперпараметара оптимизационих алгоритама (`OptimizerPicker.vue`)
- Приказ додатних информација и измена параметара критеријума оптималности (`FunctionPicker.vue`)

Сваки засебан дијалог у систему у својој основи користи ову компоненту, тако што њено основно понашање и изглед проширује за специфичне потребе.

### 5.1.1 Параметри компоненте

Параметри који се прослеђује компоненти у Vue.js радном оквиру називају се *props* и осликавају како спољни свет види дату компоненту. Ови параметри представљају податке потребне за рад компоненте. Постоје две врсте параметара:

- Једносмерни – искључиво родитељска компонента мења вредност ових параметара и те промене се одражавају и у компоненти детету
- Двосмерни – и родитељска компоненте и дете компонента мењају вредности овим параметрима, тако да се промене одражавају на другу компоненту

За параметре компоненте могуће је дефинисати којег су типа (енгл. *type*), да ли су неопходни за рад компоненте (енгл. *required*), која им је подразумевана вредност (енгл. *default*) и валидирајућу функцију (енгл. *validator*).

Једносмерни параметри компоненте `ModalDialog.vue` су наслов модалног дијалога и четири функције које се позивају у различитим фазама животног циклуса модалног дијалога: при отварању дијалога, снимању измена, отказивању измена и затварању дијалога (без обзира да ли су измене снимљене или не).

При отварању свих модалних дијалога у систему, прослеђена је функција која фокусира прво поље за унос параметара. При снимању измена или отказивању измена дијалога користи се концепт из Vue.js радног оквира под називом *mixin*. *Mixin* представља генеричку логику која се може искористити у различитим компонентама, тако што се дефинише издвојено, а онда увезује (енгл. *embed*) у компоненте којима је та логика потребна. У конкретном примеру генеричке логики чувања или отказивања измена, дефинисан је `modal-form-mixin.js` чији садржај приказује Листинг 5.1.1.



```
import { deepCopy } from "@utils/deep-copy.js";
export default {
  data() {
    return {
      savedForm: {},
    };
  },
  created() {
    this.savedForm = deepCopy(this.form);
  },
  methods: {
    updateForm() {
      this.savedForm = deepCopy(this.form);
    },
    resetForm() {
      this.form = deepCopy(this.savedForm);
    },
  },
};
```

Листинг 5.1.1 Генеричка логика чувања и отказивања измена параметара

Компонента која увезује `modal-form-mixin.js` треба да дефинише само поље `form` у оквиру повратне вредности `data` функције. Ово поље је Јаваскрипт објекат, чији атрибути су параметри (везани за конкретну форму) које корисник мења. Овај *mixin* потом дефинише своју копију форме под називом `savedForm` који служи као званично стање параметара. Уколико корисник потврди измену, званично стање параметара постаје оно које је корисник изменио (поље `form`). Уколико корисник одустане од измене поље `form` се поставља на последњу сачувану верзију (поље `savedForm`). Приликом ажурирања параметара врши се дубоко копирање, јер форме могу садржати угнежене објекте. Имплементацију дубоког копирања приказује Листинг 5.1.2. Поред основног чувања стања форме, неопходно је ажурирати одговарајући оптимизациони алгоритам или критеријум оптималности. Због тога ће у конкретним имплементацијама модалног дијалога прослеђена функција за потврду измена увек прво позивати `updateForm`, а онда, позивом одговарајућег конструктора, ажурирати дати алгоритам или критеријум оптималности.

```

const deepCopy = (obj) =>
  typeof obj !== "object"
    ? obj
    : Object.entries(obj).reduce(
      (copy, [key, value]) => {
        copy[key] =
          typeof value === "object" &&
          value !== null
            ? Array.isArray(value)
              ? value.map(deepCopy)
              : deepCopy(value)
            : value;
        return copy;
      }, {});

```

Листинг 5.1.2 Имплементација дубоког копирања рекурзијом

Компонента `ModalDialog.vue` има један двосмерни параметар - Булову променљиву `isActive` која носи информацију о томе да ли је модални дијалог активан (приказан) или није. Неопходно је да ово буде двосмерни параметар како би се активација модалног дијалога могла вршити из родитељске компоненте, а деактивација из саме компоненте модалног дијалога. Двосмерни параметри се из компоненте детета ажурирају позивом специјалне функције `$emit` која прима два параметра – назив посебне врсте догађаја по шаблону `update:<naziv_parametra>` и нову вредност параметра. Листинг 5.1.3 представља методу за затварање модалног дијалога у којој се ажурира двосмерни параметар из компоненте детета.

```

closeModal() {
  this.$emit("update:isActive", false);
},

```

Листинг 5.1.3 Ажурирање двосмерног параметра из компоненте детета

## 5.1.2 Шаблон компоненте

Шаблон компоненте (енгл. *template*) представља HTML код који ће бити приказан на месту где је компонента искоришћена. Шаблон модалног дијалога састоји се из три дела: заглавља (енгл. *header*), централног садржаја и подножја (енгл. *footer*). Заглавље служи за приказ наслова модалног дијалога и дугмета за затварање дијалога. Подножје садржи дугме за снимање и дугме за отказивање измена параметара у дијалогу.

Централни садржај, кога чини форма за измену параметара, посебан је за сваки оптимизациони алгоритам односно критеријум оптималности. У ситуацијама када део шаблона компоненте у многоме варира, односно када зависи од контекста њене родитељске компоненте, згодно је искористити концепт слотова (енгл. *slots*). Слотови омогућавају прослеђивање HTML кода из родитељске компоненте у компоненту детета. Како је сваки засебан модални дијалог само омотач (енгл. *wrapper*) за базни модални дијалог, засебни дијалог ће при својој дефиницији базном дијалогу проследити произвољан HTML код за форму помоћу које се мењају њему специфични параметри. Специјални таг `<slot></slot>` представља место у шаблону базног модалног дијалога на које се умеће HTML код прослеђен од стране родитеља. Листинг 5.1.4 приказује шаблон базног модалног дијалога.

```
// CSS and button event listeners omitted for brevity

<div :class="{ 'modal-active': isActive }">
  <div>
    <header>
      <p> {{ modalTitle }} </p>
      <button>x</button>
    </header>
    <section>
      <slot></slot>
    </section>
    <footer>
      <button> Save changes </button>
      <button> Cancel </button>
    </footer>
  </div>
</div>
```

Листинг 5.1.4 Шаблон модалног дијалога (CSS стилови и функције везане за дугмад изостављене због сажетости)

## 5.2 Компонента `OptimizerPicker.vue`

Компонента `OptimizerPicker.vue` омогућава преглед и одабир алгоритама оптимизације, као и активацију модалних дијалога помоћу којих је могуће изменити хиперпараметре оптимизационих алгоритама. Измене над алгоритмима у овој компоненти одражавају се на компоненту `ContourPlot.vue` која служи за њихову визуализацију.

### 5.2.1 Параметри компоненте

Ова компонента има само један двосмерни параметар под називом `activeOptimizers`. Овај параметар представља низ објеката који у систему моделују оптимизациони алгоритам. Уопштени објекат који моделује оптимизациони алгоритам тренутно није од важности и биће приказан у поглављу 5.2.3. Битно је нагласити да је `activeOptimizers` иницијално празан низ и да је сама компонента у потпуности одговорна за његово ажурирање.

У специјалној методи `created`, једној од метода животног циклуса компоненте (енгл. *component lifecycle hooks*), која се позива након креирања компоненте, дефинише се интерно поље `optimizersWithModals`. Ово поље обједињује оптимизационе алгоритме и компоненте које представљају одговарајуће модалне дијалоге и пружа додатне информације о њиховој активности. На овај начин олакшано је дефинисање шаблона компоненте помоћу специјалне Vue.js директиве `v-for`, о чему ће више речи бити у поглављу 5.2.2. Листинг 5.2.1 приказује иницијализацију овог поља.

```
// js imports omitted for brevity

const initOptimizers = [
  sgd(0.05, 100),
  momentum(0.05, 0.5, 100),
  nesterov(0.05, 0.5, 100),
  adagrad(0.8, 100),
  rmsprop(0.1, 0.9, 100),
  adadelta(0.9, 100),
  adam(0.091, 0.9, 0.99, 100),
];

const modals = [
  shallowRef(SgdModal),
  shallowRef(MomentumModal),
  shallowRef(NesterovModal),
  shallowRef(AdagradModal),
  shallowRef(RmspropModal),
  shallowRef(AdadeltaModal),
  shallowRef(AdamModal),
];
```

```

created() {
  this.optimizersWithModals = initOptimizers.map(
    (optimizer, idx) => ({
      optimizer: {
        active: true,
        value: optimizer,
      },
      modal: {
        active: false,
        component: modals[idx],
      },
    })),
  );
}

```

Листинг 5.2.1 Иницијализација поља `optimizersWithModals` (учитавање Јаваскрипт фајлова изостављено због сажетости)

Двосмерни параметар ажурира се користећи концепт Vue.js радног оквира под називом „посматрач“ (енгл. *watcher*). Посматрача је могуће дефинисати за поље компоненте и тада он мотри промене које се над тим пољем дешавају. За посматрача је неопходно везати методу (енгл. *handler*), која ће се извршити сваки пут када се конкретно поље промени. Овој методи се прослеђује нова и стара вредност поља за које је везан посматрач, или само нова вредност, тако да су ове вредности доступне за манипулацију. Посматрачу се могу специфицирати и додатни атрибути попут `immediate` и `deep`. Булов атрибут `immediate` служи за контролу извршавања методе посматрача одмах након његовог креирања. Булов атрибут `deep` даје до знања посматрачу да прати измене дубоко угнеждених објеката посматраног поља.

У компоненти `OptimizerPicker.vue` посматрач је дефинисан за поље `optimizersWithModals`. Ово поље је везано за шаблон компоненте и корисник га мења кроз интеракцију са компонентом, активирајући и деактивирајући оптимизационе алгоритме, или мењајући хиперпараметре алгоритама. При свакој промени овог поља, извршава се функција посматрача која ажурира стање двосмерног параметра `activeOptimizers`, што приказује Листинг 5.2.2 . Атрибут `immediate` има вредност `true`, како би се при иницијализацији компоненте, одмах ажурирала вредност параметра `activeOptimizers` и попунио празан низ. Атрибут `deep` има

вредност `true`, како би посматрач мотрио на корисникову измену угнеждених поља објеката у низу `optimizersWithModals`.

```
watch: {
  optimizersWithModals: {
    handler(newValue) {
      const newActiveOptimizers = newValue
        .filter((optMod) => optMod.optimizer.active)
        .map((optMod) => optMod.optimizer.value);

      this.$emit(
        "update:activeOptimizers",
        newActiveOptimizers
      );
    },
    deep: true,
    immediate: true,
  },
}
```

Листинг 5.2.2 Посматрач поља `optimizersWithModals`

## 5.2.2 Шаблон компоненте

Шаблон ове компоненте уводи два нова концепта из Vue.js радног оквира, а то су:

- директива `v-for` и
- динамичка компонента

Директива представља специјалан део шаблона, дефинисан од стране Vue.js радног оквира. Везује се за елемент, попут специјалног атрибута, проширујући могућности обичног HTML кода. Директива `v-for` омогућава исцртавање елемента или дела шаблона више пута на основу Јаваскрипт променљиве која се за њега веже, а над којом се може итерирати (нпр. низ, мапа, скуп, објекат итд.).

```
<element v-for="(value, index) in iterable">
  // use both value and index anywhere inside element
</element>
```

Листинг 5.2.3 Општи облик директиве `v-for`

Општи облик директиве `v-for`, који приказује Листинг 5.2.3, указује на то да је у свакој итерацији доступан наредни елемент и његов индекс.

Динамичка компонента користи се када је неопходно да се компоненте динамички смењују на истом месту у шаблону. Листинг 5.2.4 приказује како се динамичка компонента дефинише у шаблону.

```
<component :is="someImportedComponent"></component>
```

Листинг 5.2.4 Општи облик динамичке компоненте

У случају ове компоненте, коју приказује Листинг 5.2.5, променљива везана за директиву `v-for` је `optimizersWithModals`. То значи да је у свакој итерацији у шаблону доступан наредни елемент овог низа. Поље за потврду (енгл. *checkbox*) везано је за активност оптимизационог алгорита. Кликом на дугме за приказ модалног дијалога одговарајућег алгорита, он прелази у стање активан. Сви дијалози за приказ детаља оптимизационих алгоритама примају двосмерне параметре истог облика (одговарајући оптимизациони алгоритам и активност дијалога) који се из родитељске компоненте везују директивом `v-model`. Ово омогућава коришћење динамичке компоненте, како би сваки засебни дијалог добио одговарајуће двосмерне параметре.

```
<div v-for="(optMod, idx) in optimizersWithModals">
  <div>
    {{ optMod.optimizer.value.id }}
  </div>
  <div>
    <input
      type="checkbox"
      v-model="optMod.optimizer.active"
    />
    <button
      @click="optMod.modal.active = true"
    ></button>
    <!-- Modal -->
    <component
      :key="optMod.optimizer.value.id"
      :is="optMod.modal.component"
      v-model:optimizer="optMod.optimizer.value"
      v-model:active="optMod.modal.active"
    ></component>
  </div>
</div>
```

Листинг 5.2.5 Окосница шаблона компоненте `OptimizerPicker.vue`

### 5.2.3 Имплементација оптимизационих алгоритама

У овом поглављу биће представљен општи интерфејс `optimizer` који поштују сви објекти који моделују оптимизационе алгоритме.

За дефинисање оптимизационих алгоритама искоришћен је концепт из функционалног програмирања под називом *currying* [15]. Овај концепт, у својој основној форми, представља трансформацију функције са више параметара у ланац функција које примају по један параметар. Позивом прве у ланцу функције прослеђује се само први параметар оригиналне функције. Повратна вредност овог позива је наредна функција у ланцу, која прима следећи параметар оригиналне функције. Овакво уланчавање функција врши се док има параметара оригиналне функције. Моћ овог концепта јесте да омогућава да се један позив функције раздвоји на више позива који могу бити извршени у различитим деловима система, где су доступни различити параметри оригиналне функције.

Користећи овај концепт, дефинисан је општи облик оптимизационог алгоритма који приказује Листинг 5.2.6.

```
const algorithm = (...hyperparams, iterations)
=> (gradF) => {
  return function* (x0) {
    let nextPoint = x0;
    // define helper variables
    for (let i = 0; i < iterations; i++) {
      yield nextPoint;
      // update nextPoint according to algorithm
    }
  };
};
```

Листинг 5.2.6 Општи облик оптимизационог алгоритма

Сваки оптимизациони алгоритам прима прво њему специфичне хиперпараметре и број итерација које треба да изврши. Као повратну вредност враћа функцију која прима градијент критеријума оптималности `gradF`. Ова функција затим враћа генератор који прима почетну тачку `x0`. Генератор, који се у Јаваскрипту дефинише посебном генераторском функцијом која започиње са `function*`, враћа једну по



једну (енгл. *yield*) срачунату вредност, без прављења непотребних низова у којима се чува срачуната путања алгоритма, а који заузимају пуно меморије.

Овакав приступ за дефинисање алгоритама, из три корака (позива функције), узет је из разлога што су параметри у различитим позивима доступни у различитим деловима система. Хиперпараметри су доступни у компоненти `OptimizerPicker.vue`, градијент критеријума оптималности у `FunctionPicker.vue`, док се почетна тачка добија у `ContourPlot.vue`. Ови параметри се такође мењају различитом учесталости, где се почетна тачка много чешће мења него остали параметри. Ово представља само један, функционалан начин за дефинисање алгоритама који не допушта мутирање стања (енгл. *immutable state*).

Интерфејс `optimizer` приказује Листинг 5.2.7, где `factory` представља алгоритам који задовољава претходно наведени општи облик алгоритма.

```
const optimizer = {id, title, factory};
```

Листинг 5.2.7 Интерфејс `optimizer`

Имплементација једног објекта који задовољава интерфејс `optimizer` приказана је на примеру методе градијентног спуста са моментом описаног у поглављу 2.1.3 (Листинг 5.2.8).

```
const momentumSGD = (alpha, omega, iterations) =>
(gradF) => {
  return function* (x0) {
    let nextPoint = x0;
    let [vX, vY] = [0, 0];
    let x, y, gradX, gradY;
    for (let i = 0; i < iterations; i++) {
      yield nextPoint;
      [x, y] = nextPoint;
      [gradX, gradY] = gradF(x, y);
      vX = omega * vX - alpha * gradX;
      vY = omega * vY - alpha * gradY;
      nextPoint = [x + vX, y + vY];
    }
  };
};
```

```
const momentum = (alpha, omega, iterations) => ({
  id: "momentum",
  title: "Momentum Gradient Descent",
  factory: momentumSGD(alpha, omega, iterations),
});
```

Листинг 5.2.8 Имплементација методе најбржег спуста са моментом

## 5.3 Компонента FunctionPicker.vue

Компонента `FunctionPicker.vue` омогућава преглед и одабир критеријума оптималности, као и активацију модалних дијалога помоћу којих је могуће изменити параметре критеријума. Измене над критеријумима у овој компоненти одражавају се на компоненту `ContourPlot.vue` која служи за њихову визуализацију. У оквиру ове компоненте приказан је и тродимензионални график критеријума оптималности, како би корисник што боље појмио критеријум.

### 5.3.1 Параметри компоненте

Ова компонента има један двосмерни параметар под називом `plot`. Овај параметар задовољава истоимени интерфејс, који моделује податке неопходне за исцртавање графика, а чији општи облик приказује Листинг 5.3.1.

```
plot: {
  xRange: [xMin, xMax],
  yRange: [yMin, yMax],
  zRange: [zMin, zMax],
  optimizationCriterion,
}
```

Листинг 5.3.1 Интерфејс `plot`

Интерфејс `plot` чине атрибути `xRange`, `yRange` и `zRange` који моделују границе домена  $x$ ,  $y$  и  $z$ , односно минималну и максималну вредност сваке од ових променљивих. Границе домена  $x$  и  $y$  задаје корисник, док ће на основу њих и одговарајућег критеријума оптималности бити израчунате границе  $z$  домена, неопходне за правилно исцртавање контурног дијаграма.

Поред ових атрибута присутан је и критеријум оптималности. Критеријум оптималности је у систему моделован помоћу интерфејса `optimizationCriterion`, чији општи облик приказује Листинг 5.3.2.

```
optimizationCriterion: {
  id,
  title,
  f: <function>
  gradF: <gradientFunction>,
}
```

Листинг 5.3.2 Интерфејс `optimizationCriterion`

Интерфејс `optimizationCriterion` чине поља:

- `id` – јединствени идентификатор
- `title` – назив критеријума оптималности
- `f` – функција која прима координате  $(x, y)$  и враћа вредност критеријума оптималности у датој тачки
- `gradF` – функција која прима координате  $(x, y)$  и враћа вредност градијента критеријума оптималности у датој тачки

Уместо да се градијент рачуна аналитички за сваку функцију, врши се његова апроксимација, што је нарочито згодно при дефинисању нових критеријума. Листинг 5.3.3 приказује функцију за генерисање функција које врше апроксимацију градијента, дефинисане у фајлу `optimizer-criteria.js`

```
const getGradientApproximator = (f) => {
  const h = 1e-7;
  return (x, y) => {
    const gradX = (f(x + h, y) - f(x, y)) / h;
    const gradY = (f(x, y + h) - f(x, y)) / h;
    return [gradX, gradY];
  };
};
```

Листинг 5.3.3 Factory функција за креирање апроксиматора градијента

У истом фајлу дефинисани су критеријуми оптималности доступни при подизању система. Сваки конструктор критеријума прима њему специфичне параметре и враћа објекат који задовољава интерфејс

optimizationCriterion. Пример дефинисања критеријума оптималности приказује Листинг 5.3.4.

```
// math function factory
const quadraticFactory = (a, b, c) => (x, y) =>
  a * x ** 2 + b * x * y + c * y ** 2;

// optimizationCriterion factory
const quadratic = (a, b, c) => {
  const quad = quadraticFactory(a, b, c);
  return {
    id: "quad",
    title: "Quadratic",
    f: quad,
    gradF: getGradientApproximator(quad),
  };
};
```

Листинг 5.3.4 Пример креирања квадратног критеријума оптималности

Након објашњења шта чини двосмерни параметар `plot`, битно је нагласити како је он иницијално празан и како је ова компонента у потпуности одговорна за његово ажурирање. Све промене се преко родитељске компоненте пресликавају на компоненту `ContourPlot.vue`.

### 5.3.2 Компонента `SurfacePlot.vue`

Пре описа шаблон компоненте `FunctionPicker.vue`, ово поглавље ће приказати његов битан део, компоненту `SurfacePlot.vue`. Улога ове компоненте је да прикаже тродимензионални график одабраног критеријума оптималности.

Параметри ове компоненте представљају појединачна поља атрибута `plot` из родитељске компоненте `FunctionPicker.vue`. Поља `xRange`, `yRange` и `optimizerCriterion` су једносмерни параметри, док `zRange` представља двосмерни параметар за чије израчунавање је одговорна компонента `SurfacePlot.vue`.

За исцртавање тродимензионалног графика критеријума оптималности коришћена је библиотека *Plotly.js* описана у поглављу 3.2. Неопходно је

прво дефинисати елемент шаблона у оквиру којег ће график бити исцртан., што приказује Листинг 5.3.5.

```
<template>
  <div class="surface-container" ref="plotly"></div>
</template>
```

Листинг 5.3.5 Цео шаблон компоненте SurfacePlot.vue

Дефинисањем специјалног атрибута `ref="plotly"`, омогућено је лако референцирање овог елемента у самом Јаваскрипт коду компоненте, преко `this.$refs.plotly`. Приликом монтирања компоненте (енгл. *mounted lifecycle hook*), дефинишу се подаци неопходни за исцртавање графика. Затим се на основу података ажурира вредност двосмерног параметра `zRange`. Након тога се позивом методе `plotly.newPlot()` врши исцртавање графика.

```
mounted() {
  const { x, y, z } = this.surfaceData;
  const zRange = d3.extent(z.flat());
  this.$emit("update:zRange", zRange);
  // other paramse omitted for brevity
  const data = [{ x, y, z, type: "surface", ... },,];
  // GUI related options omitted for brevity
  const layout = {...};
  plotly.newPlot(this.$refs.plotly, data, layout);
}
```

Листинг 5.3.6 Иницијално цртање графика при монтирању компоненте

Атрибут `surfaceData` представља специјалан тип „срачунатог атрибута“ (енгл. *computed property*), који дефинише Vue.js радни оквир. Одлика овог атрибута јесте да се аутоматски ажурира сваки пут када дође до промене података од којих зависи. У конкретном случају, атрибут `surfaceData`, ће се поново израчунати сваки пут када се промени било који параметар позване методе `get3Dspace`, што приказује Листинг 5.3.7. Метода `get3Dspace` од параметара прима границе  $x$  и  $u$  домена, математичку функцију критеријума оптималности и корак интерполације (утиче на прецизност приказаног графика). Повратна вредност ове методе су  $x$ ,  $y$  и  $z$  у координате у формату који прописује библиотека *Plotly.js*

```

computed: {
  surfaceData() {
    return this.get3Dspace(
      this.xRange,
      this.yRange,
      this.optimizationCriterion.f,
      0.05
    );
  },
}

```

Листинг 5.3.7 Срачунати атрибут surfaceData

Како би се приликом измене поља родитељског атрибута `plot`, од којих зависи компонента `SurfacePlot.vue`, аутоматски поново исцртавао тродимензионални график, дефинисан је посматрач на срачунати атрибут `surfaceData` (који зависи од променљиве `plot`). При измени овог атрибута ажурира се `zRange` двосмерни параметар и поново исцртава график позивом функције `plotly.restyle()`. Ова функција прима нове  $x$ ,  $y$  и  $z$  координате и ажурира изглед графика без потпуног поновног исцртавања (као што је случај са методом `plotly.newPlot()`). Листинг 5.3.8 приказује описани посматрач.

Метода `d3.extent` представља методу из библиотеке *D3.js* која за прослеђену листу бројева проналази минималну и максималну вредност. Како  $z$  представља листу листа (формат прописан *Ploty.js* библиотеком), да би `d3.extent` вратила праве вредности, неопходно је прво спљоштити  $z$  (енгл. *flatten*).

```

watch: {
  surfaceData: {
    handler({ x, y, z }) {
      const zRange = d3.extent(z.flat());
      this.$emit("update:zRange", zRange);
      plotly.restyle(
        this.$refs.plotly,
        { x: [x], y: [y], z: [z] },
        0
      );
    },
  },
},

```

Листинг 5.3.8 Посматрач који ажурира тродимензионални график

### 5.3.3 Шаблон компоненте

Шаблон компоненте `FunctionPicker.vue` у многоме подсећа на шаблон `OptimizerPicker.vue`. При дефинисању изгледа корисничког интерфејса за избор критеријума оптималности, поново је искоришћен принцип спајања, овог пута критеријума оптималности и одговарајућих дијалога, у јединствени атрибут назван `plotsWithModals`. Овај атрибут, типа низ, везује се за `v-for` директиву и омогућава динамичко везивање компоненти одговарајућих модалних дијалога. Разлика у односу на компоненту `FunctionPicker.vue` јесте та да само један критеријум оптималности може бити одабран. Због тога се уместо поља за потврду користи група радио дугмади (енгл. *radio buttons group*), која пружа могућност избора искључиво једног критеријума оптималности.

Новина у шаблону ове компоненте, поред компоненте `SurfacePlot.vue` описане у претходном поглављу, представља и компонента за пагинацију `SimplePagination.vue`. Ова компонента служи за приказ критеријума оптималности по страницама. Компонента за пагинацију функционише тако што јој се проследи листа елемената за које треба да изврши пагинацију и она ће, на основу странице коју је корисник одабрао, вратити подскуп елемената низа који одговара тој страници, узимајући у обзир величину странице. Корисник бира страницу кликом на једно од два дугмета, дугме за приказ претходне странице и дугме за приказ следеће странице. Уколико приказ претходне или следеће странице није могућ (кориснику приказана прва, односно последња страница), биће онемогућено кликнути на одговарајуће дугме.

Поред компоненте за пагинацију ту је и дугме за додавање новог критеријума оптималности. Кликком на ово дугме отвара се модални дијалог за дефинисање нове функције описан у наредном поглављу.

Листинг 5.3.9 приказује шаблон компоненте `FunctionPicker.vue`.

```

<surface-plot
  :xRange="plot.xRange"
  :yRange="plot.yRange"
  :optimizationCriterion= "plot.optimizationCriterion"
  v-model:zRange="plot.zRange"
/>

<div v-for="plotMod in displayedPlotMods">
  <div class="function-title">
    {{ plotMod.plot.optimizationCriterion.title }}
  </div>
  <div class="function-activate-more">
    <input
      v-model="activeId"
      type="radio"
      :value=
        "plotMod.plot.optimizationCriterion.id"
    />
    <button
      class="function-more-button"
      @click="plotMod.modal.active = true"
    ></button>
  </div>
</div>

<!-- Modals -->
<component
  v-for="plotMod in plotsWithModals"
  :is="plotMod.modal.component"
  v-model:plot="plotMod.plot"
  v-model:active="plotMod.modal.active"
></component>

<simple-pagination
  :pageSize="pageSize"
  :items="plotsWithModals"
  v-model="displayedPlotMods"
></simple-pagination>

<button @click="addCustomFunction">
  Add function
</button>

```

Листинг 5.3.9 Шаблон компоненте FunctionPicker.vue (CSS стилови изостављени због сажетости)



### 5.3.4 Модални дијалози критеријума оптималности

Сви модални дијалози критеријума оптималности примају исте двосмерне параметре – објекат који поштује интерфејс `plot` и Булов параметар `isActive` и деле се у две групе:

- Модални дијалози предефинисаних критеријума оптималности
- Модални дијалог кориснички дефинисаних критеријума оптималности

Модални дијалози предефинисаних критеријума оптималности у многоме су слични дијалозима за приказ информација оптимизационих алгоритама. Поред форме за измену параметара критеријума оптималности, ове дијалоге краси и приказ математичке формуле датог критеријума. `MathJax` библиотека, описана у поглављу 3.5, олакшава приказивање формула тако што интерпретира популарну `TeX` [16] синтаксу и на основу ње црта одговарајуће математичке симболе. Део шаблона компоненте `Quadratic2GaussiansModal.vue` који исцртава математичку формулу приказује Листинг 5.3.10.

```
<h3 ">
  {{ " `$$f(x, y) =
      0.5x^2 + 0.5y^2
      - a e^{-\frac{(x - x_{1})^2}{c} + (y - y_{1})^2}{c}}
      - b e^{-\frac{(x - x_{2})^2}{d} + (y - y_{2})^2}{d}}$$` "
  }}
</h3>
```

Листинг 5.3.10 Употреба `TeX` синтаксе за дефинисање математичке формуле

Формулу исцртану на основу овакве синтаксе приказује Слика 5.1.

$$f(x, y) = 0.5x^2 + 0.5y^2 - ae^{-\frac{(x-x_1)^2+(y-y_1)^2}{c}} - be^{-\frac{(x-x_2)^2+(y-y_2)^2}{d}}$$

Слика 5.1 Математичка формула исцртана помоћу `MathJax` библиотеке

Модални дијалог кориснички дефинисаних критеријума оптималности, представљен компонентом `CustomFunctionModal.vue`, треба да реши проблем прихватања кориснички дефинисане функције и њеног

претварања у Јаваскрипт извршиву функцију. Овај специфичан задатак решен је помоћу библиотеке `math expression evaluator`, описане у поглављу 3.4. Како би ова библиотека могла да интерпретира математичке изразе дефинисане помоћу променљивих  $x$  и  $y$ , неопходно је у глобални објекат `mexp` додати токене ових променљивих. То је урађено одмах при креирању ове компоненте, позивом методе `mexp.addToken` којој је прослеђена листа од два објекта који представљају токене променљивих, што сигнализира број 3 као вредност атрибута `type`.

```
created() {
  mexp.addToken([
    { type: 3, token: "x", show: "x", value: "x" },
    { type: 3, token: "y", show: "y", value: "y" },
  ]);
  this.emitPlot();
}
```

Листинг 5.3.11 Додавање токена променљивих и ажурирање двосмерног параметра `plot`

Листинг 5.3.11 приказује дефиницију токена променљивих, након које следи позив методе `emitPlot`. Ова функција позива се и при креирању компоненте (како би двосмеран параметар `plot` одмах био иницијализован), али и након сваког чувања измена параметара критеријума оптималности. Методу `emitPlot` приказује Листинг 5.3.12

```
emitPlot() {
  const f = compileFunction(this.savedForm.function);
  const gradF = getGradientApproximator(f);

  this.$emit("update:plot", {
    xRange: this.savedForm.xRange,
    yRange: this.savedForm.yRange,
    optimizationCriterion: {
      id: this.plot.optimizationCriterion.id,
      title: this.savedForm.title,
      f,
      gradF,
    },
  });
}
```

```
const compileFunction = (functionString) => {
  const lexed = mexp.lex(functionString);
  const postfix = lexed.toPostfix();
  const f = (x, y) => postfix.postfixEval({ x, y });
  return f;
};
```

Листинг 5.3.12 Приказ emitPlot функције и функције која компајлира математички израз

Функција `compileFunction` прихвата кориснички дефинисан, валидан, математички израз у облику низа карактера. Помоћу `mexp` објекта тај низ карактера претвара у Јаваскрипт извршиву функцију. Ова функција представља валидну математичку функцију `f` у интерфејсу `optimizationCriterion`.

## 5.4 Компонента ContourPlot.vue

Компоненте `OptimizerPicker.vue` и `FunctionPicker.vue` описане у претходна два поглавља пружају податке који су неопходни за рад централне компоненте система `ContourPlot.vue`, која ће бити описана у овом поглављу.

### 5.4.1 Параметри компоненте

Компонента `ContourPlot.vue` прима два једносмерна параметра:

- `plot` – одабрани критеријум оптималности, добављен од стране `FunctionPicker.vue`
- `optimizers` – одабрани оптимизациони алгоритми којима су већ прослеђени одговарајући параметри (`OptimizerPicker.vue`) и градијент критеријума оптималности (атрибут параметра `plot`), тако да им једино недостаје почетна тачка како би израчунали путању оптимизације (поглавље 5.2.3).

### 5.4.2 Шаблон компоненте

Шаблон компоненте представља искључиво елемент у оквиру којег ће помоћу библиотеке *D3.js*, описане у поглављу 3.3, бити исцртан контурни дијаграм. Приказује шаблон ове компоненте Листинг 5.4.1

```
<template> <div :id="svgContainerId"></div> </template>
```

Листинг 5.4.1 Шаблон компоненте `ContourPlot.vue`

### 5.4.3 Цртање контурног дијаграма и алгоритама

Сви детаљи исцртавања контурног дијаграма и алгоритама оптимизације имплементирани су у модулу `d3-contour.js`. Овај модул пружа на употребу три функције које користи компонента `ContourPlot.vue`, а то су:

- `render` – иницијално креирање SVG елемента и исцртавање контурног дијаграма
- `rerender` – ажурирање SVG елемента контурним дијаграмом новог критеријума оптималности
- `updateOptimizers` – ажурирање листе одабраних алгоритама

При монтирању ове компоненте позива се функција `render`, како би се SVG елемент адекватно иницијализовао. Дефинисани су посматрачи за оба једносмерна параметра. При свакој промени једносмерног параметра `plot` позива се функција `rerender`, како би се исцртао нови ажурирани критеријум оптималности. Свака измена једносмерног параметра `optimizers` врши позив функције `updateOptimizers`, којом се ажурирају одабрани алгоритми у модулу. Листинг 5.4.2 приказује употребу `d3-contour.js` модула у овој компоненти.

```
mounted() {
  render({
    svgContainerId: this.svgContainerId,
    ...this.plot,
    optimizers: this.optimizers,
  });
},
watch: {
  plot() {
    rerender(this.plot);
  },
  optimizers() {
    updateOptimizers(this.optimizers);
  },
},
```

Листинг 5.4.2 Употреба модула `d3-contour.js` у компоненти `ContourPlot.vue`

У наставку овог поглавља биће представљени одабрани детаљи модула `d3-contour.js` од интереса за визуализацију алгоритама.

Основне функције овог модула, о којима је већ било речи, приказује Листинг 5.4.3

```
let svg;
let optimizers;
let xScale, yScale;

const render = (options) => {
  updateOptimizers(options.optimizers);
  svg = createSvg(options);
  renderSvg(svg, options);
};

const rerender = (options) => {
  svg.selectAll("*").remove();
  renderSvg(svg, options);
};

const updateOptimizers = (newOptimizers) => {
  optimizers = newOptimizers;
};
```

Листинг 5.4.3 Основне функције модула d3-contour.js

Даље ће се анализирати функција `renderSvg` која помоћу библиотеке *D3.js* исцртава контурни дијаграм и алгоритме. Листинг 5.4.4 приказује имплементацију ове функције.

```
const renderSvg = (options) => {
  setupScales(options);
  const contoursGroup = addContours(options);
  addAxes();
  const gradientPathGroup = svg.append("g");
  addMouseEvent(contoursGroup, gradientPathGroup);
};
```

Листинг 5.4.4 Функција `renderSVG`

Исцртавање контурног дијаграма састоји се из неколико корака. Први корак представља иницијализацију скала, које се користе кроз цео модул. За дефинисање  $x$  и  $y$  скала користи се функција модула *D3.js* `scaleLinear`, која прима два параметра - домен (енгл. *domain*) и распон (енгл. *range*) и дефинише линеарну скалу мапирања домена на распон. Помоћу ове функције олакшано је мапирање картезијанских координата (координата математичког простора, у којем је описан

критеријум оптималности) на координате простора пиксела самог SVG елемента. Дефинисање скала приказује Листинг 5.1.1, где глобалне променљиве `width` и `height` описују ширину и висину SVG елемента.

```
const setupScales = ({ xRange, yRange }) => {
  xScale = scaleLinear(xRange, [0, width]);
  yScale = scaleLinear(yRange, [height, 0]);
};
```

Листинг 5.4.5 Дефинисање скала  $x$  и  $y$  у променљивих

Функција `addContours` исцртава контурни дијаграм и као повратну вредност враћа специјалан *D3* омотач око основног Јаваскрипт објекта који представља елемент групе (таг `<g></g>`) по SVG спецификацији.

Функција `addAxes` исцртава осе независних променљивих  $x$  и  $y$  са свих страна контурног дијаграма.

Имплементација функција `addContours` и `addAxes` ослања се на чланак творца *D3.js* библиотеке Мајка Бостока о цртању контурних дијаграма [17]. Обзиром да ова имплементација није од важности за схватање генералне идеје имплементације визуализатора, биће изостављена.

Функција `addMouseClickedEvent` прима за параметре променљиву `contoursGroup` у којој се чува претходно дефинисан SVG елемент групе који садржи контурни дијаграм, као и нови SVG елемент групе који ће садржати путање оптимизационих алгоритама, а који се чува у променљивој `gradientPathGroup`.

```
const addMouseClickedEvent =
(contoursGroup, gradientPathGroup) => {
  contoursGroup.on("click", (event) => {
    const startPoint = pointer(event);
    // convert pixel to cartesian coordinates
    const x0 = [
      xScale.invert(startPoint[0]),
      yScale.invert(startPoint[1])
    ];
    startOptimization(
      gradientPathGroup, x0, optimizers);
  });
};
```

Листинг 5.4.6 Функција `addMouseClickedEvent`

Листинг 5.4.6 приказује како се у функцији `addMouseEvent` за контурни дијаграм везује функција која ће бити извршена сваки пут када корисник кликне на њега. Ова функција помоћу *D3.js* функције `pointer` преузима координате клика у пикселима. Како имплементирани алгоритми раде са картезијанским координатама, врши се неопходно мапирање (овог пута инверзно мапирање распона у домен), користећи претходно дефинисане скале. Када су координате преведене у добар формат, позивом функције `startOptimization` систем започиње се процес оптимизације и исцртавања путања.

```
const startOptimization =
  (gradientPathGroup, x0, optimizers) => {
    const paths = optimize(x0, optimizers);
    redrawAllPaths(gradientPathGroup, paths);
  };
```

Листинг 5.4.7 Функција `startOptimization`

У функцији `startOptimization`, коју приказује Листинг 5.4.7, прво се врши позив функције `optimize` која сваком од одабраних алгоритама прослеђује почетну тачку у картезијанским координатама. Како генератори оптимизационих алгоритама рачунају путању у картезијанским координатама, дату путању треба превести у пикселе. Генератор `cartesianToPixelPathTransformer` ради управо то.

```
const optimize = (x0, optimizers) =>
  optimizers.map(({ id, generatorFactory }) => {
    const cartesianPath = generatorFactory(x0);
    const pixelPath =
      cartesianToPixelPathTransformer(cartesianPath);

    // convert generator to array
    return { id, path: [...pixelPath] };
  });

function*
  cartesianToPixelPathTransformer(cartesianPath) {
    for (const point of cartesianPath) {
      yield [xScale(point[0]), yScale(point[1])];
    }
  }
```

Листинг 5.4.8 Израчунавање путања оптимизационих алгоритама у пикселима

Повратна вредност функције `optimize`, коју приказује Листинг 5.4.8, представља низ објеката које чине два атрибута:

- `id` – јединствени идентификатор оптимизационог алгорита који ће бити искоришћен за одабир адекватне боје кружића путање
- `path` – низ тачака (дужине једнаке броју итерација) са координатама у пикселима, где свака тачка представља низ од  $x$  и  $y$  координате.

Када су израчунате вредности путања одабраних алгоритама, у функцији `startOptimization` позива се `redrawAllPaths` функција која поново исцртава срачунате путање.

```
const redrawAllPaths = (gradientPathGroup, paths) => {
  removeAllDrawnPaths(gradientPathGroup);
  drawAllPaths(gradientPathGroup, paths);
};

const removeAllDrawnPaths = (gradientPathGroup) => {
  gradientPathGroup.selectAll("path").remove();
  gradientPathGroup.selectAll("circle").remove();
};

const drawAllPaths = (gradientPathGroup, paths) => {
  paths.forEach((path) =>
    drawSinglePath(path, gradientPathGroup)
  );
};
```

Листинг 5.4.9 Исцртавање путања оптимизационих алгоритама

Функција `redrawAllPaths`, коју приказује Листинг 5.4.9, прво брише све путање алгоритама (дефинисаних помоћу SVG елемената *path* и *circle*), а затим започиње појединачно исцртавање сваке од поново срачунатих путања. Функција `drawSinglePath`, исцртава прослеђену путању, тако што анимира исцртавање црне линије (SVG елемент *path*) и кружића (више SVG *circle* елемената) у боји која је на нивоу система дефинисана за дати оптимизациони алгоритам.



## 5.5 Интеракција главних компоненти

Сада, када је приказано како основне компоненте функционишу, следи приказ коренске странице `HomeView.vue`. Ова компонента не прима параметре, а њен шаблон приказује како основне компоненте међусобно интерагују.

```
<aside class="optimizer-picker-container">
  <optimizer-picker
    v-model:activeOptimizers="optimizers"
  />
</aside>
<div class="contour-plot-container">
  <contour-plot
    v-if="plot.zRange && plot.zRange.length"
    v-bind="{
      plot,
      optimizers: optimizersWithGradF
    }"
  />
</div>
<aside class="function-picker-container">
  <function-picker v-model:plot="plot" />
</aside>
```

Листинг 5.5.1 Шаблон `HomeView.vue` компоненте

Као што је помињано у претходним поглављима, компонента `FunctionPicker.vue` ажурира одабрани критеријум оптималности, док `OptimizerPicker.vue` ажурира активне оптимизационе алгоритме. Битан детаљ је да се компоненти `ContourPlot.vue` не прослеђује директно атрибут `optimizers`, већ срачунати атрибут `optimizersWithGradF`. Ово је учињено са намером да се градијент над којим раде алгоритми аутоматски ажурира када се промени одабрани критеријум оптималност, што приказује Листинг 5.5.2

```
optimizersWithGradF() {
  const gradF = this.plot.optimizationCriterion.gradF;
  return this.optimizers.map((opt) => ({
    id: opt.id,
    generatorFactory: opt.factory(gradF),
  }));
}
```

Листинг 5.5.2 Срачунати атрибут `optimizersWithGradF`



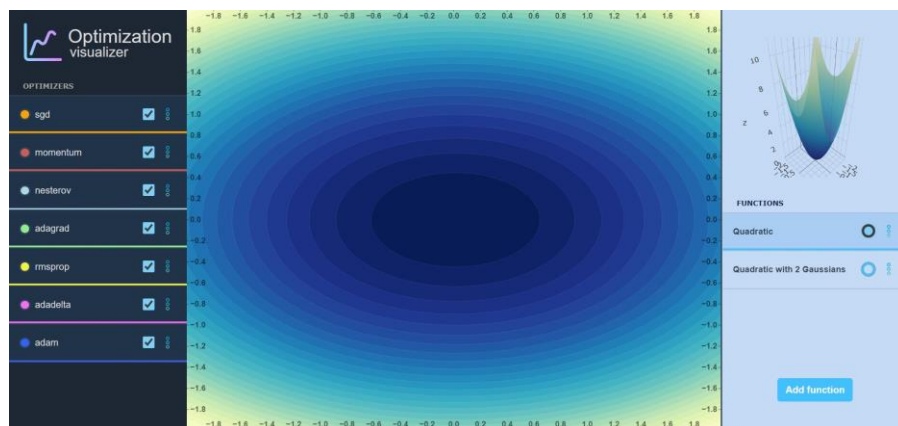
## 6. ДЕМОНСТРАЦИЈА

У овом поглављу биће демонстрирано на који све начин корисник може да интерагује са системом, како би тестирао градијентне оптимизационе алгоритме над различитим критеријумима оптималности. Демонстрација ће бити извршена кроз два сценарија:

- Сценарио 1: Тестирање градијентних оптимизационих алгоритама над предефинисаним критеријумом оптималности, уз измену хиперпараметара алгоритама и параметара критеријума оптималности
- Сценарио 2: Тестирање градијентних оптимизационих алгоритама над кориснички дефинисаним критеријумом оптималности

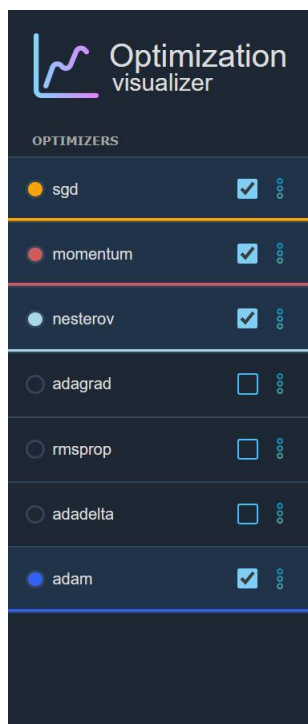
### 6.1 Сценарио 1

Кориснику је приказана почетна страница апликације (Слика 6.1)



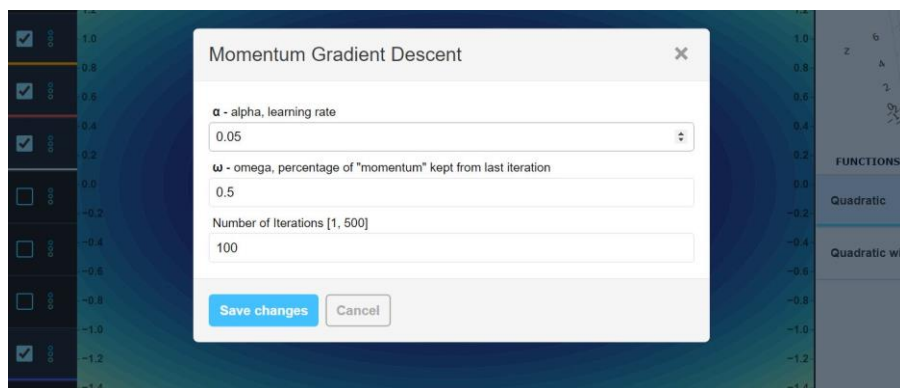
Слика 6.1 Иницијални изглед апликације

Корисник прегледа избор имплементираних градијентних оптимизационих алгоритама и кликом на поље за потврду бира оне који жели да визуализује. Изглед компоненте `OptimizerPicker.vue` након избора корисника приказује Слика 6.2 . Активност алгоритама додатно је наглашена бојом која је јединствена за сваки алгоритам. Боја кориснику такође сугерише која путања (са истобојним кружићима) је везана за који алгоритам.



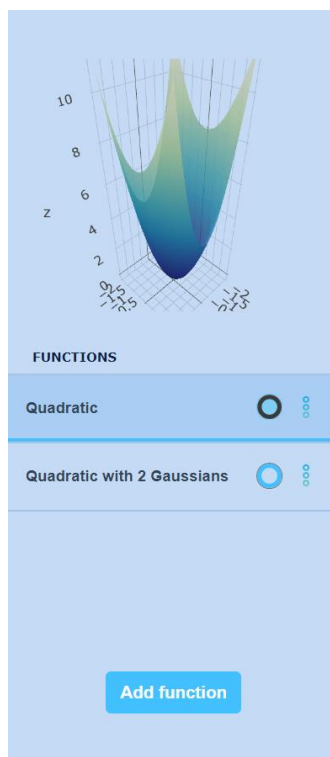
Слика 6.2 Приказ оптимизационих алгоритама одабраних од стране корисника

Корисник кликом на дугме за више информација („три тачке“) код momentum алгорита отвара модални дијалог који приказује његове хиперпараметре (Слика 6.3). Након измене параметра  $\omega$ , корисник потврђује измене и приказује му се обавештење о прихваћеној измени.



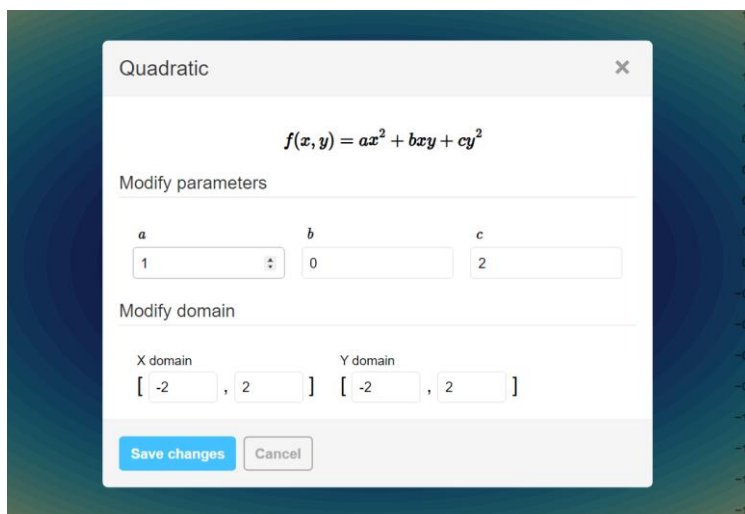
Слика 6.3 Модални дијалог за измену хиперпараметара алгорита

Корисник потом прегледа предефинисане критеријуме оптималности и кликом на одговарајуће радио дугме врши избор (Слика 6.4).



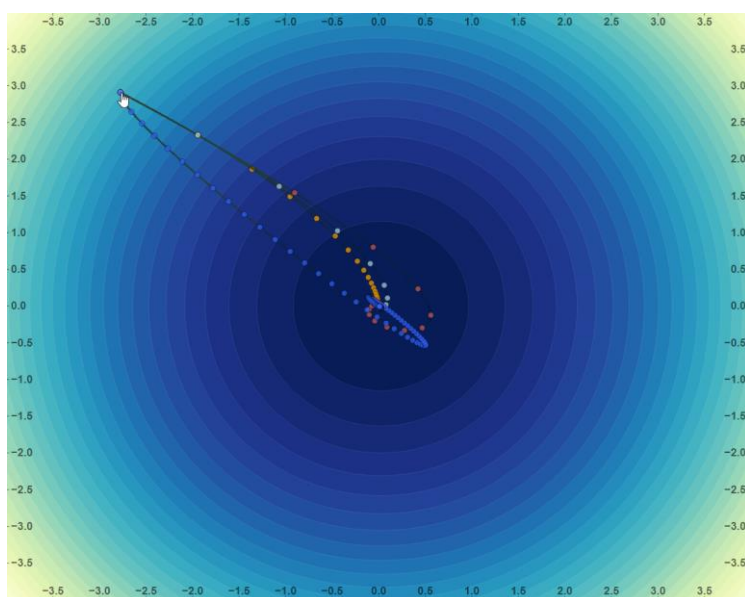
Слика 6.4 Приказ критеријума оптималности у компоненти FunctionPicker.vue

Корисник кликом на дугме за више информација код квадратне функције отвара дијалог који приказује њене параметре. Измене над параметрима и доменом функције корисник потврђује кликом на дугме за снимање измена (Слика 6.5). По затварању модалног дијалога моментално се ажурирају и контурни дијаграм и тродимензионални површински дијаграм, како би сваки од њих осликавао најновије измене. Поред овога, кориснику се приказује обавештење да је систем прихватио унете измене.



Слика 6.5 Модални дијалог за измену параметара критеријума оптималности

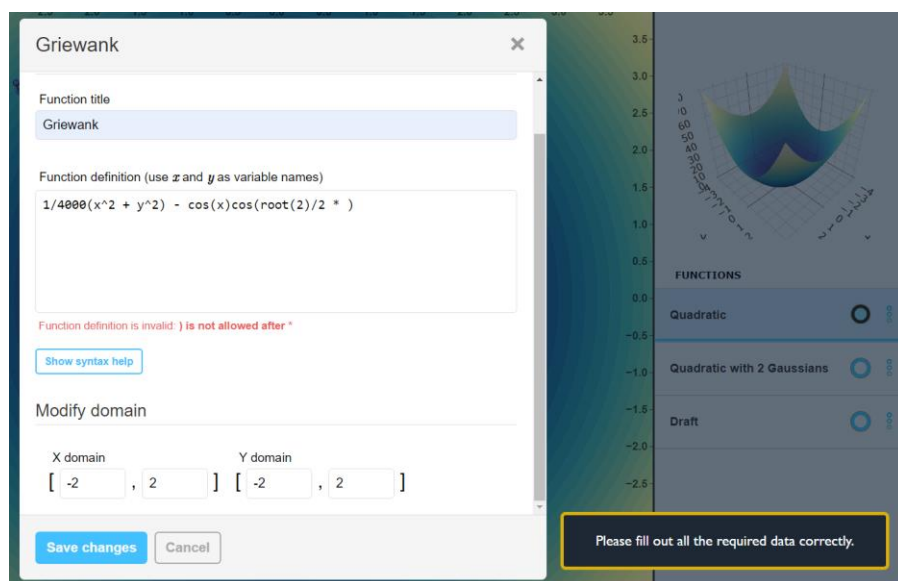
Када је корисник задовољан свим изменама, спреман је да тестира алгоритме. Кликом на контурни дијаграм, бира се почетна тачка за оптимизацију (Слика 6.6), а затим се исцртавају оптимизационе путање одабраних алгоритама.



Слика 6.6 Визуализација алгоритама на ажурираном критеријуму оптималности

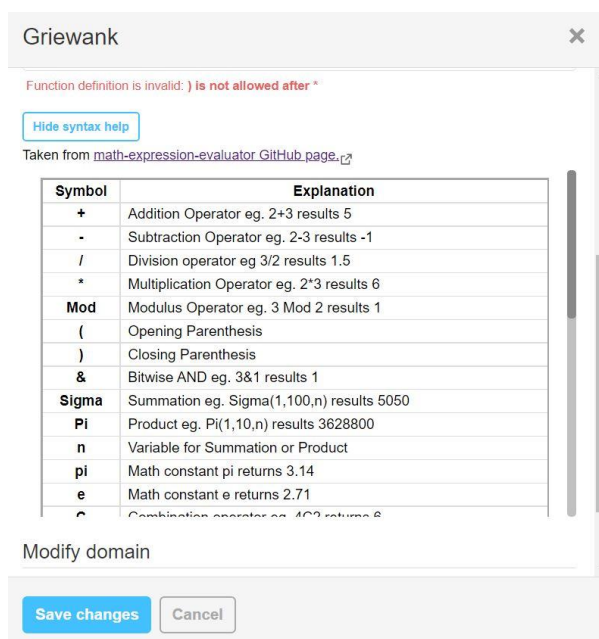
## 6.2 Сценарио 2

Како би корисник што мање био ограничен, пружена је могућност дефинисања новог критеријума оптималности. Кликом на дугме за додавање нове функције (Слика 6.4), отвара се модални дијалог који пружа форму за унос података нове функције. Током попуњавања форме валидира се унети математички израз и исписује грешка уколико израз није исправан. Уколико корисник покуша да сачува измене са невалидним математичким изразом или празним насловом функције, биће обавештен како то није могуће учинити (Слика 6.7).



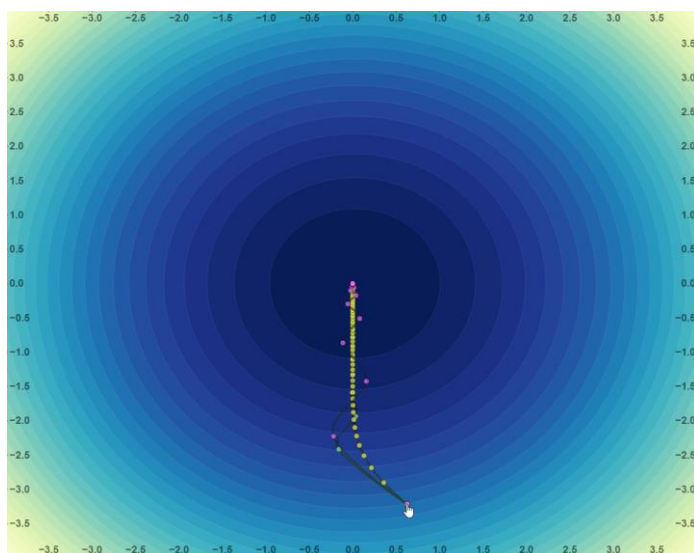
Слика 6.7 Покушај чувања измена кориснички дефинисаног критеријума оптималности са невалидним математичким изразом

Уколико корисник није сигуран који су све математички симболи и функције доступни и на који начин се записују приликом дефинисања математичке функције, омогућено му је да кликом на дугме прикаже, односно сакрије помоћ за синтаксу (Слика 6.8). Након задавања валидног математичког изрази, корисник потврђује измене и систем га обавештава о успешном дефинисању новог критеријума оптималности.



Слика 6.8 Приказ синтаксне помоћи при дефиницији математичког израза

Корисник потом бира нови критеријум оптималности и над њим тестира алгоритме који нису били изабрани у првом сценарију (Слика 6.9).



Слика 6.9 Тестирање алгоритама над кориснички дефинисаним критеријумом



## 7. ЗАКЉУЧАК

У овом раду описан је један приступ визуализације градијентних оптимизационих алгоритама. Како се ови алгоритми користе за обучавање неуронских мрежа у машинском учењу, стицање основне интуиције како ови алгоритми раде у многоне олакшава избор алгорита који може ефикасније и боље да реши конкретан проблем.

Кориснику је пружена могућност тестирања неких од најпопуларнијих градијентних оптимизационих алгоритама над предефинисаним или кориснички дефинисаним тродимензионалним критеријумима оптималности. Измена хиперпараметара алгоритама омогућена је са циљем тестирања њиховог утицаја на понашање алгоритама.

Могућност избора алгоритама за визуализацију и критеријума оптималности над којим ће се они визуализовати, издваја овај систем од њему сличних. Дефинисање нових критеријума оптималности, као и подешавање хиперпараметара алгоритама, у потпуности предаје контролу над апликацијом у корисникове руке и развија жељу за експериментисањем.

Даљи развој описаног система ишао би у два правца:

- повећање едукативног аспекта система
- повећање употребљивости система

Повећање едукативног аспекта система огледало би се у пружању више информација о самим алгоритмима, конкретно како сваки од њих функционише и на који начин хиперпараметри утичу на њихово понашање.

Повећање употребљивости система огледало би се првенствено у стварању упутства за коришћење у форми модалног дијалога који би кориснику представио све функционалности система. Ово упутство приказало би се одмах при активирању система, и било доступно за консултацију током употребе система. Поред тога, усавршавање анимације исцртавања путање алгоритама знатно би улепшало корисничко искуство у извесним случајевима коришћења система.



## ЛИТЕРАТУРА

- [1] Gradient <https://en.wikipedia.org/wiki/Gradient>
- [2] Gradient descent [https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent)
- [3] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4:1–17, 1964.
- [4] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In International conference on machine learning, pages 1139–1147, 2013.
- [5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12:2121–2159, 2011.
- [6] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. arXiv preprint arXiv:1212.5701, 2012.
- [7] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, pages 1–13, 2015.
- [8] Clément Mihailescu - Pathfinding visualizer <https://clementmihailescu.github.io/Pathfinding-Visualizer>
- [9] Emilien Dupont - Interactive Visualization of Optimization Algorithms in Deep Learning <https://emiliendupont.github.io/2018/01/24/optimization-visualization>
- [10] Vue.js framework <https://vuejs.org/>
- [11] Plotly.js <https://plotly.com/javascript/>
- [12] D3.js, data driven documents <https://d3js.org/>
- [13] Math expression evaluator <http://bugwheels94.github.io/math-expression-evaluator>
- [14] MathJax <https://www.mathjax.org>
- [15] Currying <https://en.wikipedia.org/wiki/Currying>
- [16] TeX <https://en.wikipedia.org/wiki/TeX>
- [17] Mike Bostock, Contours in D3.js <https://observablehq.com/@d3/contours>



## **БИОГРАФИЈА**

Жарко Благојевић је рођен 18.02.2000. у Сремској Митровици. Одрастао је у Руми, где је стекао своје основно и средње образовање. Школске 2018/19 године уписује Факултет техничких наука у Новом Саду на студијски програм Рачунарство и аутоматика. Положио је све испите предвиђене планом и програмом и стекао услов за одбрану завршног рада.