# Sučelje za pristup funkcijama strukture protona

**Kontrec, Andrej**

**Master's thesis / Diplomski rad**

**2021**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:217:623168

*Rights / Prava:* In copyright / Zaštićeno autorskim pravom.

*Download date / Datum preuzimanja:* **2024-03-29**

*Repository / Repozitorij:*

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Andrej Kontrec

# Sučelje za pristup funkcijama strukture protona

Diplomski rad

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER: NASTAVNIČKI

**Andrej Kontrec**

Diplomski rad

# Sučelje za pristup funkcijama strukture protona

Voditelj diplomskog rada: Prof. dr. sc., Krešimir Kumerički

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2021.

# Sažetak

Cilj rada je izraditi i prezentirati aplikaciju koja će služiti kao sučelje za pristup funkcijama strukture protona. Prije nego razradimo taj problem moramo uvesti osnovne koncepte nuklearne i fizike čestica , te objasniti što točno mislimo pod izrazom struktura protona. Kako bi pronašli odgovor na to pitanje proći ćemo kroz razne fizikalne eksperimente koji se koriste u istraživanju strukture protona. Doći ćemo do modela strukture koje ćemo koristiti u samoj aplikaciji. Nakon toga ćemo proći kroz razne faze razvoja aplikacije u Pythonu pomoću Dash aplikacijskog okvira, te objasniti kako je nastala finalna verzija aplikacije. Proći ćemo i kroz proces implementacije aplikacije na Heroku web servis kako bi joj se moglo pristupiti kroz internet preglednike sa bilo kojeg računala. Nakon toga ćemo proći kroz finalnu verziju aplikacije te pogledati par primjera raznih konfiguracija kako bi pokazali mogućnosti aplikacije. Za metodički dio rada ćemo se fokusirati na potencijalno korištenje Dash aplikacijskog okvira u nastavi pomoću kojega bi učenici mogli jednostavno ,uz vođenje nastavnika, napraviti vlastitu web aplikaciju. Time bi omogućili učenicima da steknu iskustvo s radom na projektu kroz koji će steći korisna iskustva za potencijalni posao u IT industriji.



Ključne riječi: proton, struktura protona, Python, web aplikacija, Dash, Plotly

# Interface for access to proton structure functions


## Abstract

The goal of this paper is to create and present an application which will serve as an interface for accessing proton structure functions. Before we dive deeper into that problem, we first need to introduce the basic concepts of nuclear and particle physics and explain what we mean by proton structure functions. To find the answer to that question we will need to explore different physical experiments which are used in proton structure research. We will arrive at the physical model we will be implementing in our application. After that we will run through the stages of development of our Python application, created within the Dash application framework. We will also discuss the process of deployment on the Heroku web service. After that we will show some examples from our final web application version to showcase some of its capabilities. For the methodology part of this paper, we will focus on the potential use of the Dash framework in class with which our students could simply, with the guidance of their teacher, create their own web application. This would enable them to gather valuable experience in working on projects which will be very useful for potential future employment in the IT industry.



Keywords: proton, proton structure functions, Python, web application, Dash, Plotly

# Sadržaj

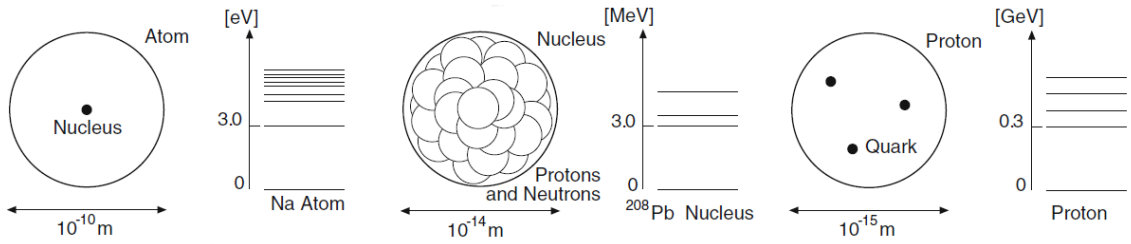

1 Uvod ... 2
1.1 Osnovna građa svijeta ... 2
1.2 Fundamentalne interakcije ... 3
1.3 Otkriće prvih čestica ... 4
1.4 Otkriće anti-čestica ... 7
1.5 Akceleratori ... 8
1.6 Detektori ... 9
1.7 Standardni model ... 11
1.8 Web aplikacije ... 12
2 Struktura protona ... 14
2.1 Elastično raspršenje ... 14
2.2 Duboko neelastično raspršenje ... 18
2.3 Partonski model ... 21
2.4 Duboko virtualno komptonsko raspršenje ... 24
2.5 Modeli komptonskih funkcija strukture ... 26
3 Aplikacija ... 28
3.1 Plotly i Dash ... 28
3.2 Alpha ... 29
3.3 Beta ... 33
3.4 Prva puna implementacija ... 36
3.5 Finalna verzija aplikacije ... 38
4 Metodički dio ... 44
4.1 Projekti u nastavi ... 44
4.2 Kostur aplikacije ... 45
4.3 Projekt za jednog učenika ... 47
4.4 Projekt za parove učenika ... 48
4.5 Projekt za cijeli razred ili grupu učenika ... 49
4.6 Priprema za sat ... 50
4.6.1 Uvodni dio ... 51
4.6.2 Središnji dio ... 51
4.6.3 Završni dio ... 53

5 Zaključak ..... 54
Dodaci .....
A Kod aplikacije .....
B Kostur aplikacije .....
C Prijmer zadatka .....
Literatura .....

# 1. UVOD

## *1.1 Osnovna građa svijeta*

Ljudi su kroz svoju povijest tražili odgovor na pitanje od čega je izgrađen svijet. Otprilike 400 godina prije Krista Grčki filozofi Demokrit i Leukup predložili su da se tvar sastoji od nedjeljivih čestica koje su nazvali atomi, riječ koja je proizašla iz a (ne) i tomos (rezati ili dijeliti) [4]. Ova ideja je bila zaboravljena do 1804. g. kada je Engleski znanstvenik Dalton, koji se smatra ocem moderne kemije, otkrio je mnoge kemijske fenomene koji bi se mogli objasniti ako uzmemo da su atomi svakog elementa osnovne nedjeljive sastavnice tvari. Do kraja 19. stoljeća bilo je poznato da se sva tvar sastoji od atoma. Također postojanje skoro 100 elemenata sa svojstvima koja se periodno ponavljaju bio je jasan znak da ti atomi također imaju neku unutarnju strukturu. Moderni koncept atoma stvoren je početkom 20. stoljeća kao rezultat eksperimenata Rutherford-a i njegovih suradnika. Atom se sastoji od guste jezgre, takozvanog nukleusa, koji je okružen elektronskim oblakom. Nukleus se može još podijeliti na manje čestice. Otkrićem neutrona 1932. g. više nije bilo sumnje da se nukleus sastoji od protona i neutrona koji su se po tome nazvali nukleoni. Elektronu, neutronu i protonu je kasnije nadodana četvrta čestica neutrino, koja je postulirana kako bi se β-raspad složio sa osnovnim zakonima očuvanja energije, impulsa i kutne količine gibanja. Tako su već sredinom tridesetih godina prošlog stoljeća otkrivene ove čestice koje se još i danas smatraju osnovnim dijelovima tvari. Ali one nisu mogle objasniti sve fenomene atomske i nuklearne fizike, tako je daljnje istraživanje u akceleratorima čestica pokazalo da su protoni i neutroni samo dio veće obitelji čestica koje danas zovemo hadroni. Više od 200 hadrona su otkriveni do danas te su se oni poput atoma mogli grupirati preko sličnosti njihovih svojstava. Zbog toga se vjerovalo da se oni također sastoje od nekih osnovnijih čestica [1].




Slika 1.1: Skale veličina u atomskoj hijerarhiji [13]

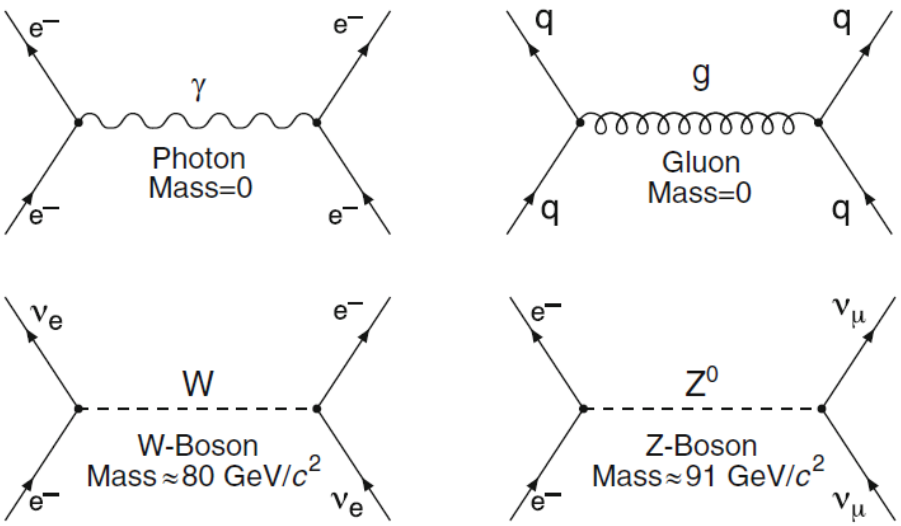
Krajem šezdesetih godina prošlog stoljeća došlo je do stvaranja kvarkovskog modela. U tom modelu svi poznati hadroni se sastoji od kombinacije dva ili tri kvarka. Postoje dva osnovna

tipa čestica od kojih se sastoji svijet: leptoni, koji uključuju elektrone i neutrine, te kvarkovi. Danas poznajemo šest leptona i šest kvarkova, te njihove anti-čestice. Oni se mogu grupirati po takozvanim generacijama ili obiteljima ovisno o njihovim karakteristikama.

### *1.2 Fundamentalne interakcije*

Zajedno sa novim saznanjima o elementarnim česticama došlo je do promjene našeg razumijevanja osnovnih sila prirode kao i osnovnih interakcija između elementarnih čestica. Oko 1800. g. smatralo se da su gravitacijska, električna, magnetska te neka nepoznata sila između atoma i molekula četiri osnovne sile u prirodi. Na kraju 19. stoljeća shvaćeno je da su električna i magnetska sila samo manifestacije jedne sile, takozvane elektromagnetske. Kasnije je pokazano da atomi imaju strukturu i da se sastoje od pozitivno nabijene jezgre i elektronskog omotača, te da se atom drži na okupu pomoću elektromagnetske interakcije. Sveukupno atomi su električno neutralni. Na kratkim udaljenostima električna polja atoma se ne neutraliziraju u potpunosti, tako da susjedni atomi i molekule utječu jedni na druge. Te različite vrste kemijskih sila kao npr. Van der Waalsova sila su posljedice elektromagnetske sile.

Sa razvojem nuklearne fizike dvije nove sile kratkog dometa su pridodane osnovnim silama prirode. To su nuklearna sila, koja djeluje između nukleona, i slaba sila, koja se manifestira tokom β-raspada. Danas znamo da nuklearna sila nije fundamentalna. Analogno tome da su sile između atoma posljedice elektromagnetske sile, nuklearna sila je posljedica jake sile koja veže kvarkove u protone i neutrone. Ove jake i slabe sile vode do odgovarajućih fundamentalnih interakcija između elementarnih čestica. Četiri fundamentalne sile koje su osnova svih fizikalnih fenomena su gravitacija, elektromagnetska sila, jaka sila i slaba sila. Gravitacija je bitna za postojanje zvijezda, galaksija i planetarnih sistema, te za naš svakodnevni život. Ona nema nikakav učinak na nuklearnu fiziku, preslaba je da bi znatno utjecala na interakciju između elementarnih čestica. Prema današnjim saznanjima interakcije se provode preko izmjene bozona tj. čestica koje imaju spin 1. To su fotoni u elektromagnetskim interakcijama, gluoni u jakim interakcijama, te $W^+$, $W^-$ i $Z^0$ bozoni u slabim interakcijama. Sa svakom od ovih interakcija povezana je jedna vrsta naboja, električni naboj, slabi naboj i jaki naboj koji se također naziva boja. Čestica može sudjelovati u interakciji samo ako nosi odgovarajući naboj. Leptoni i kvarkovi nose slabi naboj, kvarkovi i neki leptoni imaju električni naboj, te samo kvarkovi imaju boju.

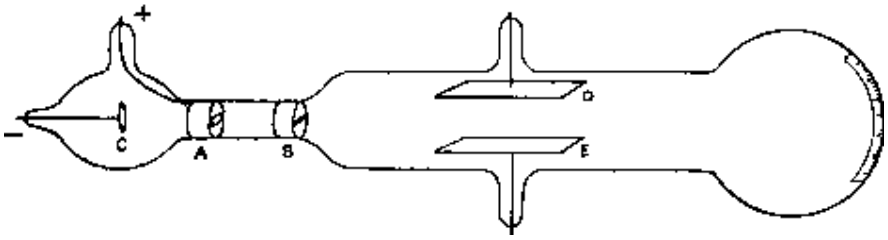



Slika 1.2: Feynmanovi dijagrami interakcija između čestica [14]

W i Z bozoni imaju velike mase, $M_W \approx 80$ GeV/c$^2$ a $M_Z \approx 91$ GeV/c$^2$, te po Heisenbergovom principu neodređenosti mogu se proizvesti samo kao virtualne posredničke čestice u procesu raspršenja na jako kratko vrijeme. Zbog toga slaba sila ima jako kratki domet, dok je masa mirovanja fotona nula, te je zbog toga domet elektromagnetske sile beskonačan. Gluoni poput fotona imaju masu mirovanja nula, ali za razliku od fotona koji nemaju električni naboj, gluoni imaju boju. Posljedica toga je da postoji međusobna interakcija između pojedinih gluona, te im je zbog toga domet jako kratak.

### *1.3 Otkriće prvih čestica*

Prva otkrivena elementarna čestica bila je elektron 1897. g. kada je Thomson uspio proizvesti zrake elektrona kao slobodnih čestica u elektronskim cijevima. Zakretanjem tih čestica pomoću električnih i magnetskih polja mogao je odrediti njihovu brzinu te omjer njihove mase i naboja. Rezultati su bili neovisni od tipu katode ili plina koji je koristio i tako je zaključio da je našao univerzalnu sastavnicu tvari. Kasnije je pomoću takozvane kapljične metode odredio naboj elektrona čime je također fiksirao i njegovu masu.

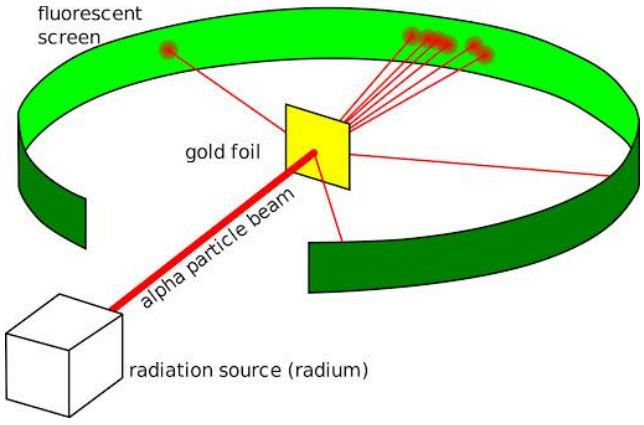



Slika 1.3: Thomsonov eksperiment [15]

Posljedica ovog otkrića je bila pojava raznih modela atoma, jedan od kojih je bio i Thomsonov model. Taj model kaže da su elektroni i jednak broj pozitivno nabijenih čestica podjednako raspoređeni kroz cijeli atom. Tako je atom bio električni neutralan. Rutherford, Geiger i Marsden su uspješno osporili ovaj model. U njihovom, sada poznatom, eksperimentu gledali su raspršenje α-čestica na tankom sloju zlata. Gledajući distribuciju kutova raspršenja α-čestica vidjeli su pojavu velikih kutova raspršenja. To nije konzistentno s ravnomjernom raspodjelom pozitivnog naboja kroz atom. Time je uspostavljeno da atom u sebi ima malu pozitivno nabijenu jezgru velike mase oko koje kruže negativno nabijeni elektroni. Rutherford je do otkrića protona došao tako da je gađao lake jezgre sa α-česticama koje su same po sebi bile jezgre helija. U ovim reakcijama gledao je pretvorbu elemenata, tj. nešto slično inverznu α-raspada. Gađajući jezgre dušika uočio je pozitivno nabijenu česticu sa neobično velikim dometom koja je morala proizaći iz atoma. Time je zaključio da je atom dušika bio uništen i da je dio jezgre bio izbačen kao ta čestica. Sličnu česticu je prije otkrio kada je gađao vodik. Time je zaključio da su te čestice jezgre vodika koje su i same sastav jezgre dušika. Vidio je ovu nuklearnu reakciju

$$^{14}N + {}^{4}He \rightarrow {}^{17}O + p$$

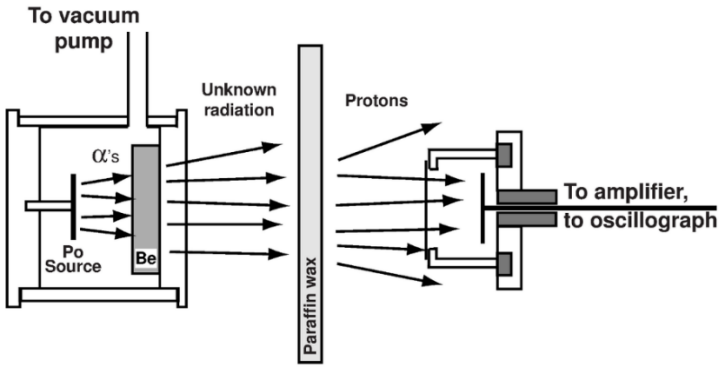
u kojem se jezgra dušika pretvara u jezgru kisika, gubitkom protona. Time možemo smatrati da je jezgra vodika osnovni dio jezgre atoma. Rutherford je također pretpostavio kako bi mogao dobiti daljnji uvid u jezgru atoma kada bi imao mogućnost gađanja tih jezgri sa česticama sve veće energije. Time je postavio temelj moderne nuklearne fizike.




Slika 1.4: Rutherfordov eksperiment [16]

Einstein je objasnio fotoelektrični efekt 1905. g. tako da je pretpostavio da je energija elektromagnetskih valova kvantizirana., to jest da dolazi u malim paketima koje je nazvao fotonima, svaki s energijom $E = h \bullet f$ gdje je f frekvencija, a h Planckova konstanta. Atomi i

jezgre mogu emitirati i apsorbirati fotone. Promatrajući ih kao čestice fotoni nemaju ni naboj ni masu mirovanja. Neutron je također pronađen gađanjem jezgara α-česticama. Rutherfordova metoda detektiranja i brojanja čestica pomoću scintilacije na ekranu od cink sulfida ne funkcionira kod neutralnih čestica. Izrada ionizacijskih i magljenih komora pojednostavile su detekciju nabijenih čestica ali ni to nije pomoglo u pronalasku neutrona. Neutralne čestice se mogu detektirati samo indirektno. Chadwick je 1932. g. našao primjereni eksperimentalni pristup. Koristio je ozračivanje berilija α-česticama koristeći polonij kao izvor, i time je uspostavio neutron kao jednu od osnovnih sastavnica jezgre. Neutralna radijacija bila je opservirana je i ranije u sličnim eksperimentima, ali se nije razumjela njena priroda. Chadwick je postavio pokus tako da se njegova neutralna radijacija se sudarila s vodikom, helijem, i dušikom, te je mjerio odbojne energije tih jezgri u ionizacijskoj komori. Iz zakona sudara dobio je da je masa čestice neutralne radijacije slična masi protona. Chadwick je tu česticu nazvao neutron.

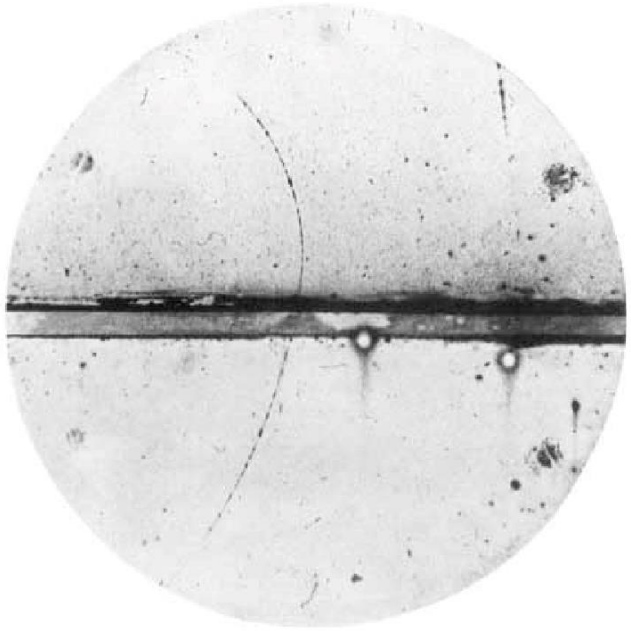



Slika 1.5: Shematski dijagram Chadwickovog eksperimenta [17]

Ovim otkrićima su pronađene glavne sastavnice atoma. Razvoj ionskih izvora i masenih spektrometara dozvolili su istraživanje sila koje vežu sastavnice nukleusa tj. protone i neutrone. Ta sila je očito puno jača od elektromagnetske sile koja te drži atom na okupu, obzirom da se nukleus može razbiti samo gađanjem visoko energetskim česticama. Energija vezanja sustava daje informaciju o vezanju i stabilnosti sustava. Ova energija je razlika mase sustava i sume masa sastavnica sustava. Ispada da je za nukleus ova razlika blizu 1% nuklearne mase. Ovaj fenomen, povijesno nazvan defekt mase, je bio jedan od prvih eksperimentalnih dokaza poznate relacije $E=mc^2$.

### *1.4 Otkriće anti-čestica*

Prva otkrivena anti-čestica bila je pozitron koju je otkrio Američki fizičar Anderson 1932. g. tokom istraživanja čestica koje bombardiraju zemlju iz svemira. Gledajući tragove čestica u plinskoj komori uočio je nešto neobično: česticu koja je prošla kroz tanku olovnu plohu unutar komore. Njen trag je bio više zakrivljen iznad plohe nego ispod, što pokazuje da je iznad plohe njena brzina bila manja nego ispod. Što znači da je čestica putovala prema gore, jer ne bi mogla ubrzati tokom prolaska kroz olovnu plohu. Veličina traga i zakrivljenost je pokazivala da čestica ima jednaku masu i naboj kao i elektron. Ali smjer gibanja i magnetskog polja su pokazivali da bi čestica trebala imati pozitivan naboj. Anderson joj je dao ima pozitron [6].



Slika 1.6: Putanja pozitrona u Andersonovom eksperimentu [18]

To otkriće je razveselilo teoretičare poput Paula Diraca koji je još 1928. g. tijekom svog rada na relativističkoj generalizaciji Schrodingerove jednadžbe postulirao nešto slično pozitronu. Primijetio je da jednadžba za slobodni elektron predviđa ne samo kontinuirano energetsko stanje veće od energije mirovanja nego i kontinuirano stanje negativne energije. To je predstavljalo problem kojeg je Dirac pokušao riješiti svojom interpretacijom da ako pretpostavimo da postoji more negativnih stanja ispunjenih elektronima i da se ti elektroni zbog nekog razloga ne mogu opservirati. Tada bi se praznina u tim energetskim nivoima, slično kao i praznina u valentnoj traci poluvodiča, mogla smatrati pozitivnim nabojem. Ova teorija je imala još jednu posljedicu, a to je činjenica da se slično kao fotoni, elektroni se

također mogu stvoriti i uništiti. Razlika je u tome da se elektroni mogu stvoriti ili uništiti samo u elektron-pozitron paru, kako se nebi narušio zakon očuvanja naboja [4].

Feynman je 1949. g. pokazao da se pozitron može matematički opisati kao elektron koji se giba unazad u vremenu. Njegova preformulacija Diracove teorije maknula je teške račune vezane za more negativnih energetskih stanja te je postavila elektron i proton na istu razinu. Pokazalo se da Diracova teorija naznačuje činjenicu da u fizici čestica svaka čestica ima svoju anti-česticu. Pozitron je anti-čestica elektronu, ali je isto tako elektron anti-čestica pozitronu. Pošto se pozitroni ne pojavljuju inače u tvarima koje proučavamo dogovorom je došlo do konvencije da je elektron čestica a pozitron anti-čestica. Za neke neutralne čestice moguće je da su čestica i anti-čestica iste čestice. Foton je primjer jedne takve čestice. U kasnijim eksperimentima su pronađene anti-čestice za proton i neutron. Anti-proton je pronađen 1955. g. na Berkleyu, dok je anti-neutron pronađen godinu dana kasnije. Pošto je neutron neutralna čestica njegova anti-čestica je također neutralna te se one ne razlikuju po električnom naboju nego po suprotnom barionskom broju.

### *1.5 Akceleratori*

Akceleratori čestica nam daju različite vrste zraka čestica čije energije mogu doseći i nekoliko TeV-a. Te znake nam u jednu ruku služe kao izvor energije kojom ako gađamo jezgre možemo stvoriti različita pobuđena stanja ili čak nove čestice. U drugu ruku mogu nam služiti kao probe kojima možemo istražiti strukturu mete.

Elektrostatski akceleratori se sastoje od generatora visokog napona, terminala i cijevi za izlaz zrake. U ovim akceleratorima se izravno koristi izraz $E = ZeU$, gdje je Z atomski broj, e naboj elektrona, a U napon u akceleratoru. Najčešći primjer takvog akceleratora je Van de Graaff akcelerator, koji može postići energije do 15 MeV. Duplo veće energije je moguće dobiti tandemom Van de Graaff akceleratora.

Linearni akceleratori mogu postići energiju reda veličine GeV-a. To postižu neprestanom akceleracijom čestica. Izrađeni su od puno akceleratorskih cijevi koje su raspoređene u ravnu crtu i čestice prolaze kroz središnju os. Svaki par susjednih cijevi ima suprotno raspoređen potencijal tako da se čestice između njih ubrzavaju. Ovakvi akceleratori ne mogu stvoriti kontinuirane zrake nego proizvode akcelerirane pakete čestica koji su u fazi sa frekvencijom generatora čestica. Najveći svjetski linearni akcelerator SLAC dug je oko 3

km i prolaskom kroz njegovih sto tisuća akceleratorskih cijevi elektroni dobivaju energiju oko 50 GeV-a.

Sinkrotroni imaju cirkularni oblik te za razliku od linearnih akceleratora, gdje čestica samo jednom prođu kroz akceleratorsku strukturu, ovdje čestice više puta prolaze kroz istu akceleratorsku strukturu. Time čestice mogu postići puno veće energije od čestica koje prolaze kroz linearni akcelerator. Čestice se drže u kružnoj putanji pomoću magnetskih polja. Sustavi za akceleraciju se postavljaju samo na neke dijelove unutar kružne putanje. Sinkrotroni također proizvode samo pakete čestica a ne kontinuirane zrake. Akcelerirane čestice u sinkrotronu gube dio svoje energije zbog sinkrotronske radijacije. Sinkronska radijacija je emisija fotona od strane nabijene čestice koja je prisiljena na gibanje kružnom putanjom, zbog čega ima radijalnu akceleraciju. Ova pojava jako ovisi o masi te je gubitak energije za elektron znatno manji u odnosu na protone jednakih energija. Zbog toga je maksimalna energija modernih elektronskih sinkrotona oko 100 GeV-a. Za protone, gdje energija ne ovisi toliko od sinkrotronskoj radijaciji, maksimalna energija je limitirana snagom magnetskih polja koje drže česticu u kružnoj putanji. Pomoću supravodljivih magneta moguće je postići energije reda veličine par TeV-a. Čestice koje akceleriramo pomoću sinkrotrona možemo koristiti na dva načina. Možemo ih zakrenuti iz sinkrotona i usmjeriti prema stacionarnoj meti, ili ih možemo pohraniti u sinkrotronu gdje ih možemo sudariti sa nekom umetnutom metom ili drugim snopom čestica.

Najpoznatiji akcelerator čestica danas na svijetu je LHC koji se nalazi na granici Francuske i Švicarske gdje je izrađen od strane europskog centra za nuklearna istraživanja CERN-a kako bi nam dao odgovore na fundamentalna pitanja fizike. Punog imena veliki hadronski sudarivač i opsegom od oko 27 km najveći je akcelerator na svijetu. Sastoji se od dosta manjih akceleratora koji su izgrađeni kroz povijest ali i danas služe za postepeno ubrzavanje čestica na sve veće energije. Prvi na redu je Linac 2 koji ubrzava protone do energije 50 MeV-a, zatim slijedi PSB gdje se postiže energija od 1.4 GeV-a, zatim dolaze PS i SPS koji podižu energiju na 450 GeV-a. Takvi protoni ulaze u LHC gdje se ubrzavaju do energije od 6.5 TeV-a i sudaraju sa drugim česticama [19].

### *1.6 Detektori*

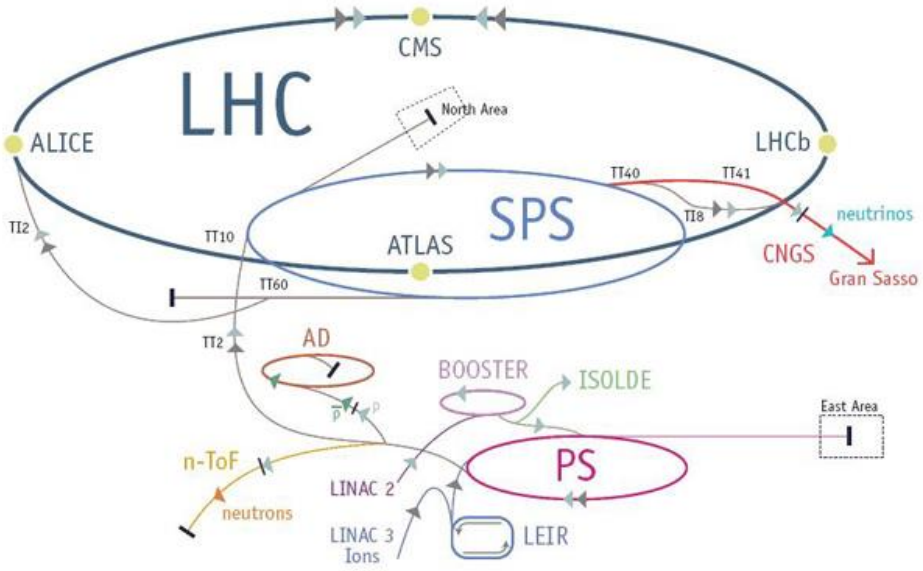
Detekcija čestica se vrši na jedan od dva načina. Nabijene čestice međudjeluju sa plinovima, tekućinama, amorfnim krutim tvarima i kristalima. Te interakcije proizvode

električne ili optičke signale u tim materijalima koje naznačuju prolaz čestica. Neutralne čestice se detektiraju indirektno kroz sekundarne čestice. Fotoni stvaraju slobodne elektrone, pomoću fotoelektričnog ili Komptonskog efekta, ili elektron-pozitron parove preko tvorbe parova. Neutroni i neutrini proizvode nabijene čestice u reakcijama s jezgrom.

Detektori čestica se mogu podijeliti u sljedeće kategorije:

- Scintilatori daju brzu povratnu informaciju ali imaju samo srednje dobru prostornu rezoluciju
- Plinski brojači pokrivajući veliki prostor daju dobru prostornu razlučivost i koriste se u kombinaciji s magnetskim poljima za mjerenje impulsa.
- Poluvodički brojači imaju veoma dobru energetsku i prostornu rezoluciju.
- Čerekovljevi brojači i brojači temeljeni na tranzicijskoj radijaciji se koriste za identifikaciju čestica
- Kalorimetri mjere ukupnu energiju na visokim energijama

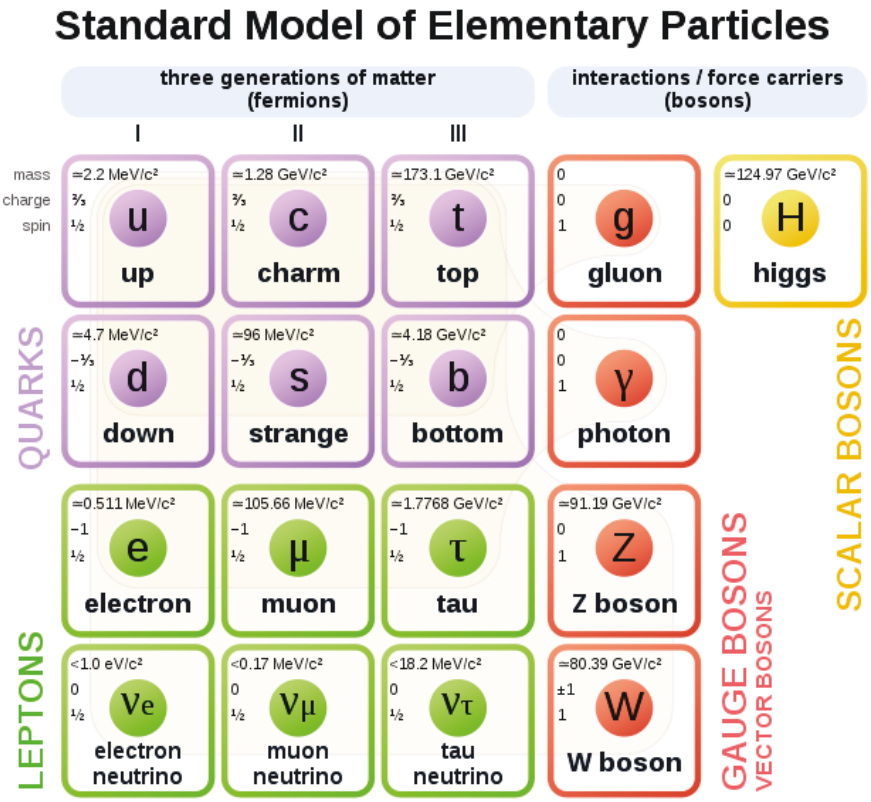
Detektori se slažu u sustave gdje se kombinacije raznih detektora pomno odaberu kako bi optimizirali mjerenje energije i impulsa te identifikaciju proizvoda mjerene reakcije. Takve sustave koristimo u svim većim eksperimentima u fizici čestica. Postoje razni veliki sustavi detektora, kao primjer možemo navesti detektore koji se nalaze na LHC-u a to su ALICE, ATLAS, CMS i LHCb [19].




Slika 1.7: Grafički prikaz LCH kompleksa na CERN-u [19]

### *1.7 Standardni model*

Standardni model u sebi uključuje unificiranu teoriju elektroslabe sile, teorije koja želi ujediniti elektromagnetsku i slabu silu, te kvantnu kromodinamiku, teoriju kvark-gluonske strukture čestica. U standardnom modelu sadržane su tri obitelji čestica: šest leptona koji ne interagiraju s jakom silom, šest kvarkova od koji se stvaraju hadroni, i čestice koje prenose sile. Čestice koje prenose jaku silu su gluoni, fotoni prenose elektromagnetsku silu, a W i Z bozoni prenose slabu silu. Leptoni i kvarkovi su fermioni i svaki od njih ima odgovarajući anti-fermion, te se mogu podijeliti u tri generacije. Prva generacija leptona sadrži elektron i elektronski neutrino, druga mion i mionski neutrino, a treća generacija tauon i tauonski neutrino. Prva generacija kvarkova sadrži up i down kvarkove, druga charm i strange kvarkove, a treća top i bottom kvarkove. Standardni model predviđa da neutrini nemaju masu ali opservacija neutrinskih oscilacija je pokazala da ipak imaju masu. Zbog konzistencije standardnog model morala je postojati čestica spina 0 koja se veže s ostalim česticama proporcionalno njihovim masama. Ta čestica, Higgsov bozon, je pronađena 2012. g. u CERN-u [20].




Slika 1.8: Grafički prikaz standardnog modela [20]

### *1.8 Web aplikacija*

Cilj ovog rada je izraditi web aplikaciju koja će omogućiti posjetiteljima web stranice da prikazuju strukturu protona sa parametrima koje sami odaberu. Web aplikacija, za razliku od standardne desktop ili mobilne aplikacije, je program koji se vrti direktno u web pregledniku. Web aplikacije rade na klijent-server principu te serviraju klijentu grafičko sučelje definirano u aplikaciji. Danas se koriste razne web aplikacije npr. web-mail, internet bankarstvo, internet aukcije, oglasnici i slični servisi. Razlika između dinamičke internet stranice i web aplikacije nije jasno definirana, ali jednostavne web stranice nećemo smatrati web aplikacijama dok ćemo neki servis koji slično radi kao i desktop ili mobilna aplikacija nazvati web aplikacijom. Web aplikacije su obično programirane u nekom jeziku koji podržavaju suvremeni web preglednici. Najčešće su to aplikacije napisane u JavaScript i HTML jezicima koji su kompatibilni s internet preglednicima. Možemo podijeliti web aplikacije u dvije vrste: dinamičke aplikacije koje zahtijevaju obradu koda na serveru, i statičke koje ne zahtijevaju nikakvu obradu koda na serveru.

Web aplikacija radi na serveru, što znači da server upravlja zahtjevima i upitima samog koda aplikacije. Kada klijent zatraži izvođenje nekog dijela koda aplikacija taj upit izvodi na serveru nakon čega korisnik vidi rezultat u svojem internet pregledniku. Osim samih upita web aplikacije često rade s nekom bazom podataka. Baza se nalazi na serveru te se upiti izvode na relaciji aplikacija-baza, a krajnji rezultat se isporučuje klijentu putem web preglednika. Server može biti ASP.NET, ASP, ColdFusion, PHP ili JSP. Sve su to tehnologije koje se danas koriste za najpoznatije web aplikacije.

Web aplikacija radi na sljedeći način:

- Klijent šalje zahtjev web serveru putem svojeg internet preglednika.
- Web server prosljeđuje zahtjev aplikacije na server.
- Aplikacija izvodi zadatak, generira rezultat i šalje ga na web server.
- Web server prosljeđuje rezultat klijentu u njegov internet preglednik.

Web aplikacija može biti obična forma za popunjavanje, koja može generirati neki rezultat ili pohraniti podatke u neku bazu. Kompleksne web aplikacije nude razne mogućnosti npr. obradu teksta, obradu skeniranih dokumenata, pripremu dokumenata za ispis, obradu fotografija u web pregledniku itd. Primjeri kompleksnijih web aplikacija stvorenih od strane tehnoloških divova su Gmail, Microsoft 365, Google Docs, itd.

Prednost web aplikacije je činjenica da ju možemo pokrenut neovisno o OS kojeg koristimo (MS Windows, Linux, macOS, UNIX, Android, iOS). Važno je imati instalirani internet preglednik (Edge, Internet Explorer, Google Chrome, Firefox, Opera) koji podržava izvođenje aplikacije. Također svi korisnici aplikacije u danom trenutku koriste istu verziju, što znači da nema nekompatibilnosti. Održavanje web aplikacije je lakše jer se rutinska održavanja i nadogradnje izvode na samom serveru, što ukida potrebu da se radi intervencija na lokalnim računalima klijenata. Možda glavna prednost je da web aplikaciju korisnik ne mora instalirati na svom računalu, čime bi zauzeo svoju lokalnu memoriju, te može odmah koristiti aplikaciju čim naiđe na nju tokom surfanja interneta.

Nedostatci web aplikacije su činjenica da je za rad potrebna brza i stabilna internetska veza, te da naš server koji sadrži aplikaciju mora biti dobro konfiguriran i dostupan čitavo vrijeme izvođenja aplikacije. Također se može reći da su serveri izloženiji napadima te zbog toga manje sigurni, ali nisu ni lokalne aplikacije u potpunosti zaštićene. Ali ipak mogućnost napada na server je realnija te ovdje sigurnost nije u potpunosti u rukama korisnika.

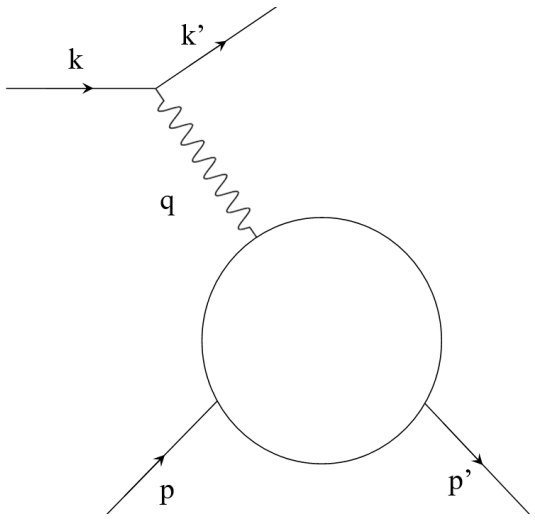
## 2. Struktura protona

Ako želimo napraviti web aplikaciju koja crta 3D strukturu protona prvo moramo odgovoriti na pitanje kakva je to točno struktura protona. I ne samo to nego moramo razumjeti način na koji mi tu strukturu promatramo. Naravno nemoguće ju je direktno vidjeti ali zato postoje fizikalni modeli kojima ju možemo opisati. Ti modeli u kombinaciji sa raznim eksperimentima daju nam sliku strukture protona. Tu sliku onda možemo prikazati u našoj aplikaciji.

Glavni eksperimenti kojima ćemo se baviti su elastično raspršenje, duboko neelastično raspršenje i duboko virtualno komptonsko raspršenje. Promatrajući form faktore koje dobivamo u tim eksperimentima dolazimo do raznih informacija o strukturi protona.

### *2.1 Elastično raspršenje*

U elastičnom raspršenju nalazimo iste čestice prije i poslije sudara. Meta ostaje u osnovnom stanju tako da samo apsorbira odbojni impuls tj. kinetičku energiju (Slika 2.1). Jedina razlika između čestica prije i poslije sudara je iznos njihovih impulsa i energija. Kako bi raspoznali što manje mete, tj. unutarnju strukturu mete trebamo koristiti zrake sve viših energija.




Slika 2.1: Feynmanov dijagram elastičnog raspršenja [21]

Elastično raspršenje elektrona od vodika, daje nam informaciju o protonu. Pojedine suptilnosti se trebaju uzeti u obzir kod takvih eksperimenta, te ne možemo koristiti Mottov udarni presjek koji se inače uzima u laboratorijskim sustavima kod raspršenja ovakvih čestica kada u obzir uzimamo spin dane čestice. Mottov udarni presjek je dan sljedećom formulom.

$$\left(\frac{d\sigma}{d\Omega}\right)^{*}_{\{Mott\}} = \frac{4Z^2\alpha^2(\hbar c)^2E'^2}{|\boldsymbol{q}c|^4}cos^2\left(\frac{\theta}{2}\right)$$

Gdje je α Sommerfeldova konstanta fine strukture tvari, a E' energija odbijene čestice. Radijus protona je otprilike 0.8 fm, sto znaci da za njihovo proučavanje trebamo energije od par stotina MeV-a do nekoliko GeV-a. Ako usporedimo te energije s masom jezgre M≈938MeV/c2 vidimo da su te energije jednakog reda veličine. Zbog toga se vise ne može zanemariti odboj mete. Zbog toga dobijemo dodatni faktor u Mottovom udarnom presjeku.

$$\left(\frac{d\sigma}{d\Omega}\right)_{\{Mott\}} = \left(\frac{d\sigma}{d\Omega}\right)^{*}_{\{Mott\}} \cdot \frac{E'}{E}$$

Pošto gubitak energije elektrona zbog odboja nije zanemariv raspršenje se vise ne može opisati pomoću impulsa u tri dimenzije. Sada nam je potrebna generalizacija klasičnog impulsa u četverovektor impulsa u prostor-vremenu, čiji je kvadrat Lorentzova invarijanta:

$$q^2 = (p - p')^2 = 2m_e^2c^2 - (\frac{EE'}{c^2} - |p||p'|cos\,\theta) \approx \frac{-4EE'}{c^2}sin^2\frac{\theta}{2}$$

Kako bi radili samo sa pozitivnim vrijednostima definiramo $Q^2 = -q^2$. U Mottovom udarnom presjeku $\mathbf{q}^2$ se zamijeni s $q^2$ ili $Q^2$. Nakon toga uzimamo u obzir i interakciju struje elektrona s magnetskim momentom jezgre. Magnetski moment nabijene cestice spina -1/2 koji ne posjeduje nikakvu unutarnju strukturu (Diracova cestica) dan je sa:

$$\mu = g \cdot \frac{e}{2M} \cdot \frac{\hbar}{2}$$

gdje je M masa cestice a faktor g=2 je rezultat relativističke kvantne mehanike (Diracova jednadžba). Magnetska interakcija je povezana sa izvrtanjem spina jezgre. Magnetski član je velik na visokom $Q^2$ i ako je kut raspršenja velik. Ovaj član uzrokuje da se udarni presjek manje smanjuje na velikim kutovima raspršenja i kod izotopne raspodjele nego što bi samo električna interakcija uzrokovala.

Raspodjela naboja i struja može se opisati pomoću dva form faktora, isto kao i u slučaju nukleusa. Za proton, dva form faktora su potrebna kako bi se okarakterizirala magnetska i električna distribucija. Udarni presjek za raspršenje elektrona od protona opisan je Rosenbluthovom formulom:

$$\left(\frac{d\sigma}{d\Omega}\right) = \left(\frac{d\sigma}{d\Omega}\right)_{Mott} \cdot \left[\frac{G_E^2(Q^2) + \tau G_M^2(Q^2)}{1+\tau} + 2\tau G_M^2(Q^2) tan^2\frac{\theta}{2}\right]$$

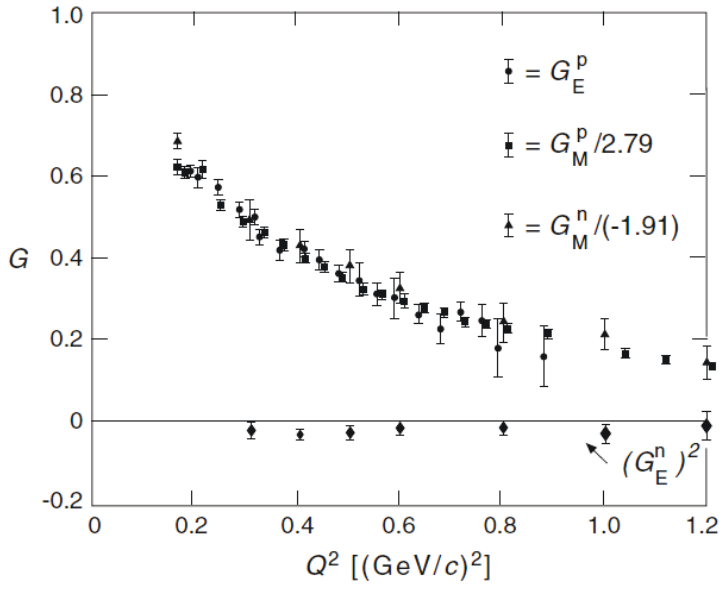
Gdje $G_E(Q^2)$ i $G_M(Q^2)$ su električni i magnetski form faktori, koji oboje ovise o $Q^2$, a $\tau$ je:

$$\tau = \frac{Q^2}{4M^2c^2}$$

Mjerene ovisnosti form faktora o $Q^2$ nam daju informaciju o radijalnoj distribuciji naboja i magnetskih momenata. Granična vrijednost $Q^2 \rightarrow 0$ je izrazito važna. U tom slučaju $G_E$ jednak je električnom naboju mete, normaliziran na elementarni naboj e, a $G_M$ jednak je magnetskom momentu μ mete, normaliziran na nuklearni magneton. Granične vrijednosti su:

$$G_E^P(Q^2 = 0) = 1 \quad G_M^P(Q^2 = 0) = 2.79$$

Kako bi neovisno odredili $G_E(Q^2)$ i $G_M(Q^2)$ udarni presjeci moraju biti mjereni pri fiksiranom $Q^2$, za različite kutove raspršenja θ. Tada se izmjereni udarni presjeci podjele s Mottovim udarnim presjecima. Ako prikažemo rezultate kao funkciju od $\tan^2(\theta/2)$, tada izmjerene točke stvaraju ravnu liniju, kao što predviđa Rosenbluthova formula. $G_M(Q^2)$ je tada određen nagibom linije, a presjek $(G_E^2 + \tau G_M^2)/(1+\tau)$ na θ = 0 tada daje $G_E(Q^2)$. Ako provedemo ovu analizu za različite vrijednosti $Q^2$ možemo onda dobiti ovisnost form faktora o $Q^2$.




Slika 2.2: Ovisnost form faktora protona i neutrona o $Q^2$ [22]

Mjerenja elektromagnetskih form faktora do vrlo visokih $Q^2$ vrijednosti većinom su provedena krajem šezdesetih i početkom sedamdesetih godina prošlog stoljeća na raznim akceleratorima elektrona u Europi i SAD-u [1]. Na slici (Slika 2.2) je graf dobiven rezultatom eksperimenata na Stanford-u gdje su elektrone elastično sudarali sa jezgrama vodika i deuterija. Ispada da protonski električni form faktor i magnetski form faktor od protona i neutrona slično opadaju s povećanjem $Q^2$. Mogu se dobro aproksimirati takozvanim dipolnim form faktorom.

$$G_E^2(Q^2) = \frac{G_M^P(Q^2)}{2{,}79} = \frac{G_M^n(Q^2)}{-1{,}91} = G^{dipole}(Q^2)$$

Gdje je

$$G^{dipole}(Q^2) = \left(1 + \frac{Q^2}{0{,}71(\mathrm{GeV/c})^2}\right)^{-2}$$

Preko ovisnosti form faktora o $Q^2$ možemo dobiti distribuciju naboja i magnetizacije unutar protona. Form faktore možemo interpretirati kao Fourierov transformat raspodjele naboja, te dipolni form faktor odgovara raspodjeli naboja koja opada eksponencijalno:

$$\varrho(r) = \varrho(0)e^{-ar} \text{ gdje je } a = 4{,}26\ \mathrm{fm}^{-1}$$

Vidimo da proton nije točkasta čestica niti je homogeno nabijena sfera nego poprilično složen sustav. Srednji kvadrat radijusa distribucije naboja i magnetizacije u protonu su slične veličine te se mogu dobiti iz naboja odgovarajućih form faktora u točki $Q^2 = 0$. Za dipolni form faktor dobijemo

$$\sqrt{\langle r^2\rangle_{dipol}} = 0{,}81\ \mathrm{fm},$$

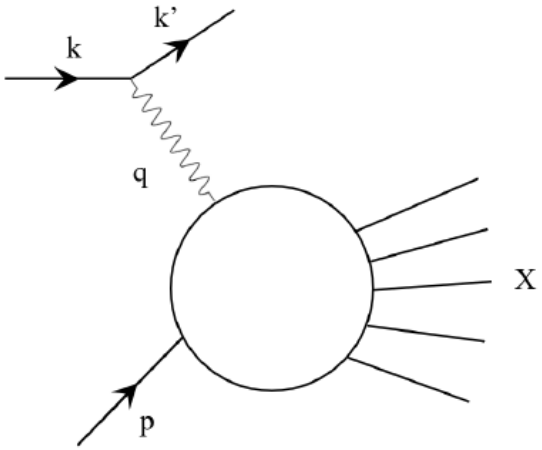
dok kod preciznijih mjerenja form faktora za male vrijednosti $Q^2$ dobijemo malo odstupanje od dipola za proton. Nagib kada $Q^2$ teži u 0 dobiven iz eksperimentalnih podataka nam daje radijus distribucije naboja protona

$$\sqrt{\langle r^2\rangle_p} = 0{,}879\ \mathrm{fm}$$

Ovo je i glavna informacija koju možemo dobiti iz form faktora elastičnog raspršenja: opis distribucije naboja u protonu, te njegovu prosječnu veličinu. Za detaljniji opis strukture protona morati ćemo prodrijeti dublje u proton drugačijom vrstom eksperimenta.

### *2.2 Duboko neelastično raspršenje*

U neelastičnom raspršenju dio kinetičke energije predane meti ju stavlja u pobuđeno stanje. Povratkom mete u osnovno stanje dogodit će se emisija lakih čestica, najčešće fotona ili π-mezona ili će se meta raspasti u više različitih čestica(Slika 2.3).




Slika 2.3: Feynmanov dijagram dubokog neelastično raspršenja [23]

U dubokom neelastičnom raspršenju mi gađamo proton leptonima vrlo visoke energije. Ti leptoni međudjeluju s protonom preko izmjene virtualnih fotona. Kako bi mogli razlučiti sastavnice protona, valna duljina virtualnih fotona mora biti mala u odnosu na radijus protona, i posljedično su potrebne visoko energijske zrake. Prva generacija takvih eksperimenata su izvedene u kasnim šezdesetim i sedamdesetim godinama prošlog stoljeća u SLAC-u koristeći linearni elektronski akcelerator sa maksimalnom energijom od 25 GeV-a. Druga generacija je bila izvedena tokom osamdesetih i devedesetih godina prošlog stoljeća na CERN-u i FNAL-u, koristeći zrake miona umjesto elektrona. Zadnja generacija takvih eksperimenta je bila izvedena između 1992. i 2007. godine na elektron proton sudarivaču HERA lociranom na DESY. Ovdje su elektroni ili pozitroni energija 27.6 GeV-a i protoni energija do 920 GeV-a kružili u suprotnim smjerovima u zasebnim prstenima za pohranu te se sudarili na dvije udarne točke. Rezultirajući $Q^2$ je reda veličine $10^4$ GeV-a [1].

Kako bi bolje razumjeli ovaj eksperiment prvo trebamo uvesti neke fizikalne veličine. Počinjemo sa invarijantnom masom stanja W koju mjerimo ovakvim eksperimentima te računamo pomoću četverovektora impulsa virtualnog fotona i protona.

$$W^2c^2 = (P + q)^2$$

Lorentzovu invarijantu ν definiramo kao

$$\nu = \frac{Pq}{M}$$

Proton miruje u laboratorijskom sustavu što odgovara tome da P = (Mc, 0) i q =((E-E')/c, **q**). Zbog toga je prenesena energija od strane virtualnog fotona u laboratorijskom sustavu jednaka:

$$\nu = E - E'$$

Također je dobro uvesti bezdimenzionalne Lorentzove invarijante

$$y := \frac{Pq}{Pp} = 1 - \frac{E'}{E}$$

i Bjorkenovu varijablu

$$x := \frac{Q^2}{2Pq} = \frac{Q^2}{2M\nu}$$

Bjorkenova varijabla je mjera neelastičnosti procesa. Za elastična raspršenja W je jednak masi nukleona M pa dobivamo:

$$2M\upsilon - Q^2 = 0 \implies x = 1,$$

dok je za neelastične procese W veći od M i dobivamo:

$$2M\upsilon - Q^2 > 0 \implies 0 < x < 1$$

Elektronska raspršenja u rasponu gdje su W, $\sqrt{Q^2}/c$ i ν/c$^2$ puno veći od nukleonske mase M nazivamo duboko neelastična raspršenja. Dinamika takvog procesa je slična dinamici elastičnog raspršenja te se može opisati form faktorima. U ovom slučaju se ti form faktori nazivaju strukturne funkcije $W_1$ i $W_2$ ili $F_1$ i $F_2$. U elastičnom raspršenju zraka dane energije E može imati samo jedan slobodan parametar. U neelastičnom raspršenju dobivamo dodatnu

dimenziju slobode zbog energije uzbuđenja protona zbog čega su strukturne funkcije i udarni presjeci ovdje funkcije dvije neovisne varijable npr. (E', θ), ($Q^2$,ν) ili ($Q^2$, x). Udarni presjek je dan sljedećom formulom.

$$\frac{d^2\sigma}{d\Omega dE'} = \left(\frac{d\sigma}{d\Omega}\right)^*_{Mott} \left[W_2(Q^2,\nu) + W_1(Q^2,\nu)\, tan^2\frac{\theta}{2}\right]$$

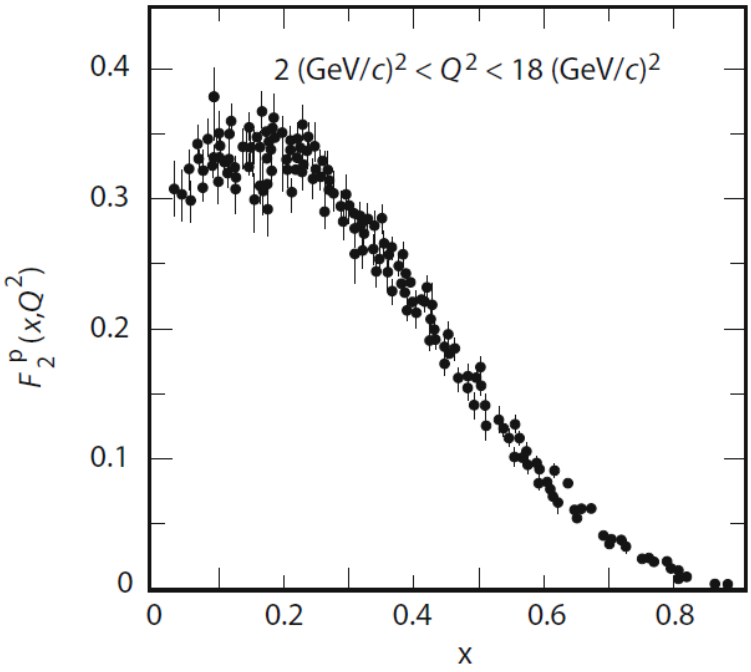
Drugi član ova formule dolazi od magnetske interakcije. Ova notacija udarnog presjeka se uglavnom koristi za povijesne razloge. Umjesto dvije strukturne funkcije $W_1(Q^2, \nu)$ i $W_2(Q^2, \nu)$ koristimo dvije bezdimenzionalne strukturne funkcije

$$F_1(x, Q^2) = Mc^2 W_1(Q^2, \nu),$$

$$F_2(x, Q^2) = \nu W_2(Q^2, \nu),$$

te preko njih izrazimo diferencijalni udarni presjek u ovisnosti od x i $Q^2$:

$$\frac{d^2\sigma}{d\Omega dx} = \frac{4\pi\alpha^2\hbar^2}{Q^4}\left[\left(\frac{1-y}{x} - \frac{My}{2E}\right)F_2(x, Q^2) + y^2 F_1(x, Q^2)\right]$$




Slika 2.4: Ovisnost $F_2$ protona o x pri vrijednosti $Q^2$ između 2 GeV/$c^2$ i 18 GeV/$c^2$ [24]

Mjerenja dubokog neelastičnog raspršenja za fiksirane vrijednosti x i $Q^2$ te za različite vrijednosti y, tj. različitim energijama E, su potrebna da bi se odredile obije

strukturne funkcije $F_1(x, Q^2)$ i $F_2(x, Q^2)$. Eksperimenti na SLAC-u u kasnim šezdesetima su sadržavali iznenađenje. Prikazom strukturne funkcije $F_2(x, Q^2)$ kao funkcije x, za podatke u rasponu vrijednosti $Q^2$ između 2 GeV/$c^2$ i 18 GeV/$c^2$ vidimo da za fiksnu vrijednost x strukturne funkcije slabo ako uopće ovise o $Q^2$ (Slika 2.4). Činjenica da strukturne funkcije ne ovise o $Q^2$ naznačuje da se elektroni raspršuju od točkastog naboja. Uzeći u obzir da protoni nisu točkasti naboji dolazimo do zaključka da: Protoni imaju podstrukturu koja je sastavljena od točkastih čestica [1].

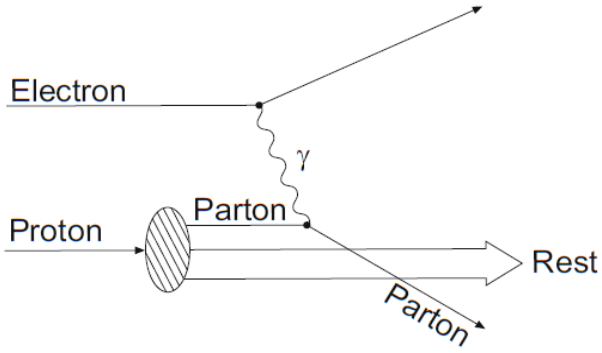
Strukturalna funkcija $F_1$ je rezultat magnetske interakcije te jednostavno nestaje za raspršenja od čestica sa nula spinom. Za Dirac čestice spina ½ vrijedi:

$$2xF_1(x) = F_2(x)$$

Strukturne funkcije dubokog neelastičnog raspršenja nam naznačuju kakva je struktura protona tj. vidimo da u dubokom neelastičnom sudaru virtualni foton udari u neki točkasti naboj koji se nalazi u protonu. Sada moramo pokušati razumjeti što su točno te točkaste čestice.

### *2.3 Partonski model*

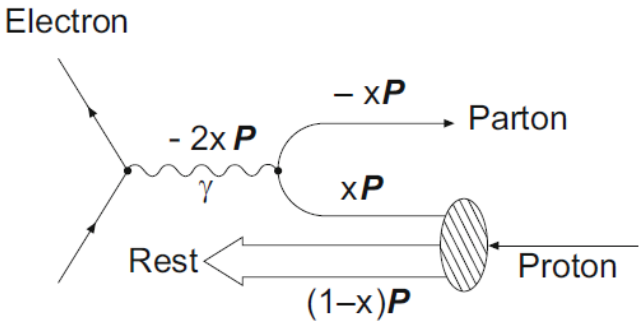
Interpretaciju dubokog neelastičnog raspršenja možemo poprilično pojednostaviti ako pametno odaberemo referentni sustav. Naravno fizika koja stoji iza te pojave se time neće promijeniti. Ako bi razmatrali proton kao sustav koji se giba velikom brzinom onda možemo zanemariti „prijenosni impuls“ i masu mirovanja sastavnica protona. Struktura protona je onda dana prvom aproksimacijom longitudinalnog impulsa sastavnica protona. Ovo je baza partonskog modela kojeg su osmislili Feynman i Bjorken. U tom modelu se sastavnice protona nazivaju partoni. Danas znamo da su nabijene čestice u protonu kvarkovi a nenabijene čestice gluoni prijenosnici jake sile. Rastavljanjem protona u pojedine partone koji se neovisno gibaju možemo interakciju elektrona s protonom razmatrati kao sumu neovisnih interakcija sa pojedinim partonima. Te interakcije možemo razmatrati kao elastična raspršenja. Ta aproksimacija vrijedi ako je trajanje foton-parton interakcije dovoljno kratko da se može zanemariti interakcija između pojedinih partona kao na primjer u sustavu koji se giba velikom brzinom. Ovo je impulsna aproksimacija koja vrijedi u dubokom neelastičnom raspršenju zato što je interakcija između partona na kratkim udaljenostima slaba.




Slika 2.5: Feynmanov dijagram partonskog modela u sustavu koji se giba velikom brzinom [25]

Ako napravimo tu aproksimaciju i pretpostavimo da su mase partona zanemarive i da je $Q^2 >> M^2c^4$, dobijemo direktnu interpretaciju Bjorkenove varijable $x = Q^2/2Mv$. Ona je dio četverovektorskog impulsa protona koji nosi pogođeni parton. Foton koji u laboratorijskom sustavu ima četverovektorski impuls $q = (\nu/c, \mathbf{q})$ interagira s partonom koji nosi četverovektorski impuls xP. Popularni referentni sustav koji zadovoljava ove uvjete je takozvani Breitov referentni sustav gdje foton uopće ne prenosi energiju ($q_0=0$).U ovom sustavu x je dio tri vektorskog tj. klasičnog impulsa partona. Prostorna rezolucija dubokog neelastičnog raspršenja je dana reduciranom valnom duljinom λ' virtualnog fotona. Ova vrijednost nije Lorentzova invarijanta nego ovisi o referentnom sustavu. U laboratorijskom sustavu ($q_0=\nu/c$) iznosi:

$$\lambda' = \frac{\hbar}{|q|} \approx \frac{2Mx\hbar c}{Q^2}$$




Slika 2.6: Feynmanov dijagram partonskog modela u Breitovom sustavu [26]

U Breitovom sustavu gdje je energija koju predaje virtualni proton jednaka 0, što se vidi iz Slike 2.6 gdje se udarenom partonu promijeni smjer, dobivamo jednostavniju jednadžbu:

$$\lambda' = \frac{\hbar}{\sqrt{Q^2}}$$

Na primjer ako je x = 0.1 i $Q^2 = 4(\text{GeV/c})^2$ onda dobivamo $\lambda' =' 10^{-17}$ m u laboratorijskom sustavu. Zbog toga vrijednost $Q^2$ ima očitu interpretaciju u Breitovom sustavu, to je mjera prostorne rezolucije kojom možemo proučavati danu strukturu. Sada možemo pokušati interpretirati strukturne funkcije koje smo prethodno spomenuli. Strukturne funkcije opisuju unutarnji sastav nukleona. Ako pretpostavimo da se nukleoni sastoje od različitih vrsta kvarkova q koji nose električni naboj $z_q \cdot e$ onda je udarni presjek elektromagnetskog raspršenja od kvarka proporcionalan kvadratu njegovog naboja tj. $z_q^2$. Funkcije koje opisuju distribuciju impulsa kvarka označavamo sa q(x) čemu slijedi da je q(x)dx očekivana vrijednost broja kvarkova tipa q=u,d,s u nukleonu čiji impuls leži u intervalu [x ,x + dx]. Distribuciju impulsa valentnih kvarkova označavamo sa $q_v(x)$ a odgovarajuću distribuciju anti-kvarkova u „moru“ označavamo sa $\bar{q}_s(x)$. Proton se sastoji od dva valentna u-kvarka i jednog valentnog d-kvarka. Zbog toga imamo

$$\int_0^1 u_v(x)dx = 2, \int_0^1 d_v(x)dx = 1$$

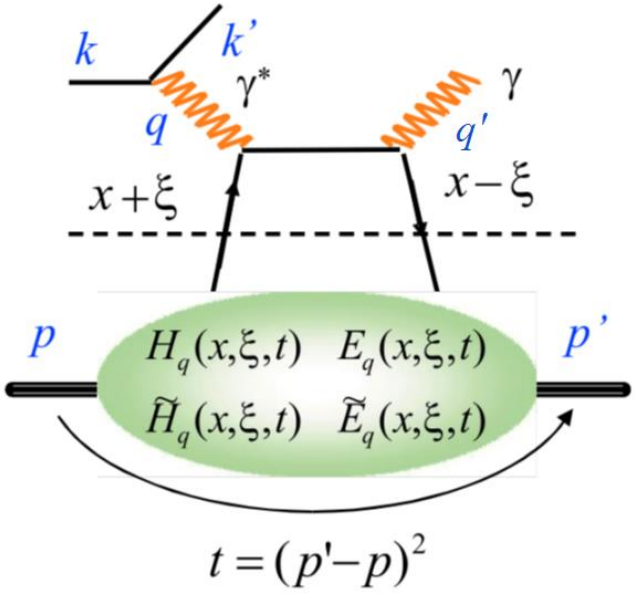
Strukturalna funkcija $F_2$ je zbroj distribucije impulsa u odnosu na odgovarajući x i $z_q^2$. Suma preko svih vrsti kvarkova i anti-kvarkova glasi:

$$F_2(x) = x \cdot \sum_{q\,=\,u,d,s} z_q^2[q(x) + \bar{q}_s(x)]$$

Gdje je $q(x) = q_v(x)+q_s(x)$ za u i d-kvarkove a $q(x)=q_s(x)$ za s-kvarkove. Te funkcije nazivamo kvarkovske distribucijske funkcije. Njih zajedno sa distribucijskim funkcijama gluona nazivamo partonske distribucijske funkcije ili PDF. Ako bi koristili ovaj model kao osnovu za prikaz strukture protona u našoj aplikaciji najvjerojatnije bi prikazivali PDF ili samo kvarkovske distribucijske funkcije kao jednu aproksimaciju strukture protona [8]. Naravno takva slika ne bi bila kompletna ali bi bila dovoljno funkcionalna za naše potrebe.

### *2.4 Duboko virtualno komptonsko raspršenje*

Duboko virtualno komptonsko raspršenje (DVCS) je proces u kojemu se elektron ili mion rasprši na nukleonu sa fotonom emitiranim od strane nukleona u konačnom stanju. Slika 2.7 prikazuje DVCS na protonu ep → e'p'γ. Ovo je fizikalni pomoću kojega su dobiveni modeli kojima ćemo prikazati strukturu protona u našoj aplikaciji [2].




Slika 2.7: Grafički prikaz DVCS [27]

Dok nam form faktori i partonske distribucijske funkcije daju vrijedne informacije o unutarnjoj strukturi protona, one još uvijek nisu dovoljne da tu strukturu u potpunosti opišemo i razumijemo. Sredinom devedesetih godina generalizirane partonske distribucije GPD su uvedene preko dubokog virtualnog komptonskog raspršenja. One daju višu razinu informacije od form faktora i partonskih distribucijskih funkcija. GPD je bogati i kompleksni okvir koji se teško raspisuje. Struktura nukleona se opisuje sa četiri različite kvarkovske GPD $H_q$, $E_q$, $\widetilde{H_q}$i $\widetilde{E_q}$ koji ovise o vrsti kvarka. GPD ovise o četiri varijable $Q^2$, x, ξ i t, ali kao i kod dubinskog neelastičnog raspršenja ovisnost o $Q^2$ se može predvidjeti pomoću kvantne kromodinamike i nećemo je pisati [6].Ovdje je t definiran kao $t=(p' - p)^2$ tj. kvadrat prenesene impulsa, a $x + \xi$ i $x - \xi$ su dijelovi longitudinalnog impulsa nukleona koje nosi pogođeni kvark prije i poslije raspršenja. Velik broj varijabli naznačuje bogatstvo informacija koje su sadržane u GPD u odnosu na PDF i FF [2].

Varijable x i ξ imaju vrijednosti između −1 i 1 ali zbog invarijantnoisti na inverziju vremena vrijedi:

$$H_q(x,-\xi,t) = H_q(x,\xi,t)$$

Gdje $H_q$ možemo zamijeniti sa $E_q$, $\widetilde{H_q}$, i $\widetilde{E_q}$ Kao posljedica možemo ograničiti raspon ξ na [0;1]. Za x< ξ, GPD se ponašaju kao distribucijska amplituda te se mogu interpretirati kao vjerojatnost pronalaska kvark anti-kvark para u nukleonu.

U DVCR varijable ξ i t se mogu izračunati iz kinematičkih varijabli čestica u početnim i konačnim položajima, ali varijabla x nije eksperimentalno dohvatljiva. Amplitude DVCR dane su integralima koje imaju sljedeći oblik:

$$\int_{-1}^{1} \frac{H_q(x,\xi,t)}{x-\xi+i\epsilon}dx = P\int_{-1}^{1} \frac{H_q(x,\xi,t)}{x-\xi}dx - i\pi H_q(\xi,\xi,t)$$

Gdje je P glavna vrijednost integrala i $H_q$ možemo zamijeniti sa $E_q$, $\widetilde{H_q}$, i $\widetilde{E_q}$. U gornjoj jednadžbi preko x-a ili integriramo ili ga izvrijednimo u x = ξ. Prvi slučaj dobijemo opservacijom realnog dijela amplitude DVCR, a drugi slučaj opservacijom imaginarnog dijela. Zbog čega DVCR dobije čak osam funkcija povezanih sa GPD-ma:

$$Re\mathcal{H}_q(\xi,t) = P\int_0^1 \left[H_q(x,\xi,t) - H_q(-x,\xi,t)\right] C^+(x,\xi)dx$$

$$Re\widetilde{\mathcal{H}}_q(\xi,t) = P\int_0^1 \left[\widetilde{H}_q(x,\xi,t) - \widetilde{H}_q(-x,\xi,t)\right] C^-(x,\xi)dx$$

$$Im\mathcal{H}_q(\xi,t) = -\pi\left[H_q(\xi,\xi,t) - H_q(-\xi,\xi,t)\right]$$

$$Im\widetilde{\mathcal{H}}_q(\xi,t) = -\pi\left[\widetilde{H}_q(\xi,\xi,t) - \widetilde{H}_q(-\xi,\xi,t)\right]$$

Naravno $\mathcal{H}_q$ i $\widetilde{\mathcal{H}_q}$ možemo zamijeniti sa $\mathcal{E}_q$ i $\widetilde{\mathcal{E}_q}$ čime dobijemo osam funkcija. Gdje je $C^{\pm}$ definiran kao

$$C^{\pm}(x,\xi) = \frac{1}{x-\xi} \pm \frac{1}{x+\xi}$$

Tih osam funkcija nazivamo komptonovi form faktori (CFF), one su eksperimentalno pristupačne te možemo uvesti dodatnu notaciju:

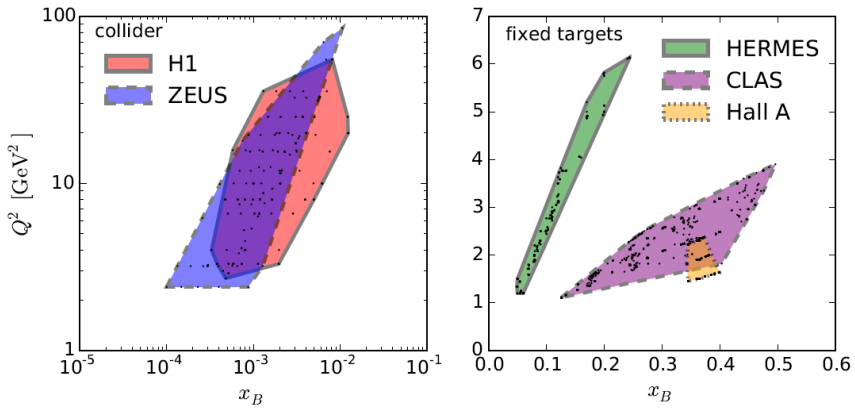
$$\mathcal{H}_q(\xi,t) = Re\mathcal{H}_q(\xi,t) + iIm\mathcal{H}_q(\xi,t)$$

Ovdje također $\mathcal{H}_q$ možemo zamijeniti sa $\mathcal{E}_q$, $\widetilde{\mathcal{H}_q}$, i $\widetilde{\mathcal{E}_q}$.

Ovime konačno imamo funkcije kojima ćemo prikazivati strukturu protona. Iako su GPD možda i najinformativnije što se tiče strukture protona malo su nezgodne za prikaz zbog svoje četvero dimenzionalne prirode. Tu su nam CFF puno pristupačniji je ovise samo o dvije varijable te se može lako prikazati u 3D prostoru, a još uvijek sadrže dovoljno informacija za naše potrebe. Razlika između GPD i CFF leži u tome da GPD opisuju događaje u samom protonu tokom DVCR, dok CFF opisuju događaje u cijelom procesu DVCS. Time naravno CFF ne opisuju strukturu protona direktno nego samo posredno preko procesa DCVS. Ovaj indirektan pristup sadrži dovoljno informacija za naše potrebe crtanja strukture protona u 3D prostoru.

### *2.5 Modeli CFF*

Kako bi mogli prikazati strukturu protona u aplikaciji moramo imati nekakav matematički model preko kojega ćemo prikazivati podatke. Matematički modeli su dobiveni prilagodbom eksperimentalno dobivenih podataka iz H1 i ZEUS detektora, te u HERMES, CLAS i Hall A eksperimentima. Ti podatci pokrivaju samo određene intervale fizikalnih vrijednosti x i $Q^2$ (Slika 2.8), a kako bi dobili vrijednosti ostatka interval mora se osloniti na prilagodbu pomoću raznih matematičkih funkcija.




Slika 2.8: Intervali podataka pokriveni DVCS mjerenjima [28]

Naravno to se ne radi nasumičnim pogađanjem nego se koristi prethodno dobiveno znanje ponašanja sličnih fizikalnih sustava. Ustvari postoje neke matematičke forme koje

očekujemo da će sustav poput protona u DVCS slijediti te prilagodbom dobivenih rezultata na te funkcije dobivamo jasniju sliku strukture protona izvan izmjerenih intervala.

Konkretno modeli koje ćemo koristiti u aplikaciji su KM modeli nazvani po njihovim tvorcima prof. Krešimiru Kumeričkom i prof. Dieter Muller. Počevši od modela KM09 koji su najstariji te se razlikuje od novijih modela uglavnom zbog činjenice su GPD komponente za kvarkove mora i gluone bile unaprijed prilagođene na podatke bez prave DVCS prilagodbe. Postoje dva KM09 modela KM09a i KM09b, oni se međusobno razlikuju tretmanom mjerenja udarnih presjeka u Hall A kolaboraciji. Model KM09a ne koriste ta mjerenja uopće dok ih KM09b koristi. Modeli KM10a i KM10b su poboljšane verzije KM09a i KM09b modela u kojima se provela prava DVCS prilagodba. Modeli KM10 i KMM12 uključuju nulte harmonike podataka dobivenih Hall A kolaboracijom, a KMM12 u sebi sadrži podatke polariziranih meta. Model KM15 je dobiven dodavanjem novih podataka iz 2015 dobivenih od Hall A i CLAS kolaboracije, te je na neki način riješio određene konflikte koji su se javljali u prijašnjim modelima [3].

## 3. Aplikacija

Za realizaciju web aplikacije korišten je programski jezik Python. Python je jednostavan za uporabu, otvorenog koda, jako dobro dokumentiran, dobro se slaže s drugim aplikacijama te je dovoljno moćan za naše potrebe [7]. Python je također jezik koji se trenutno uči u školama te se koristi općenito u industriji kada je potrebno brzo realizirati neki projekt. Također je jako korišten u vizualizaciji podataka zbog mnoštva praktičnih modula kao numpy i scipy, koji služe za obradu podataka. Analogno tome Python je jako popularan u znanstvenoj zajednici. Funkcionalnosti koje Python nema ovdje neće dolaziti do izražaja kao npr. manipulacija memorije. Dakako ako bi htjeli dodatno optimizirati izvršavanje aplikacije bilo bi pametno prijeći na neki drugi programski jezik kao npr. Java ili C#, ali bi to zahtijevalo puno dodatnog truda za malo dobitka pa se ne isplati.

### *3.1 Plotly i Dash*

Plotly je kanadska firma koja je stvorila istoimenu Python biblioteku za vizualizaciju podataka. Također su napravili Dash aplikacijski okvir koje trenutno koriste mnogi industrijski divovi kao CISCO, Tesla, nVIDIA i mnogi drugi. Plotly biblioteka u sebi sadrži mnoštvo funkcija za vizualizaciju podataka. Grafovi stvoreni Plotly funkcijama su interaktivni te se mogu zumirati i u slučaju 3D grafova se mogu rotirati. Također funkcije u Plotly-u imaju veliku mogućnost uređivanja svog izgleda čime se može postići prikaz po volji korisnika. Nama je Plotly zanimljiv jer želimo prikazati CFF u 3D prostoru, a vizualizacija je jaka strana Plotly-ja.

Dash je aplikacijski okvir kojim je moguće stvoriti cijele web aplikacije pomoću Pythona. Aplikacije se sastoje od dva glavna dijela:

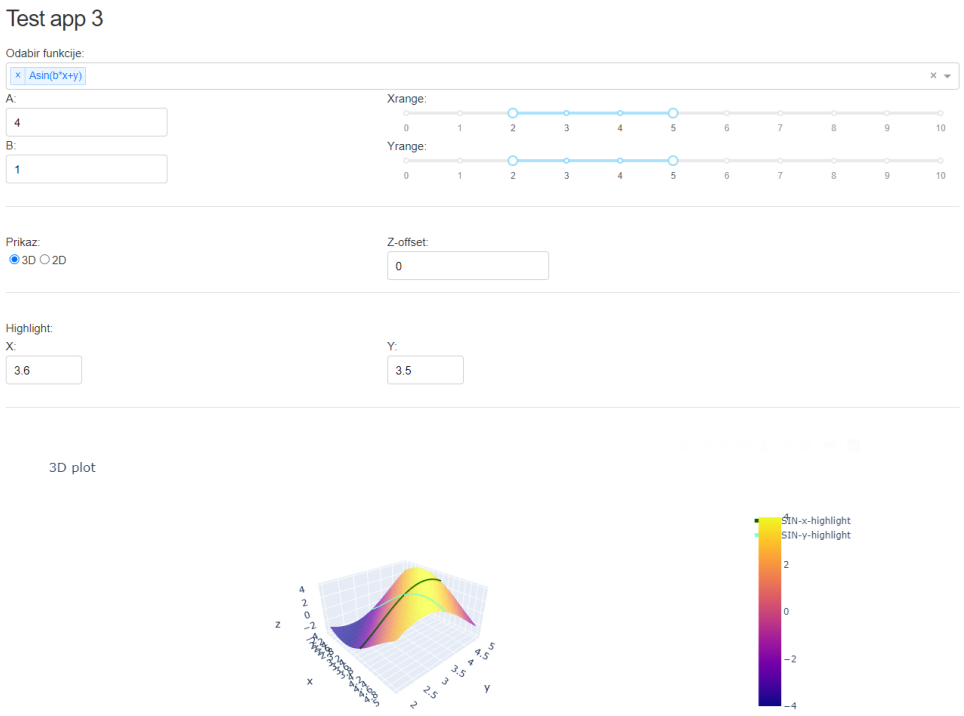
- Prvi dio u kojemu se definira izgled web aplikacije pomoću funkcija iz dcc (Dash core component) biblioteke i funkcija iz html (Dash HTML Components) biblioteke. U dcc biblioteci se nalaze razni interaktivni elementi poput polja za upis ili prikaz, a u html biblioteci se nalaze standardne html komponente kojima bi i inače uređivali neku web stranicu [9] [10].
- Drugi dio sadrži povratnu (`callback`) funkciju koja kao ulaz prima vrijednosti iz svih interaktivnih dcc funkcija. U toj funkciji definiramo što želimo da aplikacija radi svaki put kada promijenimo neku vrijednost u aplikaciji. Ovdje se u stvari nalazi cijeli Python kod koji u sebi sadrži go (Plotly graph objects) funkcije za prikaz

podataka. Povrat ove funkcije u našem slučaju je figure struktura podataka koja se prikazuje u `dcc.Graph` funkciji naše web aplikacije [11].

Stvaranje aplikacije teklo je kroz nekoliko faza u kojima su postepeno uvedene određene funkcionalnosti. Kako bi bolje opisali aplikaciju opisat ćemo kako se razvijala kroz te faze.

### *3.2 Alpha*

Alpha faza je početna faza razvoja aplikacije u kojoj je cilj bio istražiti mogućnosti Ploty biblioteka i Dash aplikacijskog okvira. U ovoj fazi aplikacije nije imala nikakvu fizikalnu poveznicu nego je crtala čisto matematičke funkcije sin, cos, sinh i cosh. Također izgled aplikacije nije bio u fokusu nego su dcc-e bile redom nadodane u službi funkcionalnosti aplikacije.




Slika 3.1: Izgled alpha verzije aplikacije

Počevši od vrha imamo padajući izbornik za odabir funkcija koje se crtaju. Želimo da se na istom grafu mogu crtati više funkcija kako bi ih mogli lakše uspoređivati. Kako bi to realizirali koristimo `dcc.Dropdown` funkciju u kojoj smo postavili parametar multi na `True`. U toj funkciji moramo postaviti parametar `options` listu vrijednosti koja sadrži opcije ovog izbornika. Svaki član te liste je rječnik koji se sastoji od dva elementa. Prvi element ima ključ `label` te se njemu pridodana vrijednost prikazuje u aplikaciji kada

otvorim padajući izbornik. Drugi element ima ključ `value` a njegova vrijednost se šalje u listu `value` parametra. U povratnu funkciju se onda vraća ti lista koja odgovara `value` parametru. Na početku smo postavili da je odabrana sinus funkcija.

Ispod toga imamo dva polja za upis vrijednosti konstanti A i B koje koristimo u funkcijama koje crtamo. Koristimo `dcc.Input` funkcije jednu za A i jednu za B, kako bi ih razlikovali dali smo ih različite vrijednosti `id` parametra, konkretno `numA` i `numB`. Također pošto su A i B brojevi treba postaviti parametar `type` u `number`. Svaki input `dcc.Inupt` ima svoju `value` parametar gdje je vrijednost koju šalje u povratnu funkciju.

Desno od toga imamo dva `dcc.RangeSlider` elementa kojima odaberemo raspon na x i y osi. Oba klizača imaju jednake minimalne i maksimalne vrijednosti 0 i 10, te korak od 0,5. Opet se međusobno razlikuju po vrijednostima svojih `id` parametra. Također je potrebno dodati oznake za vrijednost klizač jer se ne stavljaju automatski. To se radi postavljanjem `marks` parametra pomoću for petlje u kojoj se napravi rječnik u kojem su ključevi vrijednosti na klizaču, a vrijednosti ključeva su stringovi u kojima piše odgovarajuća vrijednost klizača. Također moramo postaviti parametar `allowCross` na `false` kako bi onemogućili da se maksimalna vrijednost postavi ispod minimalne što nam za odabir raspona nikako ne odgovara.

Ispod polja za upisa A i B imamo `dcc.RadioItems` za odabir vrste grafa koje ćemo crtati. Cilj naše aplikacije je da može crtati i 3D i 2D grafove te će korisnik taj odabir ovdje napraviti. Slično kao i kod `dcc.Dropdown` ima `options` listu rječnika. Ovdje su samo dva člana te liste jedan za 3D i jedan za 2D. Za razliku od `dcc.Dropdown` možemo odabrati samo jednu vrijednost te zbog toga povratnoj funkciji dajemo uvijek samo jedan string koje ima vrijednost ili '3d' ili '2d'.

Desno od gumba za odabir vrste grafa imamo polje za upis `Zoffset`-a. `Zoffset` je vrijednost na z osi za koju ćemo razmaknuti svaku funkciju na grafu čime ćemo povećati preglednost grafa. Ovo je realizirano kao i polja za upis A i B pomoću `dcc.Inputa` sa standardno postavljenom vrijednosti 0 i korakom od 1.

Sljedeće u aplikaciji imamo dva polja za upis `highlight` vrijednosti x-a i y-a. To su vrijednosti za koje se crtaju grafovi u 2D prikazu. Za te vrijednosti se također u 3D prikazu crtaju 2D funkcije koje nam onda naglašavaju oblik funkcije na toj vrijednosti. Ovo je opet

realizirano pomoću `dcc.Input` funkcije koja ima `type number`, minimalne i maksimalne vrijednosti od 0 i 10 kao i `dcc.RangeSlider` za raspon x-a i y-a. Također ova polja imaju korak od 0,1 da bi se glađe pomicala tokom rada aplikacije.

Zadnji dcc element na dnu ovog dijela aplikacije je `dcc.Graph`. Ovdje se crta naš graf tj. točnije rečeno ovdje se crta sliku `fig` koji nam vrača povratna funkcija. Pošto nam povratna funkcija uvijek vrača sliku imena `fig` dovoljno je imati samo jedan `dcc.Graph` u kojemu se onda mogu naizmjenično crtati i 2D i 3D prikazi.

Nakon što smo definirali izgled aplikacije moramo definirati `callback` aplikacije. U `callback`-u definiramo output i input naše aplikacije na u `app.callback` funkciju u prvi parametar stavimo listu outputa, a u drugi listu inputa. U našoj aplikaciji imamo samo jedan output u kojem upišemo `id dcc.Graph` funkcije i tip podatka koji ta funkcija prima, a ovdje je to figure struktura podatka. Za input je slična priča samo što ovdje moramo napraviti listu koja sadrži po jednu input funkciju za svaki parametar koji upisujemo u aplikaciji. Svaka ta input funkcija sadrži `id` od jednog dcc elementa i tip podatka koji taj dcc element predaje povratnoj funkciji, za sve naše dcc elemente je to njihov `value` parametar.

Sada je konačno vrijeme za definirati povratnu funkciju `update_graph` koja se definira kao i svaka druga Python funkcija, samo trebamo postaviti parametre te funkcije u isti redoslijed kao i u listi input funkcije u `app.callback`-u. Na početku `update_graph` funkcije definiramo x i y kao numpy liste od 100 elemenata s graničnim vrijednostima definiranim u aplikaciji pomoću `dcc.RangeSlider`-a.

Nakon toga imamo if grananje u kojem provjeravamo koja je vrsta grafa odabrana u aplikaciji. Prvo grananje je za 2D prikaz a drugo je za 3D prikaz. U svakoj grani definiramo `fig` sliku koji onda vraćamo `dcc.Graph` funkciji.

U 2D grananju prvo definiramo rječnik `funkdict2D` gdje su ključevi vrijednosti iz `dcc.Dropdown`, a vrijednosti tih ključeva su liste od 2 matematičkih funkcija po jednu za svaki graf koji crtamo. Crtamo graf za x `highlight` i graf za y `highlight`, a kako bi to napravili definiramo `fig` pomoću `make_subplots` kao figuru koja ima dva subplota. Ovdje također definiramo njihove naslove. Kako bi lakše razlikovali grafove definirali smo dvije liste k i h. Lista k sadrži razne boje koje će naši grafovi imati, a lista h sadrži liste stilova linija koje će naši grafovi imati. Također imamo jedan pomoćni brojač j pomoću

kojega ćemo implementirati `Zoffset`. Sada pomoću for petlje crtamo sve funkcije odabrane u `dcc.Dropdown` koje se nalaze u `Fselect` listi koje su nam sad ključevi za rječnik. Prolazimo kroz listu i za svaku vrijednost iz te liste u `fig` dodajemo po dvije `go.Scatter` funkcije za crtanje 2D grafova. Prva `go.Scatter` funkcija crta `x-highlight`, a druga `y-highlight` odabrane funkcije. Parametar x za prvu `go.Scatter` funkciju je y, a za drugu je x parametar x. Parametar y je funkcija izvađena iz rječnika `funkdic2D` kojoj pridodamo vrijednost `Zoffset`-a pomnoženu za j. Pomoćni brojač j nam kreče od 0 te je zato prva funkcija koja se crta uvijek centrirana, a onda je svaka slijedeća pomaknuta za `Zoffset` u plus y smjeru. Treba još napomenuti da se parametar `name` definira direktno iz brojača i, parametar mode je postavljen na `lines` kako bi grafovi bili linije. Stil tih linija se definira pomoću line parametra preko rječnika u kojoj se ključu `color` pridoda vrijednost iz k liste, a ključu `dash` pridoda vrijednost iz h liste. Također prvi `go.Scatter` nam ide u lijevi subplot a drugi ide u desni subplot naše `fig`. Na kraju petlje povećamo brojač j za jedan kako bi osigurali da `Zoffset` funkcionira kako želimo. Nakon svega toga postavimo naslove osi grafova te naslov cijele slike. Također postavimo opciju `fig.layout.uirevision` na `true` kako se manipulacije koje korisnik napravi na grafu u aplikaciji ne bi resetirali kada mijenjamo ulazne parametre. Na kraju ove grane vračamo aplikaciji `fig` figure koji se onda prikazuje u `dcc.Graph` elementu.

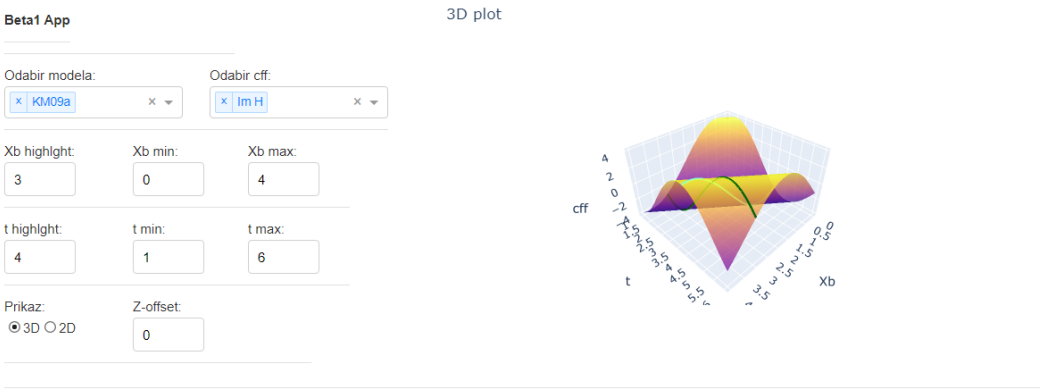
Druga grana if grananja nam služi za prikaz 3D grafa. Na početku grananja prvo stvorimo `meshgrid` iz lista x i y kako bi dobili dobar prikaz u 3D prostoru. Također kako ćemo crtati linije u 3D prostoru morat ćemo za svaku točku na liniji definirati sve tri koordinate. Što znači da moramo za svaku `hightlight` vrijednost napraviti listu od 100 elemenata koja sadrži točno tu vrijednost. Onda definiramo rječnike preko kojih ćemo pristupati odabranim funkcijama. Radi preglednosti napravljena su tri rječnika jedan za 3D plohu i po jedan za svaku 3D `hightlight` liniju. U svakom od tih rječnika ključevi su vrijednosti iz `dcc.Dropdown` elementa, te svaki ključ odgovara jednoj funkciji u svakom rječniku. Razlike između tih funkcija proizlaze iz toga što neke različite vrijednosti držimo konstantnima u različitim grafovima. Nakon toga definiramo varijablu data kao listu te stavimo pomoćni brojač j na nulu. Nakon toga ulazimo u for petlju pomoću liste ključeva za naše rječnike. U svakom prolazu for petlje u listu data pridodamo po jednu go.Scatter3d funkciju za svaki `hightlight`, te jednu `go.Surface` funkciju za crtanje plohe. Prvi parametri `go.Scatter3d` su x, y i z koordinate gdje su x i y odgovarajuće liste a z se vari

iz odgovarajućeg rječnika te zbroji za `j*Zoffset` kao i kod 2D prikaza. Ostali parametri su `name` koji se postavlja sa ključem iz rječnika kojemu se pridoda ili `-y-highlight` ili `-x-highlight` ovisno o funkciji, također definiramo mode da bude `lines` kako bi grafovi bili linije te im definiramo različite boje kako bi se lakše raspoznale. Funkcija `go.Surface` je slična što se tiče x, y i z parametara samo koristimo druge liste i drugi rječnik, te parametar `name` postavimo pomoću ključa . Postavimo prozirnost plohe sa parametrom `opacity` na 0.7 kako bi više ploha bilo preglednije na istom grafu. Također kako bi izbjegli preklapanje legendi pokraj grafa postavimo parametar `showscale` na `not(j)`. Time će se samo u prvom prolazu nacrtati legenda pokraj grafa, a na svakom slijedećem će vrijednost `not(j)` biti `false`. Sada imamo ispunjenu data listu pomoću koje možemo napraviti `fig`. To napravimo pomoću `go.Figure` funkcije kojoj zadamo listu data kao parametar. Postavljamo parametar `hoverinfo` na `skip` kako nam se ne bi prikazivali podatci o točki kada stavimo kursor miša preko neke točke na grafu. Također ćemo definirat izgled grafa pomoću `fig.update_layout` gdje definiramo naslov grafa te skale i omjere pojedinih osi na grafu. Opet moramo postavimo `fig.layout.uirevision` na `true` te vratimo `fig` aplikaciji .

Kako bi se čim pokrenemo aplikaciju pokazala neka slika na početku programa ispred `app.layout` postavimo početnu vrijednost `fig` figure. To je napravljeno jednako kao i druga grana if grananja u povratnoj funkciji. Jedina razlika je u tome što su ovdje parametri unaprijed definirani te se ovaj dio koda izvršava samo jednom i to kad pokrenemo aplikaciju. Sada je aplikacija u potpunosti funkcionalna te ima većinu mogućnosti koje ima i konačna aplikacija. Vidimo da nam je Plotly omogućio pregledan 3D prikaz zadanih funkcija, te da će biti pogodan za implementaciju fizikalnih funkcija.

### *3.3 Beta*

U beta verziji aplikacije smo više pažnje posvetili izgledu aplikacije te smo implementirali `cffs.py` Python modul koji u sebi sadrži fizikalne funkcije pomoću kojih crtamo CFF.




Slika 3.2: Izgled Beta verzija aplikacije

Kako bi malo uredili izgled aplikacije uveli smo html implementaciju u Dash aplikacijskog okvira preko `dash_html_components`. I u alpha verziji smo koristili `html.div` za primitivno razdvajanje različitih dcc elemenata, te `html.label` za označavanje dcc elemenata. Sada ćemo tome sve elemente postaviti u `html.Table` kako bi sve bilo ravnomjerno raspoređeno. Koristimo `html.Tr` za definirati redove i `html.Td` za definirati stupce u redovima. Također ćemo pomoću `style` parametra ravnomjerno rasporediti određene elemente u stupcima. `Style` parametar html funkcija je način da definiramo stil te html funkcije. Taj parametar je rječnik koji kao ključeve koristi `style` parametre koji se inače koriste u HTML-u, a kao vrijednosti ključeva se postave željene vrijednosti tih parametra.

Promijenili smo nazive koje koristimo u dcc elementima kako bi nam bolje odgovarali fizikalnim funkcijama, te smo uveli još par promjena u samim funkcijama. Ova verzija aplikacije ima dva padajuća izbornika dok je alpha verzija imala samo jedan. To je zato što ovdje biramo i model po kojemu će se crtati CFF funkcije i točno koju CFF funkciju želimo crtati. Također smo promijenili način odabira raspona vrijednosti na x i y osima. U alpha verziji smo koristili `dcc.RangeSlider` funkcije, dok ovdje koristimo po dvije `dcc.Input` funkcije za svaki raspon. Gdje u jednu upisujemo minimalnu vrijednost a u drugu maksimalnu vrijednost raspona, čime možemo lakše odabrati željeni raspon.

Što se tiče povratne funkcije došlo je do nekoliko manjih promjena. Glavna promjena je micanje svih rječnika iz aplikacije. Umjesto njih različitim funkcijama pristupamo preko različitih Python funkcija koje smo definirali na početku aplikacije. Definirali smo dvije funkcije `gcff3D` i `gcff2D`, obije funkcije imaju jednak input te pristupaju istoj funkciji iz

`cffs.py` modula koja se naziva `getcff`. Primaju dvije liste koordinata, oznaku modela i naziv CFF koji se crta, a vrijednost $Q^2$ koju također prima funkcija `getcff` je zadana globalnom varijablom i postavljena je na 4. Glavna razlika je u outputu te dvije funkcije: `gcff2D` nam vraća listu vrijednosti što nam odgovara kao input za `go.Scatter` i `go.Scatter3D` funkcije, dok `gcff3D` nam vraća matricu koja nam služi za input u `go.Surface` funkciji.

Funkcija `gcff3D` se poziva svugdje u kodu gdje je `go.Surface` funkcija kao bi izračunali vrijednosti na z osi. Funkcija u sebi sadrži dvije ugniježđene for petlje koje prolaze kroz liste koordinata te za svaki par koordinata pozovu funkciju `getcff` iz `cffs.py` modula. Ta funkcija izračuna vrijednost i spremi ju na odgovarajuće mjesto u matrici r. Nakon što se ispuni ta matrica se vrati funkciji `go.Surface` te se po njoj crta graf.

Funkcija `gcff2D` je slična funkciji `gcff3D` glavna razlika je u tome što prije nego krenemo računati vrijednosti pomoću `getcff` moramo raspoznati koja je vrijednosti stalna a koja je varijabilna za svaki određeni poziv `gcff2D` funkcije. To je posljedica činjenica da koristimo istu funkciju za crtanje svih `go.Scatter` i `go.Scatter3d` funkcija. Zbog toga `gcff2D` u sebi ima jedno if grananje gdje se provjerava koji su tip podatka proslijeđene koordinate. Uvijek će jedna koordinata biti lista vrijednosti a druga samo vrijednost. Nakon što odredimo koja koordinata je lista vrijednosti onda znamo da po njoj moramo proći for petljom. U svakom prolazu kroz petlju pozovemo `getcff` funkciju iz `cffs.py` modula te izračunatu vrijednost stavimo u listu. Tu listu onda vratimo natrag u funkciju koja je pozvala `gcff2D` funkciju gdje se koristi kao lista vrijednosti na z osi za 3D funkciju ili y osi za 2D funkciju.

Još je dodana opcija za odabir vrste skale na x osi, tj. mogućnost biranja između linearne i logaritamske skale. To je uvedeno zbog činjenice da se logaritamske skale inače često koriste u fizikalnim istraživanjima. Kako bi to postigli dodali smo još jedan `dcc.Radioitems` element u korisničino sučelje kao i kod opcije za odabir vrste grafa. Koju god opciju odaberemo u tom elementu tu vrijednost proslijedimo povratnoj funkciji. Povratne funkcije smo modificirali tako da u linijama koda gdje definiramo x os dodamo opciju za definiranje tipa skale koju ta os ima, te u tu opciju postavimo vrijednost koja je odabrana u korisničinom sučelju aplikacije. Time ako je odabrana `log` opcija postavimo `log`

te će skala na x osi biti logaritamska, u suprotnom će biti odabrana opcija `lin` te će vrijednost biti postavljena na `linear` te će skala biti linearna.

Javio nam se jedan problem u prikazu grafova koji je smanjiti čitkost samih grafova. Zbog divergencije funkcije `cffs` u `xB = 0`, u ovom dijelu rada ćemo za Bjorkenov x koristiti oznaku xB kako bi je lakše razlikovali od osi x, koja nam na grafu pomakne skalu z osi u red veličine desetak milijuna, na z osi ne prikazujemo `cffs` nego `xB*cffs` te time obuzdamo divergentno ponašanje. Kako bi to napravili modificiramo funkcije `gcff2D` i `gcff3D` tako da nakon što dobijemo rezultat funkcije iz `cffs.py` u matricu ili listu ne spremamo samo taj rezultat nego taj rezultat pomnožen sa `xB`. Za funkciju `gcff3D` i `gcff2D` u slučaju da crtamo `highlight` za statički t, u svakom prolazu petlje uzmemo vrijednost xB koji koristimo u računu vrijednosti funkcije iz `cffs.py`. Dok za slučaj kada računamo `gcffs2D` za statičku xB u svakom prolazu petlje koristimo točno taj xB. Sada smo dobili razuman prikaz funkcija u oba prikaza te su grafovi puno čitkiji.

### 3.4 Prva puna implementacija

Zbog problema sa serverom na kojemu je bila planirana aplikacija i hosting aplikacije morao sam pronaći neku alternativu. Na web stranici Dash-a se preporučuje platforma Heroku [30]. Heroku je web hosting servis za aplikacije koji podržava više programskih jezika te je u razvoju od 2007. g., također su osnovne funkcionalnosti koje nama trebaju dostupne besplatno [29]. Što se tiče samog deploymenta aplikacije u pravilu je dosta jednostavan samo što je za našu aplikaciju bilo potrebno malo dodatnog posla. Prvo je potrebno napraviti git repozitorij naše aplikacije. Git je sustav za kontrolu verzije aplikacije koji se koristi diljem IT industrije zbog svojih pogodnosti pogotovo kada je u pitanju razvoj aplikacije u timovima ljudi. Inače se koriti github za online hosting git repozitrorija ali za naše potrebe je napravljen lokalni git repozitorij za mapu u kojoj se nalazi aplikacija i sve dodatne biblioteke. Osim svih datoteka nužnih za pokretanje aplikacije potrebno je dodati i par datoteka za funkcionalnost Heroku servisa. To su gitignore, procfile i requirements.txt datoteke. Gitignore datoteka nam definira koje vrste datoteka se neće dodati u git repozitorij se te time neće prosljediti web servisu. Procfile definira  što točno pokrećemo na Heroku web servisu, u našem slučaju pokrećemo web aplikaciju preko sljedeće linije koda: `web: gunicorn app:server`. Gunicorn je Python web server poslužitelj koji je preporučen u Dash dokumentaciji a server je ime server varijable u našoj aplikaciji. Zadnja datoteka koju

trebamo je requirements.txt koji u sebi sadrži popis svih biblioteka potrebnih za pokretanje same aplikacije. Moguće je automatski ispuniti requirements.txt datoteku pomoću naredbe `pip freeze > requirements.txt`. Nakon što sve naše datoteke dodamo u git repozitorij te napravimo `commit` potrebno je još pokrenuti naredbu `git push heroku master`. Nakon toga će se aplikacija početi slati Heroku servisu, koji će iz reqirements.txt datoteke vidjeti da se radi o Python aplikaciji te će koristeći odgovarajući `buildpack` za Python osigurati da aplikacija radi. Problem u našoj aplikaciji se javlja u činjenici da pomoćne biblioteke za računanje fizikalnih funkcija u sebi sadrže kod napisan u Fortran-u. Python `buildpack` Heroku servisa u sebi ne sadrži potrebne biblioteke kako bi se osigurao rad Fortran koda. Kako bi riješili taj problem potrebno je omogućiti funkcionalnost korištenja više `buildpack`-ova istovremeno te u Atpfile datoteku upisati koji `buildpack`-ovi su nam potrebni. Nakon čega možemo opet `commit`-ati datoteke i pokrenuti `git push heroku master` naredbu. Sada će se osim funkcionalnosti za Python na server postaviti i funkcionalnost za Fortran te će sada aplikacija proraditi.

Nakon pokretanja aplikacije pojavio se jedan problem: velik dio modela nije moglo prikazati niti jednu plohu u 3D prikazu aplikacije. To je očito bila posljedica činjenice da su neki modeli puno kompleksniji od drugih zbog čega im je potrebno i duže vrijeme za stvaranje 3D prikaza. Očito Heroku ima sustav na svojim serverima koji ograničava vrijeme izvođenja implementiranih aplikacija. Kako bi to popravio napravio sam promjenu u povratnoj funkciji aplikacije. Prvotno je aplikacija na početku povratne funkcija stvorila dvije liste vrijednosti, po jednu za x i za y os, tj. za vrijednosti xB i t. Onda bi dalje s njima ili išli u granu za crtanje 2D prikaza ili 3D prikaza ovisno o tome što bi korisnik odabrao. Svaka od tih listi se sastojala od 50 vrijednosti što ne stvara probleme u 2D prikazu jer tamo imamo za svaki odabrani CFF iz odabranih modela napraviti po jedan račun za svaku vrijednost u listama tj. 100 računa. Ali za 3D model bi onda za svaki CFF iz odabranih modela morali napraviti 2500 računa. Vidimo iz rada aplikacije da je to jednostavno previše računa za kompleksnije modele te za njih nije bilo moguće napraviti niti jednu plohu. Kako bi to popravio pomaknuo sam stvaranje lista u svaku granu zasebno. Time smo uspjeli bez gubitka performanse podići broj vrijednosti u listama za 2D prikaz sa 50 na 100. Te smo pokušali pronaći balans između broja računa i glatkosti plohe u 3D prikazu. Prvo sam smanjio broj na 20 vrijednosti po osi sa 50 ali je onda ploha bila jako gruba sa puno rubova. Povećanjem broja vrijednosti na 30 po osi je rezultiralo puno glađom plohom, a još je uvijek

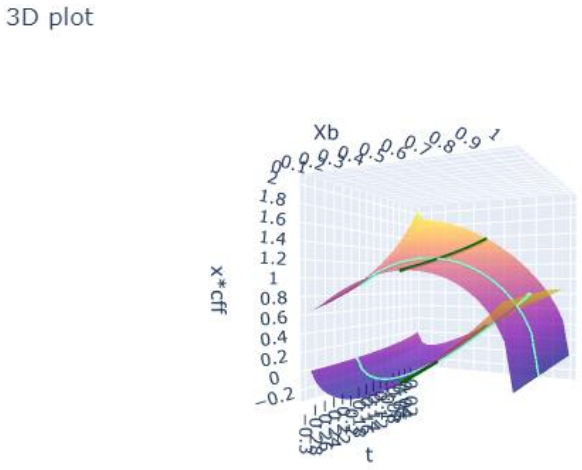
bilo moguće dobiti po barem jednu plohu ovisno o odabranom modelu, a ovisno o kombinaciji CFF i modela moguće je dobiti i puno više ploha za modele u kojima prethodno nismo mogli dobiti niti jednu plohu.

### 3.5 Finalna verzija aplikacije

Sumirajmo još jednom svu funkcionalnost aplikacije. Svrha aplikacije je da prikazuje strukturu protona za parametre koje korisnik odabere. Pod strukturom protona mislim na prikaz komptonovih form faktora, te form faktore dobivamo iz određenih teorijskih modela. Modeli koji su implementirani u ovoj aplikaciju su takozvani KM modeli, konkretno modeli KM09a, KM09b, KM10, KM10a, KM10b, KMM12 i KM15. Odabir modela se vrši preko padajućeg izbornika od strane korisnika, također korisnik u drugom padajućem izborniku odabira bira koji će se od osam postojećih komptonskih form faktora prikazivati za sve odabrane modele. U samom aplikaciji se na z osi ne prikazuje samo vrijednost komptonskih form faktora nego umnožak vrijednosti xB sa vrijednosti komptonskih form fakotora. To se radi kako bi se ukrotila divergencija komptonskih form faktora u xB=0. Nakon tog odabira korisnik može preko nekoliko polja za upis definirati raspone vrijednosti xB na x osi i t na y osi, te odabrati po jednu `hightlight` vrijednost za svaku os. Ta `highlight` vrijednost ima dvije svrhe u aplikaciji, prvo u 3D prikazu definira vrijednosti za koje će se crtati presjeci 3D plohe, čime ćemo moći naglašavati oblik plohe na tim odabranim vrijednostima. Druga svrha `highlight` vrijednosti je odabir vrijednost za koje će se crtati funkcije u 2D prikazu. Pri dnu aplikacije postoje dva para gumbova, jedan koji služi za odabir između 3D i 2D prikaza, a drugi za odabir između linearne i logaritamske skale na x osi. Pokraj tih gumbova je još jedno polje za upis vrijednosti u kojemu se upisuje željeni razmak između pojedinih odabranih CFF i modela.

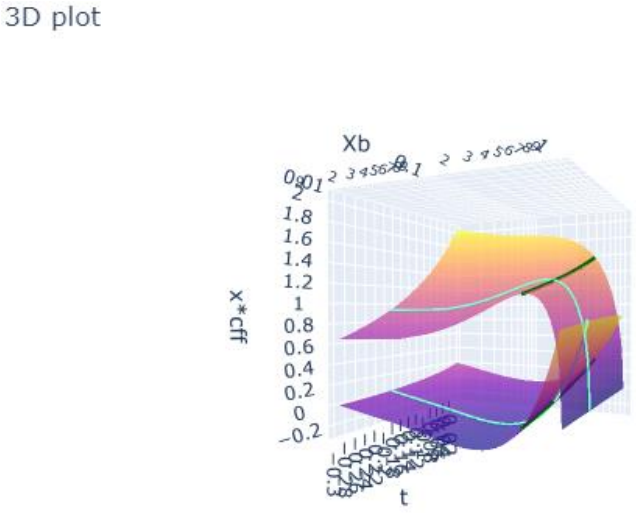
Slika 3.3: Izgled finalne verzije aplikacije

Sada ćemo proći kroz par primjera prikaza grafova iz aplikacija te objasniti kako su oni nastali. Za prvi primjer ćemo odabrati model KM09a, CFF Im H i Re H, raspon xB od 0,01 do 1, raspon t od –0,03 $GeV^2$ do –0,01 $GeV^2$, `highlight` vrijednost za xB 0,5 a za t –0,2. Odabran je 3D prikaz, s linearnom skalom na xB osi te razmakom između ploha jednakim nuli.




Slika 3.4: Prvi primjer 3D prikaza sa linearnom skalom na xB osi

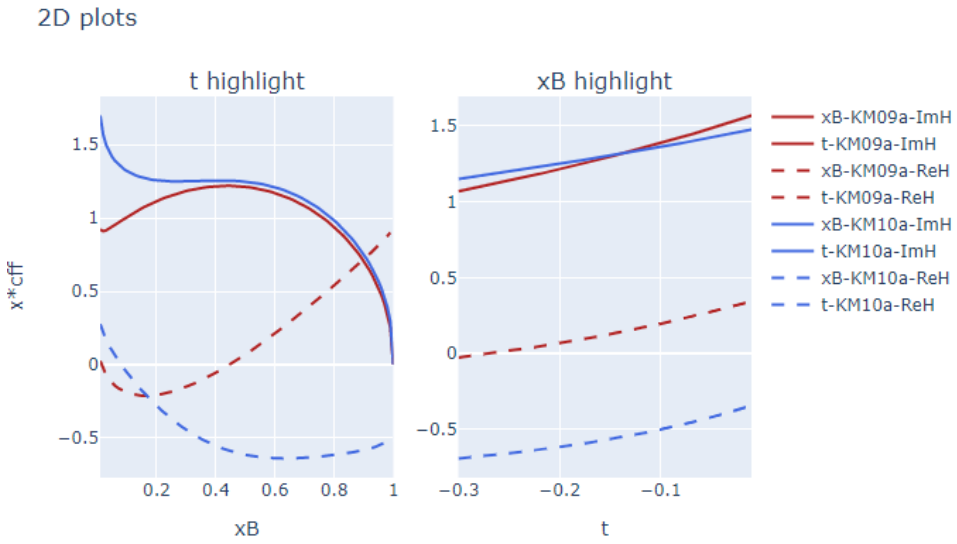
Sliku je moguće slobodno rotirati u 3D prostoru te je približavati ili udaljivati. Vidimo da su plohe prikazane gradijentom boja gdje su najniže vrijednosti na svakoj plohi tamno plave, najviše vrijednosti su žute, a između njih se prelijevaju nijanse ljubičastih i narančastih boja. Odabrani `highlight` presjeci ploha su prikazani pomoću linija tamno zelene i cijan boje kako bi se dobro isticali od plohe te jedna od druge. Kada stavimo kursor preko neke plohe prikaže nam se oznaka na kojoj piše naziv plohe u formatu naziv modela, minus te naziv komptonovog form faktora, ovdje konkretno imamo KM09a-ImH i KM09a-ReH.




Slika 3.5: Prvi primjer 3D prikaza sa logaritamskom skalom na xB osi

Na slici 3.5 su odabrani svi isti parametri kao i na prethodnoj slici samo je odabrana logaritamska skala za xB os. Vidimo da je prikaz više-manje isti samo je očekivano oblik plohe različit, te je naravno slaka na x osi logaritamska.

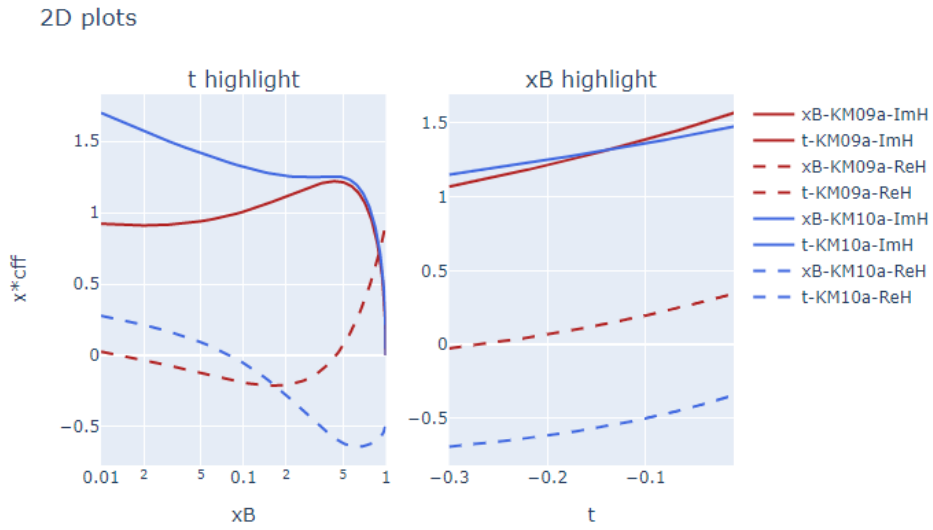
Za slijedeći primjer ćemo odabrati 2D prikaz, modele KM09a i KM10a, a sve ostale parametre ćemo ostaviti kao i na prvom primjeru. Odabrali smo modele KM09a i KM10a jer je KM10a poboljšana verzija modela KM09a te ovdje lako vidimo kakve su posljedice korištenja prave DVSC prilagodbe.




Slika 3.6: Prvi primjer 2D prikaza sa linearnom skalom na xB osi

Vidimo na prikazu da se svaki model prikazuje jednom određenom bojom, u našem primjeru KM09a je prikazan crvenom bojom, a KM10a plavom bojom. Također svaki komptonov

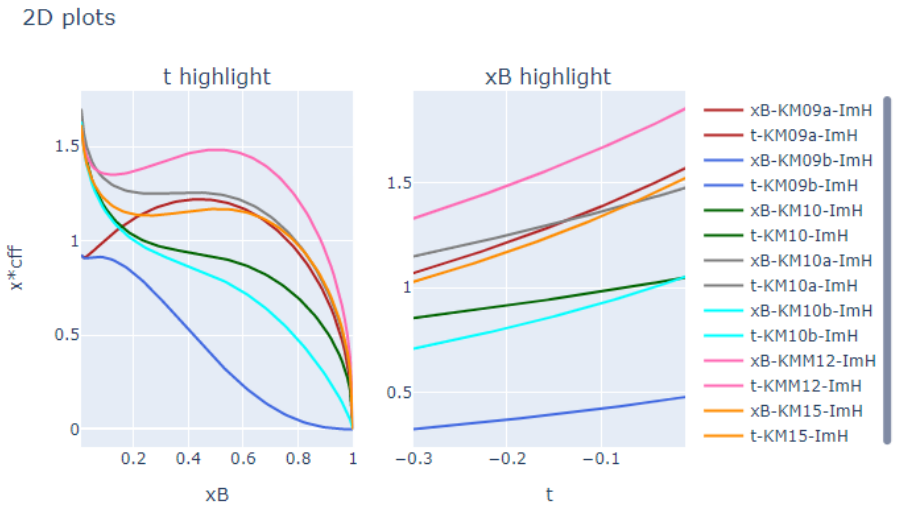
form faktor je prikazan jednom vrstom linije, u našem primjeru Im H je prikazan punom linijom dok je Re H prikazan iscrtanom linijom. Sa desne strane slika je legenda koja označuje nazive svake linije koji su u istom formatu kao i kod 3D prikaza samo što je na početku naziva dodana oznaka koja označuje koja je vrijednost na x osi u grafu.




Slika 3.7: Prvi primjer 2D prikaza sa logaritamskom skalom na xB osi

Na slici 3.7 je ista konfiguracija kao i na prethodnoj slici samo je vrsta skale na xB osi postavljena na logaritamsku. Vidimo da se oblik grafa promijenio na grafu koji prikazuje ovisnost xB*cff o xB, dok na grafu koji prikazuje ovisnost xB*cff o t je ostao isti. To je posljedica toga da se grafovi crtaju za iste vrijednosti statičkih parametara neovisno o odabranoj skali na osima. Vrijednost statičkog parametra se bira preko polja za upis `highlight` vrijednosti.

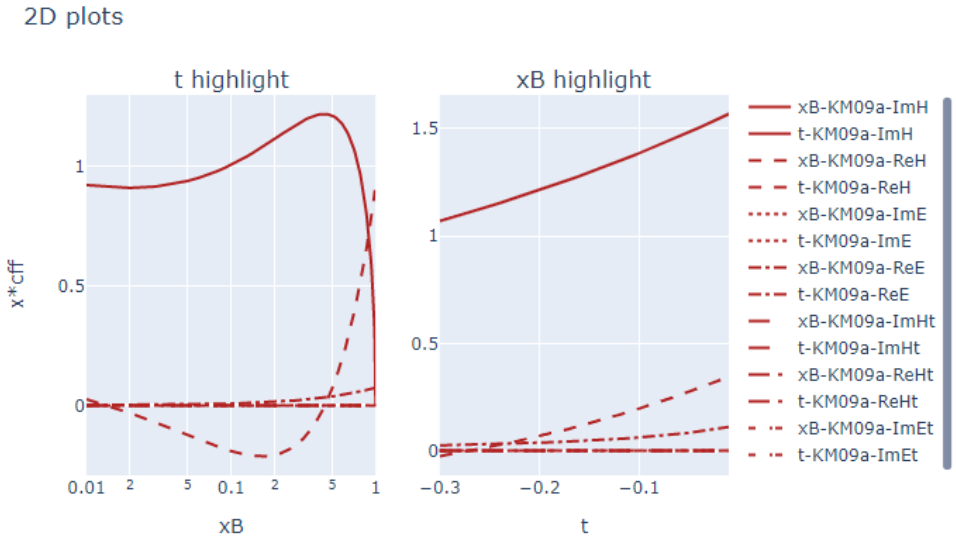
Za idući prijem ćemo odabrati sve modele ali samo jedan komptonov form faktor Im H, skala za xB os će biti linearna, a svi ostali parametri će biti isti kao i u prethodnim primjerima.




Slika 3.8: Drugi primjer 2D prikaza sa svim odabranim modelima

Ovdje možemo primijetiti kako svaki model ima svoju boju, ali te boje nisu određene samim modelima nego redoslijedom odabira modela. To znači da će prvi model koji odaberemo imati crvenu boju, drugi plavu, itd. Tako da između raznih grafova ne možemo uspoređivati samo po bojama linija nego se treba uspoređivati po njihovim nazivima.

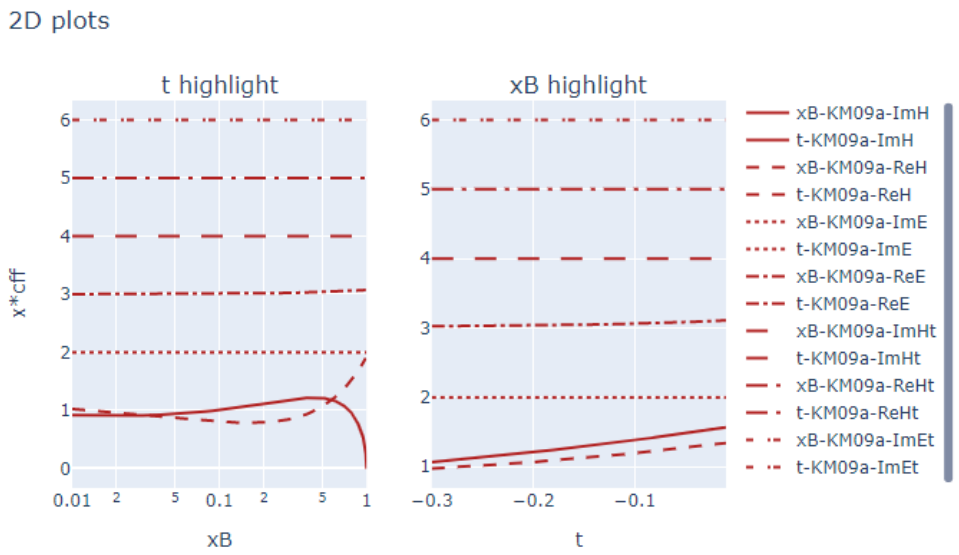
Za posljednji primjer ćemo odabrati model KM09a te svaki komptonov form faktor osim Im Ht, koji čak i kada pomnožimo sa xB odskače od ostalih te smanjuje čitkost grafa. Ostale parametre ćemo ostaviti iste kao i u prethodnim primjerima.




Slika 3.9: Treći primjer 2D prikaza sa ZOffset postavljenim na nulu

Vidimo da se za većinu komptonovih form faktora vrijednosti su oko nule te je teško raspoznati pojedine form faktore. Kako bi tome doskočili možemo koristiti `ZOffset` polje

u aplikaciji te tamo definirati razmak između pojedinih funkcija. Na sljedećoj slici je prikazana ista konfiguracija  samo je vrijednost polja `Zoffset` postavljena na 1.




Slika 3.10: Treći primjer 2D prikaza sa ZOffset postavljenim na jedan

Vidimo da je ovdje čitkost grafa puno veća te možemo lakše raspoznati oblike Comptonovih form faktora jer je nula za svaki form faktor pomaknuta za jedan prema gore.

# 4. Metodički dio

## *4.1 Projekti u nastavi*

Za ovu temu sam metodički dio odlučio fokusirati na informatičkoj strani rada a ne na fizici. Tu sam odluku donio zbog kompleksnosti fizike koja je obrađena u radu, koja nije previše pristupačna čak i na sveučilišnoj razini a kamoli za učenike u srednjim školama. Pokušaj ulaska u temu strukture protona, u više detalja od kvarkovske strukture uud, bi moglo dati učenicima više pitanja nego odgovora te ih daljnje zbuniti u već ne baš lako razumljivoj temi. Tako nam ostaje opcija da u ovom dijelu rada prođemo kroz te osnove nuklearne fizike koji su donekle primjereni srednješkolskim uzrastima. To bi na neku ruku bilo izmišljanje tople vode te se ne bi previše dotakli teme rada. Naravno to ne bih bila loša opcija ali nama se ovdje javlja jedna zanimljiva alternativa, možemo bazirati ovaj dio rada na izradi aplikacije. Ovdje nam na ruku dolazi sama priroda Dash aplikacijski okvir koji je u potpunosti implementiran u Python-u, a Python je programski jezik koji se obrađuje u većini srednjih škola. Naravno za izradu aplikacije potrebno je i malo znanja iz HTML-a, što nije gradivo koje je predviđeno u trenutnom kurikulumu ali je i sama izrada web stranice pogodna tema za učenički projekt iz informatike [12]. Plan je predstaviti nekoliko načina za uvesti izradu aplikacije u srednju školu kroz projekte za učenike, pretežno maturante koji žele uložit dodatan trud kako bi stekli iskustvo koje će im sigurno biti korisno u daljnjem životu.

Rad u IT industriji se više-manje fokusira na radu u timovima na pojedinim projektima te bi bilo dobro uvesti više projekata općenito u nastavu a pogotovo u nastavu informatike. Naravno nije potrebno odmah od prvog razreda forsirati projekte. Dobro je dati učenicima vremena da se upoznaju sa nastavnikom i gradivom, a onda kasnije početi postepeno uvoditi više projekata, pogotovo za učenike koji imaju ambicije kasnije u životu uči u IT sektor. Cilj škole bi trebao i biti da učenike priprema za daljnji život, a rad na projektima će ih sigurno pripremiti ne samo za poslovni život nego i za život tokom fakulteta. Učenici se često žale da ne vide smisao u gradivu koje se obrađuje u školi. Ako primijetimo da se većina stvari koje učenici rade u školi svede na rješavanje malih odvojenih zadataka iz kojih se teško izvlači veći smisao možemo lako shvatiti iz čega proizlaze te primjedbe. Moramo proraditi na davanju nekog šireg konteksta tim manjim zadatcima kako bi učenici u njima vidjeli smisao, ali također možemo im i davati neke veće zadatke koji bi onda sami u sebi imali neku svrhu. Ti zadatci bi bili projekti kojima bi krajnji cilj bio napraviti nešto

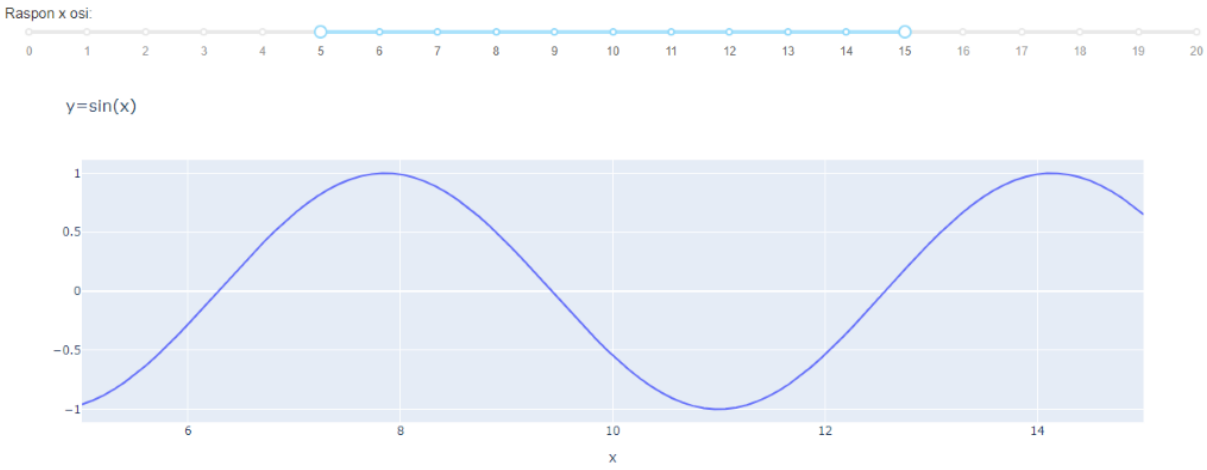
kao jednostavan program, praktička Excel tablica, multimedijski projekt, web stranica, web aplikacija, itd.. Jedino što je bitno je da nakon što se projekt izradi on ima neku primjenu tj. da učenici mogu uistinu koristiti ono što su izradili. Tako mogu direktno vidjeti svrhu svoga rada i osjetiti na vlastitoj koži kako je to izraditi nešto. Prilika za izraditi nešto korisno se ne javlja često pogotovo u današnje vrijeme konzumerizma gdje je lakše kupiti neko rješenje nego pokušati napraviti vlastito rješenje. Također se onda taj rad može prezentirati drugim učenicima što će možda te učenike motivirati da pokušaju učiniti nešto slično.

Što se tiče konkretno teme izrade web aplikacije to se trenutno jako popularna stvar u IT industriji. Mogu iz prve ruke reći da je firma u kojoj radim u zadnjih par godina odlučila sve interne aplikacije koje koristimo prebaciti na web. Time dobivaju veću kontrolu nad radom samih aplikacija, te lakše održavanje aplikacija i ispravljane grešaka. Kao krajnji privatni korisnici možda bi radije imali lokalno instalirane aplikacije zbog raznih briga o privatnosti podataka i pristupačnosti aplikacije, ali činjenica je da trenutni trendovi u software-u teže prema stalnim pristupom internetu te većina aplikacije ne funkcionira bez nekakve autentifikacije sa web serverom. A onda je mali korak od aplikacija koje ne rade bez internet veze do aplikacije koje su u potpunosti sadržane na nekoj web stranici. Tako da je definitivno dobra ideja omogućiti učenicima da skupe nešto iskustva u izradi web aplikacija.

### *4.2 Kostur projekta web aplikacije*

Zahtijevati od učenika srednjih škola da od nule naprave cijelu aplikaciju je možda malo previše ambiciozno. Naravno upoznao sam puno ljudi koji su tokom škole imali puno interesa za teme iz informatičkih područja koji su zbog nedostatka izazova tokom nastave u svoje slobodno vrijeme obradili puno naprednog gradiva čime su stekli dosta znanja, a neki su čak i napravili dosta impresivne projekte bez direktne pomoći bilo kakvih nastavnika. Ali takvu predanost očekivati od većine učenika pogotovo kad uzmemo u obzir koliko različitih predmeta u današnje vrijeme učenici slušaju u školi, te naravno koliko vremena učenici viših razreda utroše na pripremanje za državnu maturu. Kako bi učinili projekt što bezbolnijim za učenike, čime bi smanjili teret na učenicima i potencijalno privukli više učenika, pametno bi bilo napraviti takozvani kostur aplikacije koji u sebi sadrži jasno označene cjeline aplikacije. Time bi se učenik koji izrađuje aplikaciju lakše snašao na početku izrade aplikacije, pogotovo kad mu jasno označimo čime služe pojedini dijelovi aplikacije te mu damo poneki primjer što se može koristiti u svakom dijelu. Ovaj bi kostur također trebao sadržavati

osnovne import funkcije za biblioteke koje će učenici koristiti tokom izrade aplikacije, s time da bi učenici naravno mogli slobodno dodavati biblioteke koje bi im možda bile potrebne. Osim toga nastavnik bi se trebao pobrinuti za server na kojemu bi se aplikacija izvršavala i samim postavljanjem aplikacije na server, ako je to naravno moguće s obzirom na resurse kojima nastavnik raspolaže. Ako ne možemo postaviti aplikaciju na web server možemo je samo lokalno pokrenuti te je onda na taj način vrednovati. Učenicima treba također pokazati kako da tokom izrade aplikacije sami pokreću aplikaciju lokalno na svojim računalima što je neophodno za testiranje rada same aplikacije. Kako bi to testiranje bilo što lakše za učenike naš kostur bi se također trebao sam po sebi moći pokrenuti lokalno bez ikakvih grešaka. Ako je kostur takav onda bi učenici mogli biti sigurni da je svaki problem s aplikacijom posljedica njihovog rada što će im olakšati rješavanja tih potencijalnih problema. Da bi se pobrinuli da kostur radi bilo bi dobro da on ima neku jednostavnu funkcionalnost. Idealno bi bilo da kostur ima jedan element u kojemu će se odabrati neki ulazni parametar, jedan element u kojem će se ispisivati rezultat tj. crtati graf, te jednostavnu povratnu funkciju koja će pomoću ulaznog parametra nešto napraviti ili nacrtati te to vratiti u aplikaciju.




Slika 4.1: Kostur aplikacije

Kao primjer kostura napravio sam aplikaciju koja u sebi sadrži jedan `dcc.RangeSlider` u kojemu biramo raspon x osi, `dcc.Graph` u kojemu se crta graf, te povratnu funkciju koja će uzeti taj raspon x osi i vratiti aplikaciji `go.Scatter` graf gdje će x os biti definirana odabranim rasponom a y os će biti sin(x). Ovdje za računanje koristimo funkcije iz numpy biblioteke pa ju naravno na početku koda učitamo zajedno sa osnovnim bibliotekama Dash aplikacijskog okvira i Plotly-a. Također sam se pobrinuo da će se aplikacija sama pokrenuti lokalno da bilo kojem računalu kako bi učenicima olakšao daljnje

testiranje kada krenu raditi svoje promjene. Postoji još jedna linija koda kojom uvedemo standardnu Dash CSS time definiramo izgled sami elemenata aplikacije te se onda učenici ne trebaju zamarati ručnim uređivanjem izgleda pojedinih elemenata koje će koristiti u svojim aplikacijama. Naravno da ćemo zajedno sa učenicima koji će biti zainteresirani za projekt proći ovaj kostur i objasniti im čemu sve točno služi te im odgovoriti na pitanja koja bi eventualno imali. Ali je također pametno direktno u kod staviti komentare koji bi mogli služiti kao posjetnik učenicima tokom izrade aplikacije. U tim komentarima smo također stavili linkove za web stranice na kojima učenici mogu pronaći više informacija o raznim funkcijama i mogućnostima Dash-a i Plotly-a. Linkovi u komentarima su [9] [10] [11].

### *4.3 Projekt za jednog učenika*

Situacija koja se često zna pojaviti u razredu je da jedan učenik savladava gradivo puno brže od ostatka razreda. U tom slučaju potrebno je prilagoditi izvođenje nastave kako bi taj učenik također imao neki izazov za savladati tokom nastave. Ako to ne učinimo moglo bi se dogoditi da taj učenik počne stagnirati zbog dosade te ne dostigne svoj puni potencijal. Također bi prisutnost takvog učenika mogla biti demotivirajuća za ostale učenike, pogotovo one koji su jako natjecateljski nastrojeni ako zaključe da se, unatoč svojeg uloženog truda, nikad neće moći usporediti sa tim učenikom. Još jedan problem koji se može javiti je da taj učenik na neki iskrivi percepciju cijelog razreda sa stajališta nastavnika. Moguće je da nastavnik dobi ne realnu sliku svojih predavačkih metoda, gdje zbog uspjeha jednog učenika bi se moglo činiti da metode dobro funkcioniraju kada ustvari te metode nisu pogodne za veliku većinu razreda. Također bi se mogla stvori percepcija od strane nastavnika da je ostatak razreda lijen i ne zainteresiran što bi isto moglo stvoriti probleme u izvođenju nastave. Kako bi dobili realniju sliku razini znanja u razredu trebali bi, slično kao i kada u proizvoljnom mjerenju dobijemo jedan rezultat koji jako odstupa od ostalih, ignorirati rezultate najboljeg učenika te bez tih rezultat formirati naše mišljenje o razini znanja i efikasnosti naših metoda predavanja.

Kako bi izbjegli sve gore navedene probleme trebamo imati pripremljene zadatke i projekte kako bi osigurali da čak i nadareni učenici imaju dovoljan izazov tokom nastave kako bi poboljšali svoje znanje i sposobnosti. U idealnoj situaciji kada jednom razredu predajemo sve četiri godine srednje škole mogli bi imati pripremljene zadatke za učenike prvih i drugih razreda, a onda za učenike trećih i četverih razreda koje bolje poznajemo možemo pripremiti i neke samostalnije projekte. Ti projekti mogu biti raznih vrsta te bi jedan

primjer biti da u trećem razredu učenik izradi svoju web stranicu, a onda u četvrtom napravi i web aplikaciju koji bi mogao postaviti na svoju web stranicu. A tu bi onda koristio Dash aplikacijski okvir koji je korišten i u ovome radu koji bi bio jako pristupačan nekome tko već poznaje Python programski jezik.

Ideja bih bila da učenik sam odabere neku temu koja ga zanima, bilo to sport, razno razni hobiji ili nešto treće. Na tu temu bi onda uz razgovor sa nastavnikom došao do nečega što se može na zanimljiv način prikazati pomoću grafova. To ne bi trebalo biti problem pošto se u današnje vrijeme sve više-manje vrti oko prikupljanja podataka a ti podatci se onda daju prikazati na puno zanimljivih načina. Kada učenik odabere svoju temu možemo mu dati nekoliko mali vježbi koje bi ga upoznale sa nekim svojstvima Dash aplikacijskog okvira te ga možemo usmjeriti na web stranice Plotly firme gdje se može upoznati sa dodatnim mogućnostima koje bi mogao implementirati u svoju web aplikaciju. Također mu možemo pokazati primjer neke web aplikacije koju smo mi napravili te direktno proći kroz aplikaciju sa učenikom kako bi razjasnili pojedina pitanja koja bi se mogla pojaviti. Nakon toga damo učeniku slobodu da iskaže svoju kreativnost u izradi aplikacije, naravno pomno prateći učenikov napredak, te uvijek biti na raspolaganju za bilo kakva pitanja koja bi mogao imati. Ideja je da bi učenik radio na aplikaciji tokom vježbi u školi nakon što pokaže da vlada gradivom koji ostali razred obrađuje. Moguće je i da učenik doma radi na aplikaciji a vrijeme tokom vježbi koristi više za konzultacije sa nastavnikom gdje bi iskušavao ideje i rješavao potencijalne nejasnoće koje bi se mogle pojaviti. Naravno potrebno je osigurati da učenik za svoj trud bude nagrađen, prirodu nagrade bi naravno trebao odrediti sam nastavnik uz dogovora sa razrednikom i roditeljima učenika, kako se ne bi pojavili nepotrebni nesporazumi.

### *4.4 Projekt za parove učenika*

Ponekad je dobro ponuditi učenicima mogućnost rada na dodatnim projektima kako bi proširili svoje znanje i vještine, te bi izrada web aplikacije u Dash aplikacijskom okviru bio dobar projekt za parove učenika zbog prirode samih aplikacija. Aplikacija je podijeljena u dva samostalne ali povezane cjeline te bi onda svaki učenik u paru mogao biti zadužen za jedan od ta dva dijela. Prvi učenik bi se mogao baviti dizajnom izgleda same aplikacije u prvoj cjelini aplikacije. Tu bi pomoću takozvanih Dash Core Components (dcc) funkcija, koje u sebi sadrže unaprijed napravljene elemente sučelja poput polja za upis, padajućih izbornika, elementa za prikaz grafova itd., učenik definirao mogućnosti aplikacije. Učenik

bi također pomoću raznih funkcija iz html biblioteke dodatno podešavao izgled same aplikacije, te funkcije su preslika funkcionalnosti podešavanja izgleda web stranice iz HTML-a što bi moglo biti poznato učeniku ako je možda napravio neku web stranicu tokom prijašnjih projekata. Drugi učenik bi se onda pozabavio radom na povratnoj funkciji aplikacije gdje bi pomoću znanja Pythona osigurao da aplikacija funkcionira na željeni način. Tu bi se koristile razne funkcije iz Plotly, Graphical Objects, go biblioteke kako bi se podesio prikaz podataka na željeni način. Ovdje nema potrebe za nikakvim predznanjem HTLM-a, ali će biti potrebno bolje vladanje Pythonom nego u prvom dijelu aplikacije. Vidimo da ta dva dijela aplikacije imaju različite izazove te nije potrebno da oba učenika u paru imaju velike vještine programiranja. Prva cjelina aplikacije je idealna za učenike koje možda imaju želju raditi na projektu ali možda nisu najbolju u programiranju. Ovime možemo aktivirati učenike čije se vještine možda ne ističu tokom standardnih provjera znanja iz informatike. Dok druga cjelina može biti izazov za učenike koji s lakoćom savladaju probleme iz programiranja tokom redovne nastave.

Naravno ovdje bi bilo dobro unaprijed pripremiti teme o kojima će učenici raditi aplikaciju, jer bi trebali očekivati da će se dva učenika teže dogovoriti oko teme bez prethodnih prijedloga. S tim da bi uvijek prihvatili bilo koju temu oko koje bi se par učenika mogao dogovoriti. Moramo osigurati da naše teme budu dovoljno fleksibilne kako učenici mogu iskazati svoju kreativnost, ali također moramo postaviti određene ciljeve, tj. svojstva koja bi aplikacija trebala imati kako bi osigurali dovoljan izazov za učenike. Time bi pokušali izbjeći situaciju u kojoj učenici tek toliko nešto naprave bez da su uložili trud i pokazali doseg svojih znanja i vještina. Na kraju je uvijek ideja da nagradimo učenike za njihov trud i rad pa moramo pripremiti prikladne nagrade za učenike.

### *4.5 Projekt za cijeli razred ili grupu učenika*

Ponekad se može dogoditi situacija da cijeli razred ili možda jedna grupa u razredu, ako je nastava organizirana u grupama, bude jako zainteresirana za gradivo informatike. Tada bi se onda cijeloj grupi mogao dati projektni zadatak, a u četvrtom razredu nakon već odrađenih projekata u nižim razredima bi im mogli i zadati zadatak izrade web aplikacije. Taj bi projekt onda odrađivali tokom vježbi iz informatike kada već cijela grupa tj. razred sudjeluje u njemu. Započeli bi sa kratkim predavanje o Dash-u i općenito cilju ovog projekta nakon čega bi učenici sami uz vođenje nastavnika pokušali napraviti aplikaciju.

Kada bi se odlučili takav projekt sa cijelim razredom morali bi prvo odlučiti kako će učenici taj projekt odraditi, individualno ili u manjim grupama tj. parovima. Također bi bilo pametno možda prezentirati ovaj projekt kao neki zadatak koji bi onda cijela grupa pokušala riješiti. Razlika od standardnih projekata bi bila u tome da unaprijed definiramo kako bi aplikacija trebala izgledati i što bi točno trebala moći raditi. Time bi opet skratili vrijeme koje bi utrošili na ovu aktivnost, a učenici bi se bolje snašli kada bi imali čvrstu sliku kako bi aplikacija trebala izgledati. U svrhu toga bilo bi dobro osim standardnog kostura kojeg bi dali učenicima pripremiti i funkcionalnu verziju aplikacije koju bi učenicima pokazali kada bi im predstavljali zadatak, bez da im pokažemo kod naravno. Onda bi nakon što završe svoju aplikaciju mogli usporediti njihov kod sa našim i napraviti analizu kod te vidjeti u čemu su sličnosti a u čemu razlike.

Ako je razred manji ili se radi o grupi učenika mogli bi razmišljati o individualnim projektima, ali to bi onda sigurno oduzelo dosta vremena i ne čini se kao razuman trošak već ograničenog vremena kojeg imamo u razredu. Kako bi što bolje iskoristili vrijeme koje imamo najbolje bi bilo rasporediti učenike u manje grupe od dvoje do četiri učenika po svakoj grupi. Onda bi svakom učeniku u grupi odredili njegov dio posla, podjela bi bila slična kao i kada imamo par učenika. Dio grupe bi se bavio izgledom aplikacije pomoću dcc funkcija, a drugi dio bi se bavio povratnom funkcijom tj. radom aplikacije. Pošto bi sada više učenika radilo na svakom dijelu aplikacije oni koji rade na prvom dijelu bi mogli brže pronaći točno koje dcc funkcije trebaju te kako bi ih implementirali da aplikacije izgleda što bolje. Dok bi oni koji rade na drugom dijelu mogli podijelit dijelove posla u povratnoj funkciji tako da se netko bavi povratnom funkcijom da netko drugi izradi pomoćne funkcije kojima će aplikacija raditi kako želimo.

### *4.6 Priprema za sat*

Kako bi opisali sat u kojem radimo projekt sa cijelim razredom napraviti ćemo pripremu za sat. Ovaj priprema je predviđena za barem blok sat nastave, idealno ako to raspored omogućava, bi možda bilo čak i tri sata u komadu. Tema je predviđena za obradu samo u jako naprednim razredima, tj. grupama učenika, te bi spadala nastavnu cjelinu programiranja kao dodatno gradivo. Oblici rada su frontalni rad i rad u grupama. Glavna nastavna metoda je kooperativno rješavanje zadataka, u ovom slučaju rješavanje jednog zadatka u manjim grupama učenika, također koristimo i metodu pisanja i razgovora te

metodu demonstracije. Nastavna pomagala su kompjutori, te ploča i kreda. Korelacija postoji sa matematikom zbog prirode samog zadatka.

- *Obrazovni ishodi*: navesti osnovnu građu web aplikacije, opisati razliku između web i desktop aplikacije, primijeniti svoje znanje programiranja u složenom zadatku, razvijati sposobnost logičkog razmišljanja, razvijati sposobnost timskog rada
- *Odgojni ishodi*: surađivanje u timskom radu, dogovorna podjela posla u timskom radu, uvažavanje tuđeg mišljenja, izražavanje vlastitog mišljenja, razvijanje sistematičnosti i urednosti u pisanju koda.

### *4.6.1 Uvodni dio*

Sat će unaprijed biti dogovoren sa učenicima zbog svoje prirode kao dodatno gradivo. Sat ćemo započeti frontalnim predavanjem o Dash aplikacijskom okviru i Plotly bibliotekama koje ćemo koristiti za izradu aplikacije. Glavna pitanja koja želimo postaviti učenicima su:

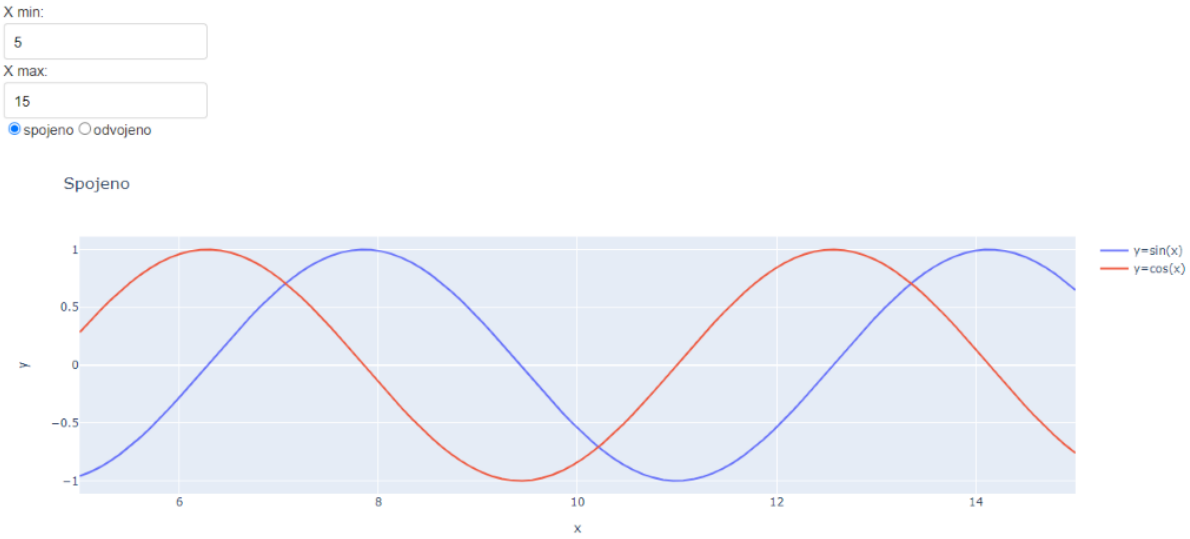
- Koja je razlika između web aplikacije i standardne aplikacije?
- Koje dijelove očekujete da web aplikacija ima?

Odgovori koje želimo dobiti bi trebali voditi prema tome da se web aplikacije izvode na serveru a ne lokalno i da aplikacije ima dva dijela, jedan dio za definiranje izgleda aplikacije i jedan dio koji definira funkcionalnost aplikacije.

Nakon toga ćemo učenike upoznati sa kosturom aplikacije koji će oni koristiti za izradu svoje web aplikacije. Pokazati ćemo učenicima direktno na kosturu koji su točno dijelovi aplikacije, te im pojasniti točno način na koji će raditi na aplikaciji. Objasnit ćemo im da će biti podijeljeni u grupe, a u svakoj će grupi netko biti zadužen za određeni dio aplikacije. Nakon toga se učenici mogu podijeliti u grupe te ćemo krenuti na rješavanje glavnog zadatka. Uvodni dio bi trebao trajati otprilike 30 minuta.

### *4.6.1 Središnji dio*

U središnjem dijelu sata učenici će u grupama rješavati zadatak u kojemu će morati modificirati kostur aplikacije koji ćemo im mi prirediti kako bi dobili aplikaciju koja izgleda kao primjer na slici 4.2. Ovaj dio sata će trajati ostatak dostupnog vremena s tim da trebamo ostaviti 10 do 15 minuta za završni dio.




Slika 4.2: Primjer zadatka

Zadatak za učenike je napraviti sljedeće promjene na kosturu aplikacije:

- Odabir raspona vrijednosti na x osi tako da unesemo odvojeno maksimalne i minimalne vrijednosti raspona.
- Na grafu će se crtati sin(x) i cos(x) funkcija
- Aplikacija će sadržavati mogućnost odabira dali se sin(x) i cos(x) crtaju zajedno na jednom grafu ili odvojeno na dva zasebna grafa
- Potrebno je označiti sve osi grafova te dati naslove svim funkcijama i grafovima

Naravno potrebno je procijeniti ima li dovoljno vremena za odraditi sve dijelove zadatka, ako nema možemo neke dijelove izbaciti. Svakako bi trebali prije nego što učenici krenu raditi aplikaciju demonstrirati naš gotov primjer bez da im pokažemo kod kako bi učenici dobili osjećaj za funkcionalnost aplikacije.

Kako bi učenici u potpunosti riješili zadatak morat će napraviti slijedeće: Moraju izbrisati `dcc.RangeSlider` funkciju te umjesto nje staviti dvije `dcc.Input` funkcije jednu za upis minimalne vrijednosti raspona i jednu za upis maksimalne vrijednosti raspona. Također će morati staviti jednu `dcc.RadioItem` funkciju gdje će morati definirati dvije opcije kojima bi se biralo dali se grafovi crtaju skupa ili odvojeno. Kako bi se to implementiralo u povratnoj funkciji potrebno je napraviti nekoliko promjena. Prvo se trebaju definirati novi inputi za povratu funkciju, po jedan za svaki novi dcc element. To će postići tako da više Input funkcija, koje odvojimo zarezima, stavimo u jednu uglatu zagradu. Nakon čega se učenici moraju pobrinuti da su varijable u povratnoj funkciji u istom redoslijedu kao

i u zagradi sa input funkcijama. Također moramo modificirati `np.linspace` funkciju da više ne uzima vrijednosti iz raspona kao granice nego da uzima vrijednosti koje su unesene preko dvije `dcc.Input` funkcije. Sada bi mogli napraviti jedno if grananje koje provjerava koja je opcija odabrana za način crtanja grafa. Ako želimo da se crtaju zajedno onda mogu iskoristi kod iz kostura samo moraju u uglatu zagradu pokraj `go.Scatter` funkcije za sinus dodat još jednu `go.Scatter` funkciju za kosinus. Također će učenici trebati dati naziv svakoj do tih funkcija kako bi ih lakše mogli razlikovati. Još je potrebno dodati naslov za y os koji nije u kosturu, te promijeniti naziv tog grafa. Za drugu if granu je potrebno napraviti subplot koji ima jedan red i dva stupca. Za to će morati učitat `make_subplots` iz `plotly.subplots`. Ovdje će imati priliku malo istražiti mogućnosti Plotly-a i uz vođenje nastavnika doći do rješenja. Nakon što pronađu što im trebat će dodati svaku `go.Scatter` funkciju u jedan subplot te im dati naziv. Također će trebati dati naziv svakoj osi grafova, te cijelom grafu.

### *4.6.1 Završni dio*

Nakon što učenici završe sa zadatkom napraviti ćemo kratki osvrt na aplikacije koje su učenici napravili, te ih usporediti sa našom aplikacijom. Proći ćemo sa učenicima naš kod te komentirati eventualne razlike koje postoje između našeg koda i koda učenika. Pitat ćemo učenika s kojim dijelovima zadatka su imali najviše problema, te ako je nešto uzrokovalo puno problema možemo ubuduće to promijeniti u zadatku. Na kraju sata pitamo učenike što misle o izradi web aplikacija, te vide li možda u tome potencijalnu karijeru u budućnosti.

## 5. Zaključak

Za početak rada bavili smo se pitanjem što je to točno struktura protona. Protoni su veoma male čestice koje ne možemo samo tako rastaviti i pogledati njihovu unutarnju strukturu. Po standardnom modelu proton se sastoji od tri kvarka, dva up kvarka i jednog down kvarka, ali je njegova stvarna struktura dosta složenija te se ona još uvijek istražuje u mnogim istraživanjima diljem svijeta. Istraživanja se provode sudaranjem raznih čestica te snimanjem rezultata sudara, ti se rezultati onda obrađuju te se pomoću njih konstruiraju matematički modeli koji bi mogli opisati potpunu strukturu protona. Fizikalni eksperiment koji je nama bio najzanimljiviji je duboko virtualno komptonovo raspršenje, razlog tome je to što smo koristili takozvane KM modele strukture protona. Oni u sebi sadrže komptonove form faktore koji opisuju to raspršenje a posljedično toma opisuju i samu strukturu protona koji sudjeluje u tom raspršenju

Cilj ovog rada bio je izrada aplikacije koja će služiti kao sučelje za pristup strukturi protona. Taj cilj je bio uspješno ostvaren  izradom aplikacije koja je opisana u trećem poglavlju ovog rada. Aplikacija pristupa KM modelima strukture protona putem vanjskog cffs.py modula te ih pomoću internih funkcija crta u 3D ili 2D prostoru. Sama aplikacija je realizirana pomoću Dash aplikacijskog okvira i Plotly grafičke biblioteke koja je i stvorena od strane Dash korporacije. Plotly grafička biblioteka je i glavni razlog odabira Dash aplikacijskog okvira zbog svojih mogućnosti prikaza funkcije u 3D prostoru. Jedan od glavnih funkcionalnosti koju smo željeli imati je bila mogućnost prikaza presjeka 3D plohe pomoću 2D funkcija u 3D prostoru, te mogućnost pomicanja tih funkcija po osima. Implementacija toga je bila vrlo jednostavna zbog mogućnosti Plotly grafičke biblioteke.

Jednostavnost Plotly-a mi je dala inspiraciju za metodički dio rada gdje sam koncipirao način da pomoću Dash-a i Plotly-a omogućimo učenicima u srednjim školama da stvore svoju aplikaciju. Ideja je da općenito uvedemo više projektnih zadataka u škole jer skoro svi poslovi u IT industriju se zasnivaju na projektima, te bi više takvog iskustva dobro došlo svima a pogotovo učenicima.

**Dodatak**

**A Kod finalne aplikacije**

```
import sys
sys.path.append('/home/gepard/cffs4mc/pype')
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objects as go
from dash.dependencies import Input, Output
import numpy as np
from plotly.subplots import make_subplots
import cffs
external_stylesheets
=['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__,
external_stylesheets=external_stylesheets)
#app.config.update({'requests_pathname_prefix':'/alpha/'})
server = app.server

Q2 = 4

def gcff3D(xB, t, ModelID, cff):
    r = np.empty((len(xB),len(t)))
    i = 0
    for y in t:
        j = 0
        for x in xB:
            rij=cffs.getcff(x,y,Q2,ModelID,cff)
            #if(rij>20):
             #   r[i][j]=(20)
            #elif(rij<-20):
             #   r[i][j]=(-20)
            #else:
            r[i][j]=(x*rij)
            j+=1
        i+=1

    return r

def gcff2D(xB, t, ModelID, cff):
    if type(xB)is np.ndarray:
        r = np.empty(len(xB))
        i=0
        for x in xB:
```

```
            ri=cffs.getcff(x,t,Q2,ModelID,cff)
            #if(ri>20):
             #   r[i]=(20)
            #elif(ri<-20):
             #   r[i]=(-20)
            #else:
            r[i]=(x*ri)
            i+=1
        return r
    if type(t)is  np.ndarray:
        r = np.empty(len(t))
        i=0
        for y in t:
            ri=cffs.getcff(xB,y,Q2,ModelID,cff)
            #if(ri>20):
             #   r[i]=(20)
            #elif(ri<-20):
             #   r[i]=(-20)
            #else:
            r[i]=(xB*ri)
            i+=1
        return r



ModelID = 'KM09a',
cff = 'ImH',
Zoff = 0

xB = np.linspace(0.01,1,50)
t = np.linspace(-0.01,-0.03,50)

xBhR = np.linspace(0.5,0.5,50)
thR = np.linspace(-0.2,-0.2,50)




data = list()
j=0
for m in ModelID:
    for c in cff:
        data.append(go.Scatter3d(
        x = xBhR,
        y = t,
```

```
        z = gcff2D(0.5,t,m,c)+j*Zoff,
        name=m+'-'+c+'-x-highlight',
        mode = 'lines',
        showlegend=not(1),
        line=dict(
        color='darkgreen',
        width=5)
        ))

        data.append(go.Scatter3d(
        x = xB,
        y = thR,
        z = gcff2D(xB,-0.2,m,c)+j*Zoff,
        name=m+'-'+c+'-y-highlight',
        mode = 'lines',
        showlegend=not(1),
        line=dict(
        color='aquamarine',
        width=5)
        ))

        data.append(go.Surface(
        x = xB,
        y = t,
        z = gcff3D(xB,t,m,c)+j*Zoff,
        showscale=not(1),
        name = m+'-'+c,
        opacity=0.7
        ))

        j+=1


fig = go.Figure(data)

fig.update_traces(hoverinfo='name')

fig.update_layout(
    title_text="3D plot",
    scene = {
    "xaxis": {"nticks": 20,"title": "Xb","type": "linear",},
    "yaxis": {"nticks": 20,"title": "t","type": "linear",},
    "zaxis": {"nticks": 20,"title": "x*cff",},
    "aspectratio": {"x": 1, "y": 1, "z": 1}
    },
```

```
    )

fig.layout.uirevision = True



app.layout = html.Table([
    html.Tr([
    html.Td([
    html.Tr([
        html.Div([
        html.Td([html.Th(children='App 1.0')]),
        ], style={"display": "grid", "grid-template-columns":
"60%"})
        ]),
    html.Tr([
        html.Div([
        html.Td([html.Label('Odabir modela:'),
                 dcc.Dropdown(
                options=[
                {'label': 'KM09a', 'value': 'KM09a'},
                {'label': 'KM09b', 'value': 'KM09b'},
                {'label': 'KM10', 'value': 'KM10'},
                {'label': 'KM10a', 'value': 'KM10a'},
                {'label': 'KM10b', 'value': 'KM10b'},
                {'label': 'KMM12', 'value': 'KMM12'},
                {'label': 'KM15', 'value': 'KM15'}
                ],
                id='Mselect',
                value=['KM09a'],
                multi=True,
                style={'width': 200}
                )
                ]),
        html.Td([html.Label('Odabir cff:'),
                dcc.Dropdown(
                options=[
                {'label': 'Im H', 'value': 'ImH'},
                {'label': 'Re H', 'value': 'ReH'},
                {'label': 'Im E', 'value': 'ImE'},
                {'label': 'Re E', 'value': 'ReE'},
                {'label': 'Im Ht', 'value': 'ImHt'},
                {'label': 'Re Ht', 'value': 'ReHt'},
                {'label': 'Im Et', 'value': 'ImEt'},
                {'label': 'Re Et', 'value': 'ReEt'}
```

```
                ],
                id='CFFselect',
                value=['ImH'],
                multi=True,
                style={'width': 200}
                )
        ])
        ], style={"display": "grid", "grid-template-columns": "50%
50%"})
        ]),
    html.Tr([
        html.Div([
        html.Td([html.Label('Xb highlght:'),
         dcc.Input(
                id='Xbhl',
                type='number',
                value=0.5,
                step=0.01,
                style={'width': 80}
                )
        ]),
        html.Td([html.Label('Xb min:'),
         dcc.Input(
                id='Xbmin',
                type='number',
                value=0.01,
                step=0.01,
                style={'width': 80}
                )
        ]),
        html.Td([html.Label('Xb max:'),
         dcc.Input(
                id='Xbmax',
                type='number',
                value=1,
                step=0.01,
                style={'width': 80}
                )
        ])
        ], style={"display": "grid", "grid-template-columns": "30%
30% 30%"})
        ]),
    html.Tr([
        html.Div([
        html.Td([html.Label('t highlght: '),
```

```
         dcc.Input(
                id='thl',
                type='number',
                value=-0.2,
                step=0.01,
                style={'width': 80}
                )
        ]),
        html.Td([html.Label('t min: '),
         dcc.Input(
                id='tmin',
                type='number',
                value=-0.3,
                step=0.01,
                style={'width': 80}
                )
        ]),
        html.Td([html.Label('t max:'),
         dcc.Input(
                id='tmax',
                type='number',
                value=-0.01,
                step=0.01,
                style={'width': 80}
                )
        ])
        ], style={"display": "grid", "grid-template-columns": "30%
30% 30%"})
        ]),
    html.Tr([
        html.Div([
        html.Td([html.Label('Prikaz: '),
            dcc.RadioItems(
                    id='g-type',
                    options=[
                        {'label': '3D', 'value': '3d'},
                        {'label': '2D', 'value': '2d'},
                    ],
                    value='3d',
                    labelStyle={'display': 'inline-block'}
        ),
            dcc.RadioItems(
                    id='a-type',
                    options=[
                        {'label': 'lin', 'value': 'linear'},
```

```
                        {'label': 'log', 'value': 'log'},
                    ],
                     value='linear',
                     labelStyle={'display': 'inline-block'}
        )
        ]),
        html.Td([
        html.Label('Z-offset: '),
        dcc.Input(
                id='z-off',
                type='number',
                value=0,
                style={'width': 80}
        )
        ])
        ], style={"display": "grid", "grid-template-columns": "30%
50%"})
        ])
        ]),
    html.Td([
        html.Div([
    dcc.Graph(id='Test-output',figure = fig)
    ],style={'width': '100%','hight': '80%', 'float': 'right'})
        ])
        ]),



    ])


@app.callback(
    Output('Test-output','figure'),
    [Input('Mselect', 'value'),
     Input('CFFselect', 'value'),
     Input('Xbmin', 'value'),
     Input('Xbmax', 'value'),
     Input('Xbhl', 'value'),
     Input('tmin', 'value'),
     Input('tmax', 'value'),
     Input('thl', 'value'),
     Input('g-type', 'value'),
     Input('a-type', 'value'),
     Input('z-off', 'value')])
```

```
def update_graph(ModelID, cff, xBmin, xBmax, xBh, tmin, tmax, th,
Gtype, Atype, Zoff):

    if Gtype == '2d':

        xB = np.linspace(xBmin,xBmax,100)
        t = np.linspace(tmin,tmax,100)

        xBhR = np.linspace(xBh,xBh,100)
        thR = np.linspace(th,th,100)

        fig = make_subplots(
            rows=1, cols=2,
            subplot_titles=("t highlight","xB highlight"))

        k =
('firebrick','royalblue','darkgreen','gray','aqua','hotpink','dark
orange')
        h =
('solid','dash','dot','dashdot','longdash','longdashdot','5px,10px
,2px,2px','2px,5px,5px,10px')

        o=0
        n=0
        for m in ModelID:
            j=0
            for c in cff:
                fig.add_trace(go.Scatter(x=xB,
y=gcff2D(xB,th,m,c)+(o)*Zoff, name='xB'+'-'+m+'-'+c, mode =
'lines', line = dict(color= k[n], dash = h[j])), row=1, col=1)
                fig.add_trace(go.Scatter(x=t,
y=gcff2D(xBh,t,m,c)+(o)*Zoff, name='t'+'-'+m+'-'+c, mode =
'lines', line = dict(color= k[n], dash = h[j])), row=1, col=2)
                j+=1
                o+=1
            n+=1
        fig.update_xaxes(title_text="xB", type=Atype, row=1,
col=1)
        fig.update_xaxes(title_text="t", row=1, col=2)

        fig.update_yaxes(title_text="x*cff",
                         row=1, col=1)
        fig.update_yaxes(title_text="",
                         row=1, col=2)

        fig.update_layout(title_text="2D plots")
```

```
    fig.layout.uirevision = True

    return fig

else:

    xB = np.linspace(xBmin,xBmax,30)
    t = np.linspace(tmin,tmax,30)

    xBhR = np.linspace(xBh,xBh,30)
    thR = np.linspace(th,th,30)

    data = list()
    j=0
    for m in ModelID:
        for c in cff:
            data.append(go.Scatter3d(
            x = xBhR,
            y = t,
            z = gcff2D(xBh,t,m,c)+j*Zoff,
            name=m+'-'+c+'-x-highlight',
            mode = 'lines',
            showlegend=not(1),
            line=dict(
            color='darkgreen',
            width=5)
            ))

            data.append(go.Scatter3d(
            x = xB,
            y = thR,
            z = gcff2D(xB,th,m,c)+j*Zoff,
            name=m+'-'+c+'-y-highlight',
            mode = 'lines',
            showlegend=not(1),
            line=dict(
            color='aquamarine',
            width=5)
            ))

            data.append(go.Surface(
            x = xB,
            y = t,
            z = gcff3D(xB,t,m,c)+j*Zoff,
            showscale=not(1),
```

```
                name = m+'-'+c,
                opacity=0.7
                ))

                j+=1

        fig = go.Figure(data)

        fig.update_traces(hoverinfo='name')

        fig.update_layout(
            title_text="3D plot",
            scene = {
            "xaxis": {"nticks": 20,"title": "Xb","type": Atype,},
            "yaxis": {"nticks": 20,"title": "t",},
            "zaxis": {"nticks": 20,"title": "x*cff",},
                "aspectratio": {"x": 1, "y": 1, "z": 1}
            },
        )

        fig.layout.uirevision = True

        return fig


if __name__ == '__main__':
    app.run_server(debug=True)
```

**B Kostur aplikacije**

```
import numpy as np
import dash
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objects as go

# linkovi za dodatnu dokumentaciju
#https://dash.plotly.com/dash-core-components
#https://dash.plotly.com/dash-html-components
#https://plotly.com/python/

external_stylesheets =
['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__,
external_stylesheets=external_stylesheets)  server = app.server
fig = go.Figure()
app.layout = html.Div([
            html.Div([
                html.Label('Raspon x osi:'),
                dcc.RangeSlider(
                id='Raspon X',
                min=0,
                max=20,
                step=0.5,
                value=[5, 15],
                marks={i: '{}'.format(i) for i in range(21)}
                ),
                ]),

            html.Div([
                dcc.Graph(id='Ispis',figure = fig)

],style={'width': '100%','hight': '80%', 'float': 'right'}) ])

#-----------------------------------------------------------------
--
@app.callback(
    Output('Ispis','figure'),
    [Input('Raspon X', 'value')])

def update_graph(X):
xr=np.linspace(X[0],X[1],100)

    fig = go.Figure([go.Scatter(x=xr,y=np.sin(xr))])
```

```
    fig.update_xaxes(title_text="x")
    fig.update_layout(title_text="y=sin(x)")

    return fig

#-----------------------------------------------------------------
--
if __name__ == '__main__':
app.run_server(debug=True)
```

**C Primjer zadatka**

```
import numpy as np
import dash
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objects as go
from plotly.subplots import make_subplots  ######

external_stylesheets =
['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__,
external_stylesheets=external_stylesheets)
server = app.server

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

fig = go.Figure()



app.layout = html.Div([

            # html.Div([
            #     html.Label('Raspon x osi:'),
            #     dcc.RangeSlider(
            #     id='Raspon X',
            #     min=0,
            #     max=20,
            #     step=0.5,
            #     value=[5, 15],
            #     marks={i: '{}'.format(i) for i in range(21)}
            #     ),
```

```
            #     ]),



             html.Div([
###
                 html.Label('X min:'),
                 dcc.Input(
                 id='X-min',
                 type='number',
                 value=5,
                 )
                 ]),

             html.Div([
                 html.Label('X max:'),
                 dcc.Input(
                 id='X-max',
                 type='number',
                 value=15,
                 )
                 ]),

             html.Div([
                 dcc.RadioItems(
                 id='prikaz',
                 options=[
                         {'label': 'spojeno', 'value': 'sp'},
                         {'label': 'odvojeno', 'value': 'od'},
                 ],
                 value='sp',
                 labelStyle={'display': 'inline-block'}
                 )
                 ]),                                                ###


             html.Div([
                 dcc.Graph(id='Ispis',figure = fig)
                 ],style={'width': '100%','hight': '80%', 'float':
'right'})
])

#-----------------------------------------------------------------
--
```

```
@app.callback(
    Output('Ispis','figure'),
    [Input('X-min', 'value'),
     Input('X-max', 'value'),           ###
     Input('prikaz', 'value')])         ###


def update_graph(Xmin,Xmax,prikaz):

    xr=np.linspace(Xmin,Xmax,100)

    if prikaz == 'sp':          ###
        fig =
go.Figure([go.Scatter(x=xr,y=np.sin(xr),name="y=sin(x)"),   ###

go.Scatter(x=xr,y=np.cos(xr),name="y=cos(x)")])  ###

        fig.update_xaxes(title_text="x")
        fig.update_yaxes(title_text="y")            #
        fig.update_layout(title_text="Spojeno")

        return fig
    if prikaz == 'od':          ###
        fig = make_subplots(                            #####
            rows=1, cols=2,
            subplot_titles=("y=sin(x)","y=cos(x)"))     #####


fig.add_trace(go.Scatter(x=xr,y=np.sin(xr),name="y=sin(x)"),row=1,
col=1)    ####

fig.add_trace(go.Scatter(x=xr,y=np.cos(xr),name="y=cos(x)"),row=1,
col=2)    ####

        fig.update_xaxes(title_text="x",row=1,col=1)    #
        fig.update_xaxes(title_text="x",row=1,col=2)    #
        fig.update_yaxes(title_text="y",row=1,col=1)    #
        fig.update_yaxes(title_text="y",row=1,col=2)    #
        fig.update_layout(title_text="Odvojeno")        #

        return fig

#-----------------------------------------------------------------
-
if __name__ == '__main__':
    app.run_server(debug=True)
```

**Literatura**


[1] B. Povh, K. Rith, C. Scholz, F. Zetsche, W. Rodejohann, *„Particles and Nuclei: An Introduction To The Physical Concepts“,* Springer, Verlag, Berlin, Heidelberg, 2015.

[2] Georges F., Deeply Virtual Compton Scattering at Jefferson Lab. Doktorski rad. Paris : Sveučilište Paris-Saclay, 2018

[3] Krešimir Kumerički, Simonetta Liuti, Hervé Moutarde, *„ GPD phenomenology and DVCS fitting “*, arXiv:1602.02763v1, 2016.

[4] H. D. Young, R. A. Freedman, *„University Physics with Modern Physics“*,13th Edition,Pearson Education Inc. , San Francisco, 2012.

[5]A.V. Belitsky, A.V. Radyushkin, *„Unraveling hadron structure with generalized parton distributions “,* arXiv:hep-ph/0504030v3, 2005.

[6] D. H. Perkins, *„Introduction to high energy physics“,* 4th edition, 2003.

[7] https://www.python.org/ 17. 6. 2020.

[8] http://www.scholarpedia.org/article/Introduction_to_Parton_Distribution_Functions 20. 6. 2020.

[9] https://dash.plotly.com/dash-core-components 18. 12. 2020.

[10] https://dash.plotly.com/dash-html-components 18. 12. 2020.

[11] https://plotly.com/python/ 18. 12. 2020.

[12] http://mzos.hr/datoteke/15-Predmetni_kurikulum-Informatika.pdf 16. 1. 2021.

[13] Slika 1.1 iz [1]

[14] Slika 1.2 iz [1]

[15] https://web.lemoyne.edu/~giunta/ea/THOMSONann.HTML 13. 11. 2020.

[16] http://large.stanford.edu/courses/2017/ph241/sivulka2/ 13. 11. 2020.

[17] http://large.stanford.edu/courses/2018/ph241/kuppermann2/ 13. 11. 2020.

[18] Slika 1.2 iz [6]

[19] https://arstechnica.com/science/2010/08/following-protons-on-a-trip-to-and-through-the-lhc/ 14. 11. 2020.

[20]https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg 13. 11. 2020.

[21] Slika 1.1 iz [2]

[22] Slika 6.1 iz [1]

[23] Slika 1.2 iz [2]

[24] Slika 7.4 iz [1]

[25] Slika 7.6a iz [1]

[26] Slika 7.6b iz [1]

[27] Slika 1.4 iz [2]

[28] Slika 9 iz [3]

[29] https://www.heroku.com/ 20. 1. 2021.

[30] https://dash.plotly.com/deployment/ 19. 1. 2021.