Razvoj jednostavnog algoritma za rekonstrukciju putanje elektrona u dvije dimenzije

Pezelj, Vana

Sveučilište u Splitu
Prirodoslovno – matematički fakultet

Razvoj jednostavnog algoritma za rekonstrukciju putanje elektrona u dvije dimenzije

Završni rad

Vana Pezelj

Split, rujan 2022.

Temeljna dokumentacijska kartica

Sveučilište u Splitu
Prirodoslovno – matematički fakultet
Odjel za fiziku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Završni rad

Razvoj jednostavnog algoritma za rekonstrukciju putanje elektrona u dvije dimenzije

Vana Pezelj

Sveučilišni preddiplomski studij Fizika

Sažetak:

U kompaktnom mionskom solenoidu (*engl. Compact Muon Solenoid* (CMS)) elektroni bivaju detektirani prolazeći kroz više slojeva silicijskih piksel detektora. Elektroni se nalaze u magnetskom polju pa je njihova putanja kružna te ju je moguće definirati znajući kroz koje je piksele elektron prošao. Putanja je definirana radijusom i središtem kružnice. Poznavajući navedeno možemo izračunati količinu gibanja elektrona koja je jednoznačno definirana radijusom. Cilj ovog rada je pronaći radijus generirane putanje elektrona te usporediti vrijednosti dobivene za putanje koje su generirane bez relativističkog i s relativističkim pristupom. Putanju generiramo numeričkim rješavanjem jednadžbe Lorentzove sile koristeći metodu Runge-Kutta 4, a radijus pronalazimo iz sjecišta putanje s koncentričnim kružnicama definiranog radijusa. Dobiveni rezultati razlikuju se za nekoliko redova veličine, pri čemu se rezultati, koji su nastali koristeći relativistički pristup, podudaraju s vrijednostima iz literature.

Ključne riječi: Runge-Kutta, elektron, putanja, radijus, detektor, relativistika

Rad sadrži: 28 stranica, 12 slika, 0 tablica, 7 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: doc. dr. sc. Toni Šćulac

Ocjenjivači: doc. dr. sc. Toni Šćulac,
doc. dr. sc. Marko Kovač,
mag. phys. Tamara Rom

Rad prihvaćen: 12. 9. 2022.

Rad je pohranjen u Knjižnici Prirodoslovno – matematičkog fakulteta, Sveučilišta u Splitu.

Basic documentation card

University of Split
Faculty of Science
Department of Physics
Ruđera Boškovića 33, 21000 Split, Croatia

Bachelor thesis

Development of a simple electron trajectory reconstruction algorithm in two dimensions

Vana Pezelj

University undergraduate study programme Physics

Abstract:

In a Compact Muon Solenoid (CMS) electrons are detected by passing through multiple layers of silicon pixel detectors. Electrons are found in the magnetic field, so their path is circular and it is possible to define it by knowing which pixels the electron passed through. The path is defined by the radius and center of the circle. Knowing the above, we can calculate the momentum of electron, which is uniquely determined by the radius. The goal of this thesis is to find the radius of the generated electron path and compare the values obtained for trajectories generated without relativistic and with relativistic approach. We generate the trajectory by numerically solving the Lorentz force equation using the Runge-Kutta 4 method and the radius is found from the intersection of the path with concentric circles of a defined radius. The obtained results differ by several orders of magnitude, where the results, obtained using the relativistic approach, coincide with the values from the literature.

Keywords: Runge-Kutta, electron, trajectory, radius, detector, relativistics

Thesis consists of: 28 pages, 12 figures, 0 tables, 7 references. Original language: Croatian.

Supervisor: Assist. Prof. Dr. Toni Šćulac

Reviewers: Assist. Prof. Dr. Toni Šćulac,
Assist. Prof. Dr. Marko Kovač,
mag. phys. Tamara Rom

Thesis accepted: September 12, 2022

Thesis is deposited in the library of the Faculty of Science, University of Split.

Sadržaj

1 Uvod

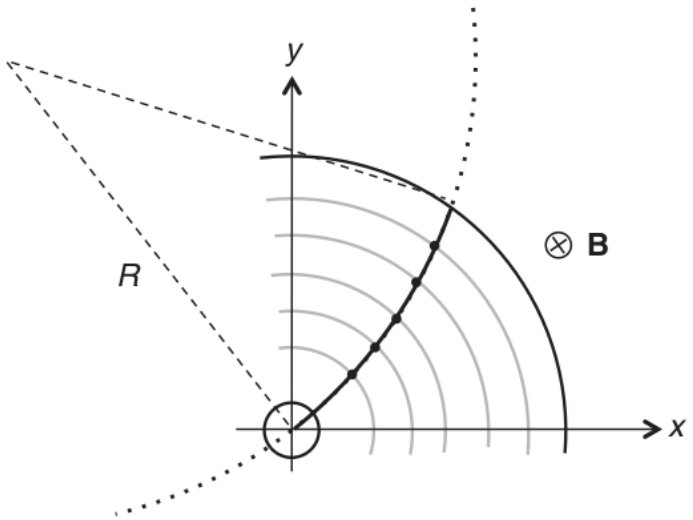
"*Zašto su lego kockice najgenijalnija igračka na svijetu?*" pitanje je koje profesor filozofije postavlja Sofiji kako bi ju naveo na razmišljanje o samoj srži tvari te joj pojasnio Demokritovu hipotezu.[1] Grčki filozof Demokrit (oko 460. - 370. g. pr. Kr) postavio je tezu koja kaže da se sva materija sastoji od nepromjenjivih jediničnih elemenata koje naziva *atomi*. Smatrao je da vidljivu materiju dobivamo spajanjem mnoštva atoma u cjeline [1], poput lego kockica koje spajamo u različite oblike. Danas, gotovo dvadeset i pet stoljeća kasnije teorijski je objašnjeno, a eksperimentalno dokazano da se sve sastoji od skupa elementarnih čestica, dakle materija doista jest djeljiva na jedinične elemente što nas dovodi do standardnog modela elementarnih čestica, model koji sadrži sve (do sada) otkrivene elementarne čestice. *Otkrivenim* česticama smatramo one čije je postojanje eksperimentalno potvrđeno koristeći detektore čestica.

1.1 CMS detektor

Detektore možemo podijeliti na takozvane sudarače i nesudarače. Sudarači su oni u kojima dolazi do planiranog sudara čestica. Jedan od značajnih, kompaktni mionski solenoid (*engl. Compact Muon Solenoid* (CMS)), nalazi se u Europskoj organizaciji za nuklearna istraživanja (*franc. Organisation européenne pour la recherche nucléaire* (CERN)). CMS je aksijalni detektor koji se sastoji od više različitih koncentričnih slojeva. U središtu detektora generiraju se sudari čestica pri čemu nastaju nove čestice koje se od tog trenutka gibaju u nasumičnim smjerovima, a zatim prolaze kroz različite slojeve detektora. Ovisno o sloju detektora u kojem se čestica zaustavi moguće je odrediti o kojoj je čestici riječ. Elektroni prolaze kroz tri sloja silicijskih piksel detektora te deset slojeva silicijskih detektora s mikrotrakama u kojima se detektiraju pozicije elektrona u različitim trenutcima. Piksel detektor sastoji se od koncentričnih prstenova sastavljenih od 1800 silikonskih modula. Svaki od ovih modula sadrži oko 66 000 pojedinačnih piksela, što je ukupno 120 milijuna piksela. Ovako velik broj piksela omogućava računanje putanje čestice s preciznošću do na $10\ \mu$m [2].

Elektroni zatim bivaju zaustavljeni u elektromagnetskom kalorimetru (*Electromagnetic Calorimeter* (ECAL)) [3]. Putanja nabijenih čestica unutar CMS-a je kružna zbog magnetskog polja približne jačine od 4 T kojeg stvara solenoidni magnet postavljen izad ECAL-a. Poznavajući piksele silicijskog detektora kroz koje je elektron prošao može se izračunati radijus kružne putanje kojom se elektron kreće. Nadalje, iz radijusa i poznate jakosti magnetskog polja može se izračunati energija elektrona koji je prošao kroz detektor [4]. Primjer putanje elektrona i skica piksel detektora nalazi se na slici 1. Teorijski opis i analitički račun navedenog slijedi u poglavlju 1.2.

[1] Profesor filozofije i Sofija su likovi filozofske knjige *Sofijin svijet* [1] kojom književnik na jedinstven način nastoji čitatelje podučiti osnovnim idejama filozofije.



Slika 1: *Skica piksel detektora kroz koji prolazi čestica po putanji radijusa R unutar magnetskog polja jakosti B. Točke označavaju sjecišta putanje čestice i slojeva detektora, odnosno piksele kroz koje je čestica prošla. (slika preuzeta iz [5])*

Ideja ovog rada je programski simulirati pojednostavljenu verziju procesa koji se odvija u CMS detektoru. Kako bi postigli željeno potrebno je odabranom numeričkom metodom riješiti jednadžbe gibanja elektrona. Zatim se iz putanje elektrona pronalaze sjecišta sa koncentričnim kružnicama koje imaju ulogu silicijskih detektora. Iz pronađenih točaka definiramo kružnicu određenog radijusa koja predstavlja putanju elektrona, a zatim računamo količinu gibanja izgeneriranog elektrona. U konačnici analiziramo preciznost programa usporedbom s analitičkim rješenjem. Također, korisna je usporedba relativističkog i nerelativističkog pristupa simulaciji.

1.2 Elektron u magnetskom polju

Na električni nabijenu česticu u elektromagnetskom (EM) polju djeluje Lorentzova sila

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{1.1}$$

gdje je q naboj čestice, $\mathbf{E}$ vektor električnog polja, $\mathbf{B}$ vektor magnetskog polja, $\mathbf{v}$ brzina čestice, a $\mathbf{F}$ sila koja djeluje na česticu. Lorentzov zakon kaže da će električna sila na točkasti naboj biti u smjeru električnog polja, a magnetska sila okomita na smjer gibanja čestice i magnetsko polje [6]. Iz jednadžbe (1.1) slijedi:

$$m\mathbf{a} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{1.2}$$

gdje je m masa čestice, a $\mathbf{a}$ akceleracija čestice. Nadalje, slijedi:

$$\frac{d\mathbf{v}}{dt} = \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{1.3}$$

pri čemu je

$$\frac{d\mathbf{r}}{dt} = \mathbf{v} \tag{1.4}$$

pa dobivamo

$$\frac{d^2\mathbf{r}(t)}{dt^2} = \frac{q}{m}(\mathbf{E}(\mathbf{r}, t) + \frac{d\mathbf{r}}{dt} \times \mathbf{B}(\mathbf{r}, t)) \tag{1.5}$$

Gornja jednadžba (1.5) je vektorska obična diferencijalna jednadžba drugog stupnja čije rješenje daje funkciju položaja $\mathbf{r}(t)$. Ovisno o složenosti funkcija magnetskog i električnog polja analitičko rješenje može biti teško izračunati analitički. Preostaju nam numeričke metode rješavanja diferencijalnih jednadžbi o čemu će biti riječ u idućem poglavlju (2). Jednadžbe (1.3) i (1.4) koriste se pri postupku numeričkog računanja.

Ukoliko je nabijena čestica koja upada okomito na magnetsko polje elektron te ukoliko električno polje iščezava, Lorentzova jednadžba (1.1) se svodi na

$$\mathbf{F} = evB\hat{\mathbf{n}} \tag{1.6}$$

gdje je e elementarni naboj, v iznos brzine elektrona, B iznos magnetskog polja, a $\hat{n}$ jedinični vektor okomit na vektor brzine $\mathbf{v}$ i vektor magnetskog polja $\hat{B}$. Kako je sila okomita na smjer gibanja elektrona zaključujemo da će se elektron gibati kružno, odnosno Lorentzova sila u ovom slučaju ima ulogu centripetalne sile pa slijedi:

$$\frac{mv^2}{R} = evB \tag{1.7}$$

odnosno

$$R = \frac{mv}{eB} \tag{1.8}$$

gdje je R radijus kružnice po kojoj se elektron giba.

Ukoliko je brzina elektrona usporediva sa brzinom svjetlosti[2], relativistički učinak nije zanemariv te je potrebno *nadograditi* Lorentzovu jednadžbu uzimajući u obzir relativističke zakone. Relativistička Lorentzova jednadžba [7] je

$$\frac{d\mathbf{p}}{dt} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{1.9}$$

[2]izraz *usporediva sa brzinom svjetlosti* se koristi za brzine pri kojima je relativistički učinak značajan, dakle ne postoji egzaktna granica

Naizgled se čini jednaka nerelativističkoj, međutim moramo uzeti u obzir da vrijedi

$$\mathbf{p} = \gamma(\mathbf{v})m\mathbf{v} \tag{1.10}$$

gdje je $\gamma(v) = 1/\sqrt{1 - \frac{v^2}{c^2}}$ Lorentzov faktor. Sada $\frac{d\mathbf{p}}{dt}$ više nije jednako $m\frac{\mathbf{v}}{dt}$ jer faktor γ također ovisi o brzini $\mathbf{v}$, stoga uvodimo relaciju odnosa energije i količine gibanja

$$E = \sqrt{m^2c^4 + \mathbf{p}^2c^2} = \gamma mc^2 \tag{1.11}$$

Iz gornjih jednadžbi (1.10) i (1.11) brzinu možemo izraziti kao

$$\mathbf{v} = \frac{\mathbf{p}c^2}{\sqrt{m^2c^4 + \mathbf{p}^2c^2}} \tag{1.12}$$

stoga vrijedi

$$\frac{d\mathbf{r}}{dt} = \frac{\mathbf{p}c^2}{\sqrt{m^2c^4 + \mathbf{p}^2c^2}} \tag{1.13}$$

Iz jednadžbi (1.9) i (1.12) slijedi

$$\frac{d\mathbf{p}}{dt} = q\left(\mathbf{E} + \frac{\mathbf{p} \times \mathbf{B}}{\sqrt{m^2c^4 + \mathbf{p}^2c^2}}c^2\right) \tag{1.14}$$

Jednadžbe (1.13) i (1.14) koriste se u postupku numeričkog izračuna.

Radijus kružnice R po kojoj se elektron giba ukoliko električno polje iščezava te mu je smjer gibanja okomit na magnetsko polje u relativističkom slučaju je

$$R = \frac{\gamma mv}{eB} \tag{1.15}$$

2 Runge-Kutta 4

Najpristupačnija metoda numeričkog rješavanja običnih diferencijalnih jednadžbi prvog reda je Eulerova metoda, ali ova metoda ujedno generira i najveću pogrešku pa se odlučujemo za složenije metode. Odabiremo metodu Runge-Kutta 4 (RK4) zbog potrebne preciznosti koja je veća nego kod Eulerove [8]. Kako bismo pojasnili RK4 metodu potrebno je prvo objasniti Eulerovu metodu jer je RK4 u osnovi niz međukoraka sastavljenih Eulerovom metodom.

Derivacija funkcije y u točki $t\epsilon D(y)$, gdje je $D(y)$ domena funkcije y je

$$y'(t) = \frac{dy}{dt} = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t} \tag{2.1}$$

Za numerički pristup je potrebno uvesti diskretizaciju domene, odrediti početne uvjete te definirati korak Δt u oznaci h koji može biti proizvoljno malen. Što je korak h manji to je potreban veći broj koraka kako bi se došlo do krajnje točke, međutim što je h veći to je i nepreciznost, odnosno greška koju program generira veća. Diskretan oblik jednadžbe (2.1) je

$$f(t_i, y_i) = \frac{y_{i+1} - y_i}{h} \tag{2.2}$$

pri čemu je funkcija f jednaka derivaciji funkcije y, a indeks i označava broj trenutnog koraka. Funkcija f predstavlja nagib krivulje funkcije y u točki t_i te ju označavamo koeficijentom k. Dakle vrijedi

$$y_{i+1} = y_i + h \cdot k \tag{2.3}$$

$$t_{i+1} = t_i + h \tag{2.4}$$

Iz gornjih jednadžbi vidimo da je Eulerova metoda iteracijska što je razlog zašto uvodimo početne uvjete.

Ukoliko želimo dobiti precizniju metodu od Eulerove potrebno je osigurati da za jednaki korak h pronađemo precizniji nagib krivulje k. Kako bi se postiglo navedeno računaju se nagibi funkcije za četiri različite točke. Na slici 2 su označene sve četiri točke te pravci koje ih povezuju i njihovi nagibi. Postavljamo početni problem [9]

$$\frac{dy}{dt} = f(t, y)$$

$$y(t_0) = y_0$$

Zatim koristeći Eulerovu metodu računamo nagibe funkcije y u četiri različite točke, početnoj

točki intervala (t_i, y_i), dvije na polovici intervala te točki na kraju intervala (t_{i+1}, y_{i+1})

$$\begin{aligned} k_1 &= f(t_i, y_i) \\ k_2 &= f(t_i + \frac{h}{2}, y_i + h\frac{k_1}{2}) \\ k_3 &= f(t_i + \frac{h}{2}, y_i + h\frac{k_2}{2}) \\ k_4 &= f(t_i + h, y_i + h \cdot k_3) \end{aligned} \tag{2.5}$$

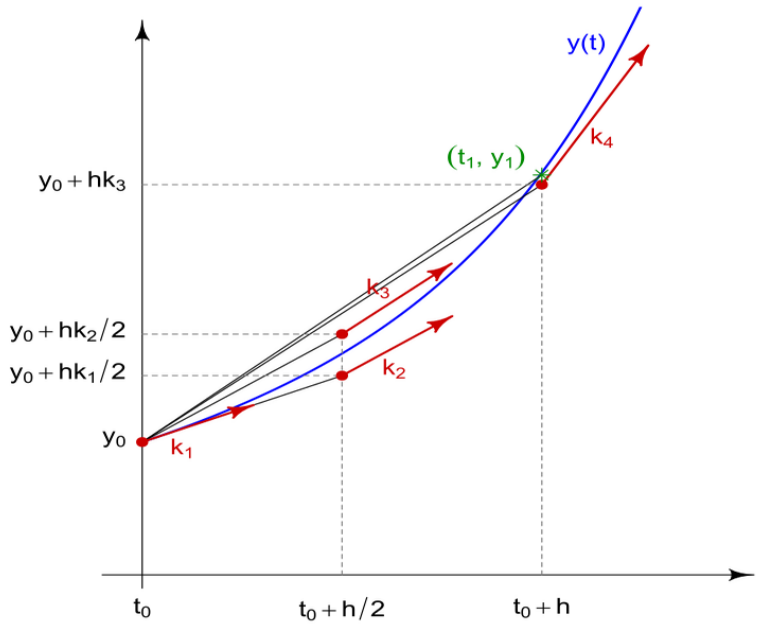
Sada tražimo prosjek navedenih nagiba, ali želimo da središnje točke imaju veći doprinos

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \cdot h \tag{2.6}$$

Također, kao i kod Eulerove metode vrijedi

$$t_{i+1} = t_i + h \tag{2.7}$$

Ovime smo definirali metodu Runge-Kutta 4.



Slika 2: *Grafički prikaz Runge-Kutta 4 metode. Računamo nagibe funkcije u četiri različite točke te pronalazimo njihov prosjek . Odabiremo početnu točku intervala, krajnju točku intervala te dvije točke na polovici intervala (slika preuzeta s https://en.wikipedia.org/ [9])*

3 Postupak rješavanja

3.1 Početne postavke

Prvi korak je izgenerirati elektron ukupne količine gibanja 10 GeV/c u središtu koordinatnog sustava koji se nalazi u homogenom magnetskom polju jakosti 3.8 T. Kao središte koordinatnog sustava uzimamo mjesto "sudara", odnosno točku iz koje elektron izlazi, ali pretpostavljamo da je ta točka ujedno i središte detektora. Električno polje iščezava. Smjer gibanja elektrona okomit je na magnetsko polje usmjereno u smjeru z-osi, dakle postoje samo x i y komponente količine gibanja, a time i brzine elektrona. Komponente količine gibanja biramo nasumično, ali tako da ukupna količina gibanja ostane $p = 10$ GeV/c, dakle vrijedi $\sqrt{p_x^2 + p_y^2} = p$, pri čemu su p_x i p_y komponente količine gibanja. U programskom dijelu rada korišten je programski jezik Python. Unutar programu koristimo funkciju *random.uniform(min,max)* koja vraća brojeve tipa float unutar zadanog intervala $[min, max\rangle$, svaka vrijednost iz intervala je jednako vjerojatna [10]:

```
115     px = random.uniform(0,_xLimit)
116     py = sqrt(p*p-px*px)
```

3.2 Numeričko računanje

Cilj nam je pronaći putanju elektrona, koristeći RK4 metodu rješavamo jednadžbu (1.5). Kako RK4 metodom možemo rješavati samo obične diferencijalne jednadžbe prvog reda jednadžbu (1.5) rastavljamo na dvije jednadžbe prvog reda (1.3) i (1.4) koje rješavamo paralelno:

```
14  def dvdt(_E,_B,_v):
15      return e/m*(_E+np.cross(_v,_B))
16
17  def drdt(_v):
18      return _v
```

Kod metode RK4 se nalazi u dodatku A (40-74). Varijabla vremena nije ukjučena jer nijedna jednadžba koju rješavamo ne ovisi eksplicitno o vremenu te bi uključivanje dodatne varijable bilo nepotrebno korištenje resursa memorije i vremena. Pri pozivu funkcije *RK4()* potrebno je subjektivno odabrati korak h i broj koraka N. U slučaju premalog koraka h potreban je veliki broj koraka N što uzrokuje duže vrijeme izvršavanja programa, a u slučaju prevelikog koraka h dolazi do veće pogreške.

Kako funkcije $RK4()$, $dvdt()$ i $drdt()$ primaju varijablu brzine, a ne količine gibanja potrebno je zadanu količinu gibanja prerečunati u ekvivalent brzine, međutim u izračun je potrebno uvesti relativistiku, u suprotnom je dobivena brzina veća od brzine svjetlosti što se protivi

postulatu specijalne teorije relativnosti. Iz relativističke relacije količine gibanja (1.10) računamo brzinu

$$p^2 = \gamma^2 m^2 v^2 \tag{3.1}$$

$$\frac{m^2}{p^2} v^2 = 1 - \frac{v^2}{c^2} \tag{3.2}$$

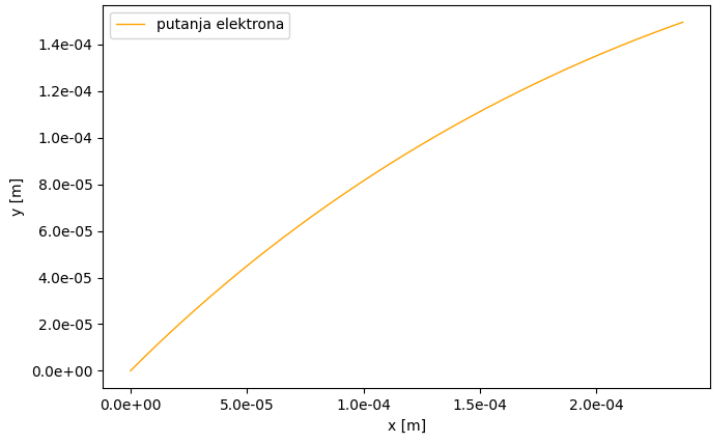
$$\left(\frac{m^2}{p^2} + \frac{1}{c^2} \right) v^2 = 1 \tag{3.3}$$

$$v^2 = \frac{1}{\frac{m^2 c^2}{p^2} + 1} c^2 \tag{3.4}$$

Ukoliko količinu gibanja izrazimo u GeV/c, a masu elektrona u GeV/c^2, jedinice ostaju očuvane. Funkcija *momentum_to_speed(_p)* prima količinu gibanja u GeV/c te vraća brzinu u m/s:

```
21 def momentum_to_speed_electron(_p):      #kolicina gibanja u GeV/c
22     _m = 0.51099895 #masa elektrona u MeV/c^2
23     _v2 = 1/((_m*_m)/(_p*1000*_p*1000) + 1)*c*c
24     return sqrt(_v2)
```

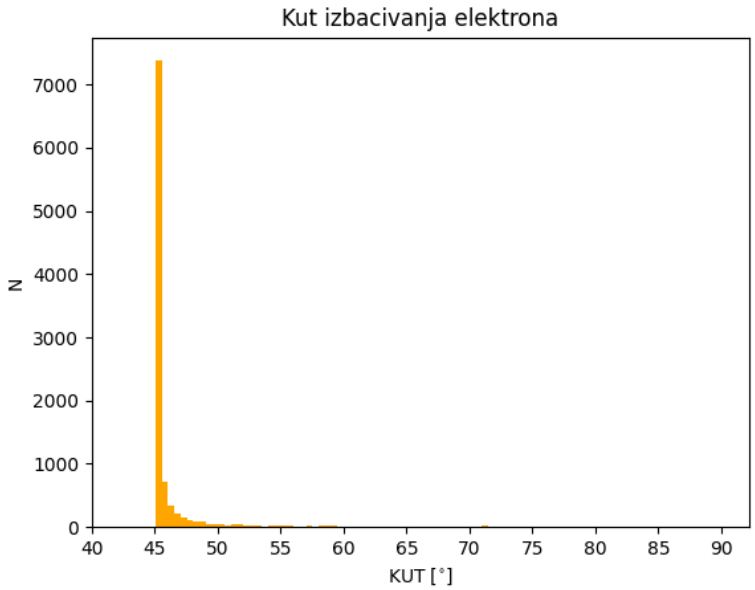
Koristeći gore navedene funkcije dobivamo polje (*engl. array*) koje sadrži uzastopne točke u kojima se elektron nalazi. Iz dobivenog polja možemo crtati krivulju putanje elektrona koristeći *matplotlib* biblioteku, primjer putanje prikazan je na slici 3.



Slika 3: *Putanja elektrona izgenerirana RK4 metodom uz korak* $h = 1e{-}16$ *te broj koraka* $N = 10000$.

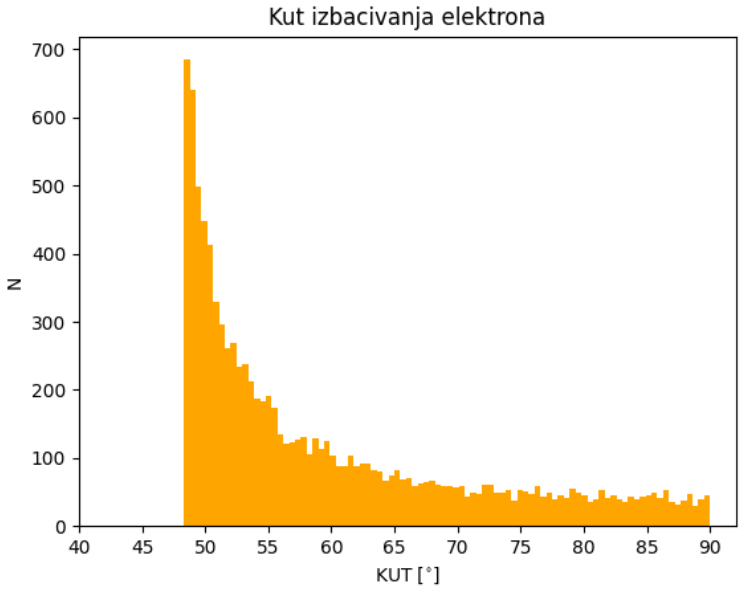
Uočimo da je u odabranom slučaju kut *izbacivanja* elektrona u odnosu na x-os relativno blizak iznosu od $45°$ što je ujedno i najvjerojatniji slučaj jer funkcijom random.uniform() nasumično odabiremo količinu gibanja, a ne brzinu što uzrokuje da je najvjerojatniji slučaj da su

komponente brzine v_x i v_y približno jednake. Razlog je taj što funkciji random.uniform() zadajemo interval količine gibanja od 0 GeV/c do 10 GeV/c, za ovako relativno velike iznose količine gibanja brzine elektrona su vrlo slične, do značajnije razlike u komponentama brzine dolazi tek kada je iznos jedne komponente količine gibanja za nekoliko redova veličine manji od druge komponente, na histogramu 4 je vidljiva raspodjela po kutovima za slučaj kada je zadano da je komponenta p_x tri reda veličine manja od komponente p_y.



Slika 4: *Zastupljenost kuta izbacivanja elektrona u odnosu na os x za nasumično odabrane komponente količine gibanja za slučaj kada je zadano da je komponenta p_x tri reda veličine manja od komponente p_y.*

Stoga odabiremo slučaj kada je komponenta p_x četiri do šest redova veličine manja nego komponenta p_y. Ovime smo osigurali da kutovi izbacivanja pokrivaju veći interval. Na histogramu 5 pokazana je raspodjela kutova za slučaj kada je jedna komponenta četiri reda veličine manja od druge.



Slika 5: *Zastupljenost kuta izbacivanja elektrona u odnosu na os x za nasumično odabrane komponente količine gibanja za slučaj kada je zadano da je komponenta p_x četiri reda veličine manja od komponente p_y.*

Sljedeći cilj nam je pronaći radijus putanje po kojoj se elektron giba. Kako bismo pronašli željeno, definiramo četiri kružnice proizvoljnog radijusa za koje smo sigurni da imaju sjecišta sa putanjom elektrona. Zatim pronalazimo točke sjecišta. Ovime pronalazimo četiri točke kroz koje elektron prolazi, a također znamo da mu je početna točka ishodište koordinatnog sustava. Kružnica je jednoznačno određena trima točkama pa uz ishodište odabiremo još dvije točke iz kojih pronalazimo kružnicu, a time i radijus putanje.

Funkcija *IsOnCircle(_position, _R1, _R2, _R3, _R4)* prima radijuse navedenih kružnica te polje u kojem se nalaze točke putanje elektrona, a vraća polje koje sadrži pronađena sjecišta:

```
75 def IsOnCircle(_position, _R1, _R2, _R3, _R4):
76     _targetPoints = []
77     _counter0 = 0
78     _counter1 = 0
79     _counter2 = 0
80     _counter3 = 0
81     _counter4 = 0
82     _numOfDec = 9
83 
84     for i in range(0,np.shape(_position)[0]):
85 
86         _temp = round( (pow(_position[i,0].real,2)+pow(_position[i,1].real↩
               ,2)) ,_numOfDec)
87 
88         if(_temp == 0.0 and _counter0 == 0):
```

```
89              _counter0+=1
90              _targetPoints.append(_position[i,:])
91
92          if(_temp == pow(_R1,2) and _counter1 == 0):
93              _counter1+=1
94              _targetPoints.append(_position[i,:])
95
96          if(_temp == pow(_R2,2) and _counter2 == 0):
97              _counter2+=1
98              _targetPoints.append(_position[i,:])
99
100         if(_temp == pow(_R3,2) and _counter3 == 0):
101             _counter3+=1
102             _targetPoints.append(_position[i,:])
103
104         if(_temp == pow(_R4,2) and _counter4 == 0):
105             _counter4+=1
106             _targetPoints.append(_position[i,:])
107
108     return np.array(_targetPoints)
```

Unutar funkcije *IsOnCircle()* definiramo broj decimala (*_numOfDec*) na koje zaokružujemo udaljenost određene točke putanje od ishodišta. Takvim postupkom generiramo određenu pogrešku jer točki dopuštamo interval unutar kojeg se može nalaziti, dakle ne mora biti točno na kružnici već mora biti dovoljno blizu nje. Međutim, ovaj korak je nužan kako bismo pronašli sjecišta, u protivnome nećemo naći željeno jer je malo vjerojatno da je udaljenost točke od središta koordinatnog sustava jednaka radijusu kružnice do u posljednju poznatu decimalu. Kako dopuštamo određeni interval moguće je da se više točaka nađe unutar tog intervala, što bi značilo da su sve točke unutar intervala prepoznate kao sjecište koje je jedinstveno. Kako bi se ograničili na jednu točku, uzimamo samo prvu nađenu. Ovim odabirom stvaramo dodatnu pogrešku jer uzimamo točku iz intervala koja je najudaljenija od sjecišta.

Poznavajući dovoljan broj točaka koje joj pripadaju, sada možemo tražiti jednadžbu kružnice. Od pet navedenih točaka biramo točku ishodišta te još dvije. Jednadžba kružnice je

$$(x-p)^2 + (y-q)^2 = R^2 \tag{3.5}$$

pri čemu su p i q koordinate središta $S(p,q)$ kružnice, a R radijus kružnice. Uvrštavajući točku $O(0,0)$ u jednadžbu kružnice (3.5) slijedi

$$p^2 + q^2 = R^2 \tag{3.6}$$

Raspisujući jednadžnu (3.5) imamo

$$x^2 - 2px + p^2 + y^2 - 2qy + q^2 = R^2 \tag{3.7}$$

pa uvrštavanjem jednakosti $p^2 + q^2 = R^2$ slijedi

$$2xp + 2yq = x^2 + y^2 \tag{3.8}$$

zatim uvrštavanjem koordinata još dviju točaka dobivamo sustav dviju linearnih jednadžbi sa dvije nepoznanice, p i q:

$$\begin{aligned} 2x_1p + 2y_1q &= x_1^2 + y_1^2 \\ 2x_2p + 2y_2q &= x_2^2 + y_2^2 \end{aligned} \tag{3.9}$$

Sustav možemo zapisati koristeći matrice

$$\begin{bmatrix} 2x_1 & 2y_1 \\ 2x_2 & 2y_2 \end{bmatrix} \cdot \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \end{bmatrix} \tag{3.10}$$

Ova matrična jednadžba ima oblik $A \cdot X = B$. Rješavamo ju tako da cijelu jednadžbu pomnožimo s desna s inverzom matrice A pa ostaje $X = A^{-1}B$.

Funkcija FindRadius() opisanom metodom pronalazi radijus kružnice R :

```
141     A = np.array([ [-2*targetPoints[1,0], -2*targetPoints[1,1]] , [-2*↩
            targetPoints[2,0], -2*targetPoints[2,1]] ])
142     B = np.array([-1*pow(targetPoints[1,0],2)-1*pow(targetPoints[1,1],2), ↩
            -1*pow(targetPoints[2,0],2)-1*pow(targetPoints[2,1],2)])
143     X = np.linalg.inv(A).dot(B)
144
145     x0 = X[0]
146     y0 = X[1]
147
148     R = sqrt(X[0]*X[0]+X[1]*X[1]).real
```

Uzimao samo tri točke jer su nam one dovoljne da definiraju kružnicu, uzimanjem četvrte točke postoji mogućnost da funkcija ne pronađe rješenje. Program do ovog trenutka uzrokuje određeno odstupanje od idealnog slučaja što znači da četiri, odnosno pet pronađenih točaka neće idealno *ležati* na istoj kružnici. Dakle, sustav od četiri linearne jednadžbe s dvije nepoznanice neće imati dva para linearno zavisnih jednadžbi pa sustav neće imati rješenje. Drugi razlog odabira samo dviju točaka je taj što neće uvijek sva sjecišta biti pronađena, ukoliko je točka putanje koja je najbliža analitičkom sjecištu izvan intervala uzrokovanog zaokruživanjem, neće biti prepoznata kao sjecište.

Kako bismo definirali preciznost algoritma potrebno je dovoljan broj puta izračunati radijus i rezultate prikazati na histogramu te proučiti dobivenu razdiobu. Iz radijusa računamo količinu gibanja kako bismo mogli usporediti početnu količinu gibanja, takozvanu generiranu p_{GEN} te količinu gibanja koju računa algoritam, takozvanu rekonstruiranu p_{REC}. Računamo ih koristeći jednadžbu (1.8) pri čemu je $p = mv$ pa vrijedi

$$p = ReB \tag{3.11}$$

gdje je p količina gibanja, R radijus putanje elektrona, B jakost magnetskog polja, a e elementarni naboj.

Ukoliko usporedimo rekonstruiranu količinu gibanja sa početnom (generiranom) količinom gibanja, odstupanje će biti značajno jer je iznos od 10 GeV/c moguće dobiti samo uz relativistički račun. U ovom algoritmu relativistički račun je korišten samo kako bismo izračunali brzinu gibanja elektrona, nadalje je račun potpuno klasičan, zato računamo generiranu količinu gibanja preko generiranog (analitički izračunatog) radijusa :

```
164 #racunamo analiticki radijus
165 R_gen = (m*momentum_to_speed_electron(10))/(e*Bz)
166 R_gen = R_gen.real
167 #print('\nR analiticki:', R_gen, '\n')
168 
169 #racunamo (analiticku) kolicinu gibanja
170 p_gen = e*Bz*R_gen    #nece bit 10Gev
```

3.3 Relativistička korekcija

Zbog brzine elektrona koja je usporediva sa brzinom svjetlosti za dobivanje realnih rezultata potrebno je koristiti relativističke relacije. Ponovno koristimo metodu RK4, ali jednadžbe koje rješavamo su korigirane. Numerički rješavamo jednadžbe (1.9) i (1.13) :

```
13 def drdt(_p):
14     return (_p*c)/sqrt(m*m*c*c+np.dot(_p,_p))
15 
16 def dpdt(_p,_B):
17     return e*c*np.cross(_p,_B)/sqrt(m*m*c*c+np.dot(_p,_p))
```

Ovo je jedina značajna korekcija algoritma u odnosu na prethodni. Također, sada možemo uspoređivati rekonstruiranu količinu gibanja sa početnom količinom gibanja jer računanjem preko generiranog radijusa dobivamo jednaku vrijednost zbog korištenja relativističkih relacija.

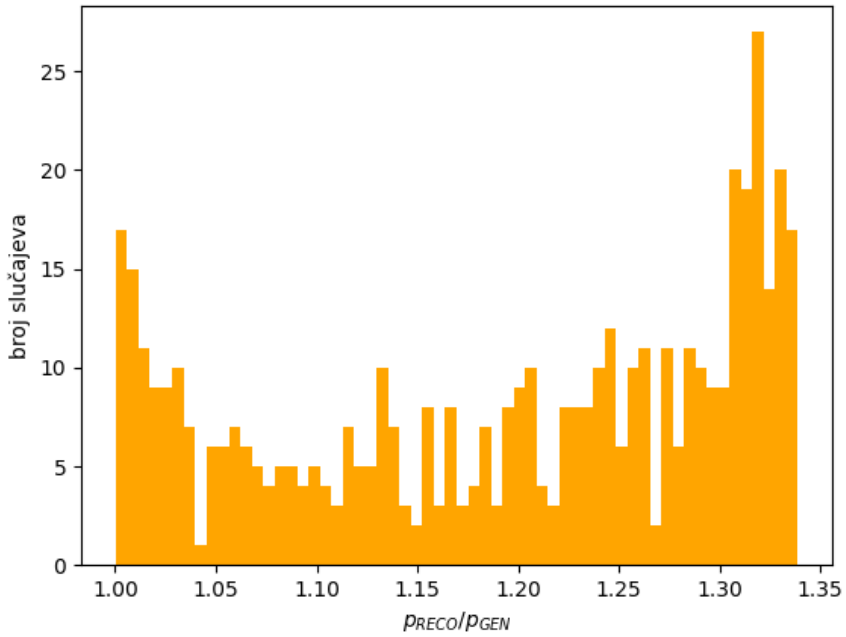
4 Rezultati

Program nam daje konačne rezultate u obliku histograma. Na y-osi je uvijek prikazan broj slučajeva kod kojih je dobivena određena vrijednost veličine prikazane na x-osi. Na x-osi se mogu nalaziti vrijednosti rekonstruiranog radijusa, omjer rekonstruiranog i generiranog radijusa ili omjer rekonstruirane i generirane količine gibanja. Ukoliko odaberemo prikazati histogram rekonstruiranog radijusa, očekujemo da raspodjela ima vrh (*engl. peak*) oko vrijednosti generiranog radijusa te da ima oblik normalne raspodjele, a ukoliko prikažemo histogram omjera rekonstruiranih i generiranih veličina, očekujemo da raspodjela ima vrh oko vrijednosti jedan. Što je program precizniji to bi normalna raspodjela trebala težiti delta funkciji. Što je program neprecizniji to bi širina razdiobe, odnosno standardna devijacija trebala biti veća ili bi vrh razdiobe trebao biti translatiran put neke vrijednosti različite od jedan.

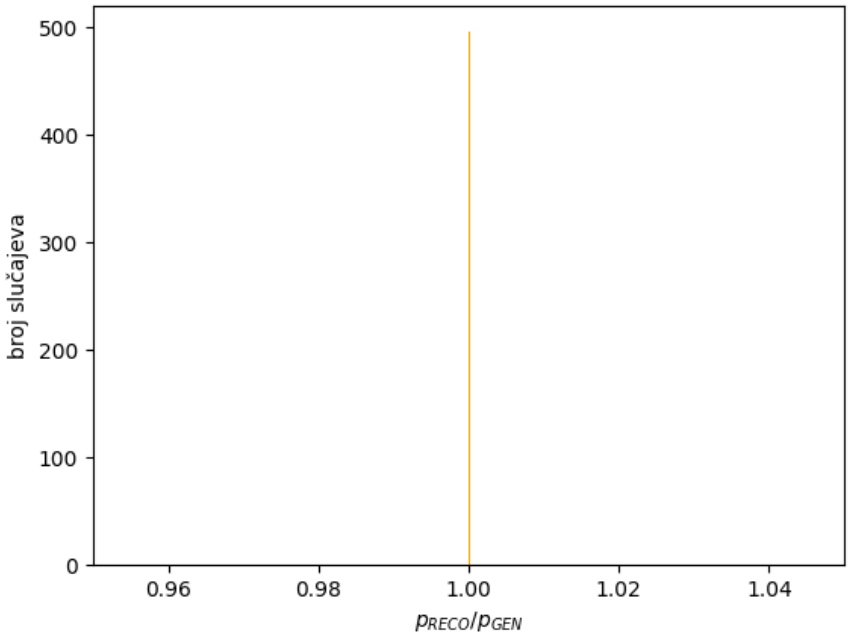
Prilikom analize dobivenih histograma uočeno je da razdioba ovisi o ukupnom broju generiranih radijusa, zatim o veličini koraka h unutar metode RK4() te o omjeru komponenti količine gibanja p_x i p_y.

Na grafu 6 se nalazi histogram u kojem je prikazana raspodjela od 500 radijusa uz korak $h = 1\text{e}{-15}$ i razliku komponenti količine gibanja od 4 reda veličine. Uočavamo da razdioba ne poprima oblik normalne razdiobe niti teži delta funkciji. Unutar 1% od očekivane vrijednosti nalazi se 22.38% slučajeva, u intervalu od 1% do 10% odstupanja od očekivane vrijednosti nalazi se 32.26% slučajeva, a preostalih 45.36% odstupa više od 10%. Ipak, svih 500 slučajeva ima odstupanje manje od 35% od očekivane vrijednosti.

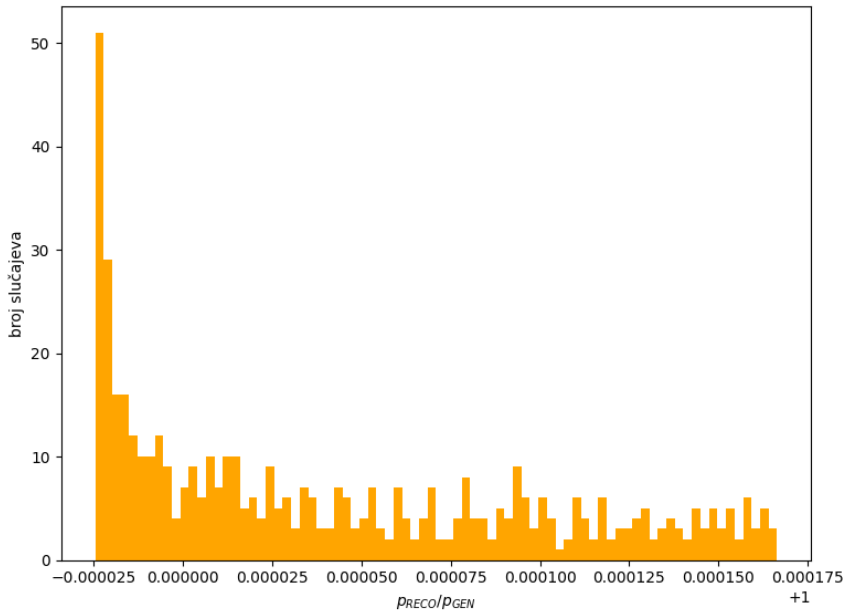
Na grafu 7 se nalazi histogram u kojem je prikazana raspodjela od 500 radijusa uz korak $h = 1\text{e}{-15}$ i razliku komponenti količine gibanja od 6 redova veličine. Uočavamo da razdioba teži delta funkciji što upućuje na visoku preciznost programa. Svi slučajevi nalaze se unutar 1% odstupanja od očekivane vrijednosti. U ovom primjeru uočeno je da postoji određeno minimalno odstupanje od očekivane vrijednosti, što je prikazano na grafu 8, ali uočavamo da razdioba nije normalna. Očekujemo da postoji određeni broj slučajeva vrijednosti manjih od očekivane te isto tako i za vrijednosti veće od očekivane te da je vrh u jedinici. Na navedenom grafu ne uočavamo vrijednosti manje od one u kojoj se nalazi vrh razdiobe te je vrh pomaknut prema nešto manjim vrijednostima od očekivane.



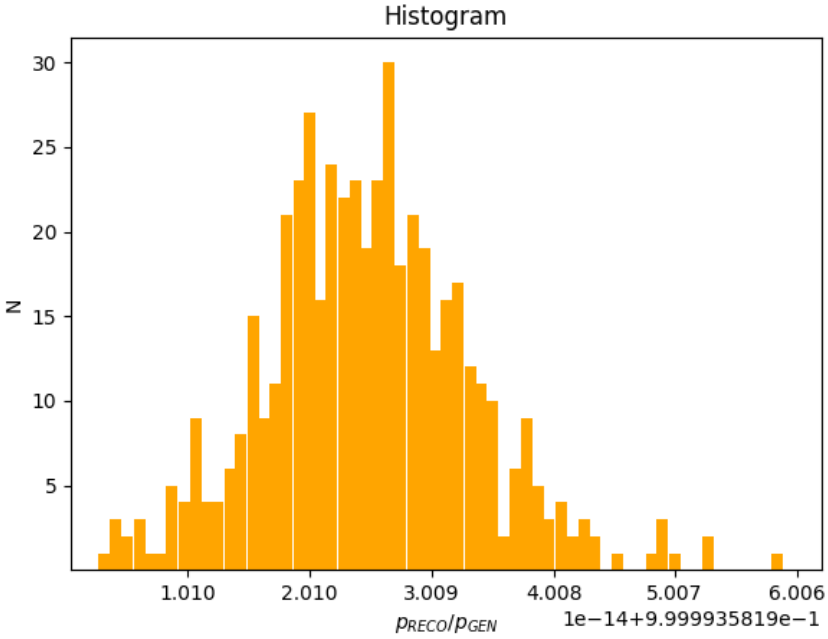
Slika 6: *Histogram generiran za 500 radijusa, uz korak $h = 1e{-}15$ i razliku između komponenti količine gibanja od 4 reda veličine.*



Slika 7: *Histogram generiran za 500 radijusa, uz korak $h = 1e{-}15$ i razliku između komponenti količine gibanja od 6 redova veličine.*

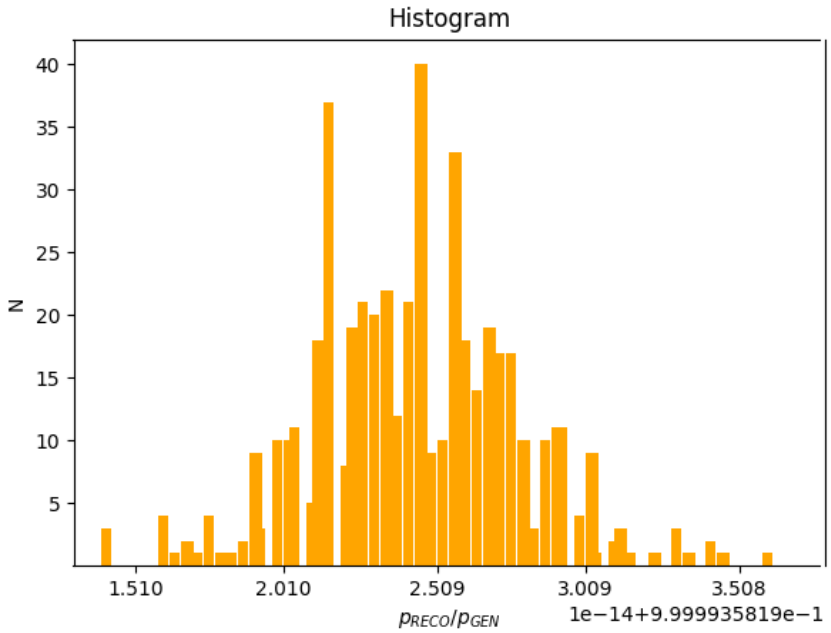


Slika 8: *Histogram generiran za 500 radijusa, uz korak* $h = 1e\text{-}15$ *i razliku između komponenti količine gibanja od 6 redova veličine.*



Slika 9: *Histogram generiran programom u koji su uvršteni relativistički zakoni, za 500 radijusa, uz korak* $h = 1e{-}11$ *te uz komponente količine gibanja istog reda veličine.*

Na grafu 9 prikazan je histogram dobiven programom u koji su uvršteni relativistički zakoni. Generiran je za ukupno 500 slučajeva, uz korak $h = 1e - 11$ te za komponente količine gibanja jednakog reda veličine. Uočavamo normalnu razdiobu, sve vrijednosti se nalaze unutar manje od 0.01% od očekivane vrijednosti, ipak sve vrijednosti su manje od očekivane, odnosno vrh razdiobe je vrlo malo pomaknut put negativne strane osi x. Ukoliko usporedimo navedeni graf sa grafom 10 uočavamo da obje razdiobe poprimaju oblik normalne razdiobe te da je interval pogođenih vrijednosti sličan. Vrh se ne postiže u potpuno jednakim vrijednostima, ali možemo zaključiti da su vrijednosti dovoljno slične te da je navedena razlika samo moguće odstupanje zbog nasumičnog izbora komponenti količine gibanja. Odnosno, kod u koji su uvršteni relativistički zakoni daje zadovoljavajuće rezultate neovisno o omjeru komponenti količine gibanja, što kod programa bez uvrštenih relativističkih zakona to nije slučaj.



Slika 10: *Histogram generiran programom u koji su uvršteni relativistički zakoni, za 500 radijusa, uz korak $h = 1e - 11$ te uz komponente količine gibanja koje se razlikuju za 4 reda veličine.*

4.1 Usporedba radijusa

Nerelativistički radijus putanje elektrona računamo koristeći jednadžbu (1.8), ali kako bismo izračunali brzinu elektrona koji ima količinu gibanja $p = 10$ GeV moramo iskoristiti relativističku jednadžbu (3.4) u protivnome bi bila znatno veća od brzine svjetlosti. Ovim izračunom i uvrštavanjem vrijednosti $B = 3.8$ T, $q = e$ i $m = m_e$ dobije se radijus koji iznosi

$$R = 0.000448555 \text{ m}$$

što je iznos koji se razlikuje za tri do četiri reda veličine od očekivanih vrijednosti.

U literaturi [5] pronalazimo jednadžbu koja povezuje količinu gibanja, jakost magnetskog polja i radijus putanje čestice u detektoru[3]

$$p = 0.3BR \tag{4.1}$$

gdje je p količina gibanja čestice izražena u GeV/c, B jakost magnestkog polja u jedinici tesla, a R radijus u metrima. Uvrštavanjem već navedenih vrijednosti u jednadžbu (4.1) dobivamo

$$R = 8.771929825 \text{ m}$$

Izračunamo li radijus putanje elektrona koristeći relativističku relaciju (1.15) dobivamo

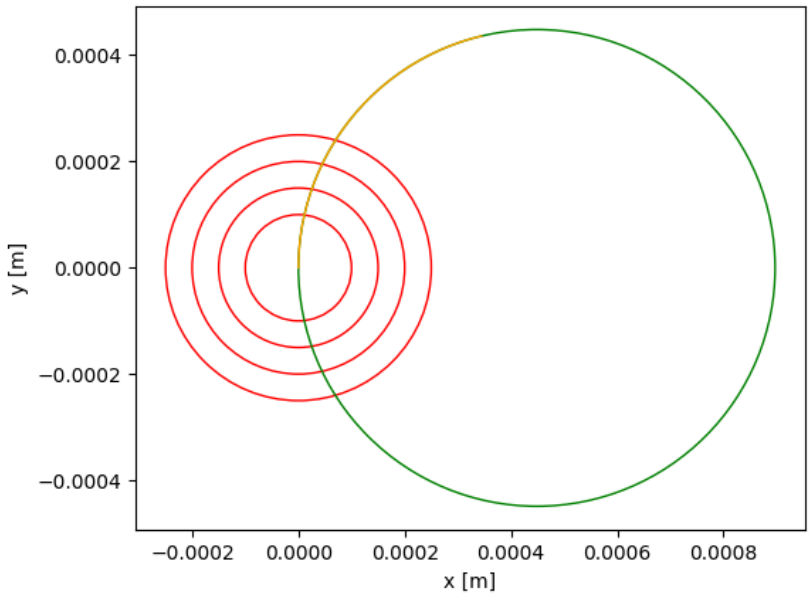
$$R = 8.778002514 \text{ m}$$

Uočavamo da je radijus koji bi trebali dobiti u relativističkom kodu gotovo pa jednak radijusu iz literature.

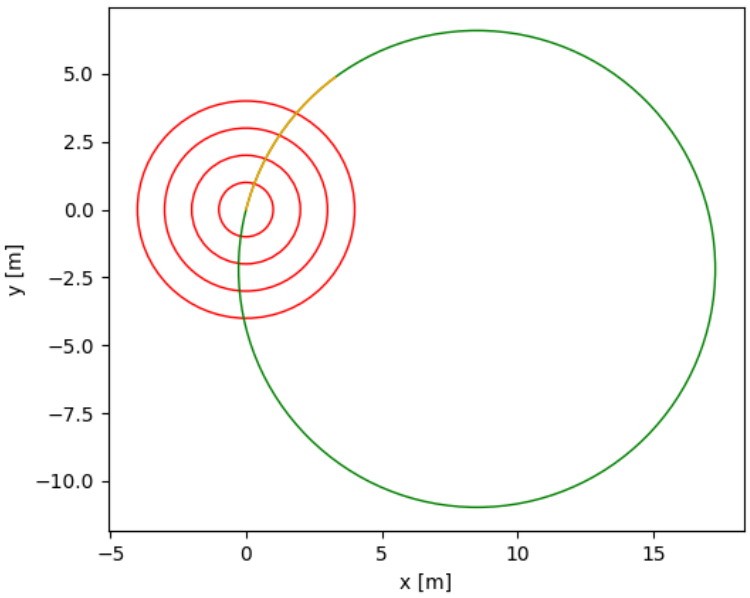
Na slici 11 je prikazana skica koju je generirao program u koji nije uvedena relativistika za korak $h = 1\text{e}{-15}$ te se komponete količine gibanja razlikuju za šest redova veličine. U ovom slučaju rekonstruirani radijus iznosi $R_{RECO} = 0.00044855$ m. Ova vrijednost je očekivana za program koji zanemaruju relativistiku, ali pokazuje potpuno odstupanje od vrijednosti iz literature.

Na slici 12 je prikazana skica koju je generirao program u koji je uvedena relativistika za korak $h = 1\text{e}{-15}$ te količine gibanja jednakih redova veličine. U ovom slučaju rekonstruirani radijus iznosi $R_{RECO} = 8.7779461$ m što je vrlo blizu očekivane vrijednosti za relativistički izračun, a time i vrijednosti iz literature.

[3] U literaturi [5] je navedena proširena jednadžba $p \cos\lambda = 0.3BR$ gdje je λ kut pod kojim čestica upada u magnetsko polje u odnosu na okomicu magnetskog polja, što u našem slučaju iznosi $\lambda = 0$ jer elektron upada okomito na magnetsko polje.



Slika 11: *Skica koju generira program u koji nije uvedena relativistika. U programu su za ovaj slučaj uzeti korak $h = 1e{-}15$ te se komponete količine gibanja razlikuju za šest redova veličine. Crvene kružnice simuliraju piksel detektore, žuta linija je generirana putanja elektrona, a zelena linija je rekonstruirana putanja.*



Slika 12: *Skica koju generira program u koji je uvedena relativistika. U programu su za ovaj slučaj uzeti korak $h = 1e{-}11$ te su komponete količine gibanja jednakog reda veličine. Crvene kružnice simuliraju piksel detektore, žuta linija je generirana putanja elektrona, a zelena linija je rekonstruirana putanja.*

5 Zaključak

Ideja rada bila je simulirati postupak koji se odvija u prvim slojevima CMS detektora, unutar silicijskih piksel detektora koji se nalaze unutar homogenog magnetskog polja. Kako bismo to realizirali bilo je potrebno riješiti diferencijalne jednadžbe gibanja elektrona koristeći metodu Runge-Kutta 4, zatim definirati koncentrične kružnice koje predstavljaju piksel detektore, pronaći sjecišta putanje elektrona sa kružnicama te iz pronađenih točaka definirati kružnicu po kojoj se elektron giba. Idući cilj bio je pronaći razdiobu vrijednosti rekonstruiranih radijusa te time procijeniti preciznost osmišljenog algoritma.

Program u kojem nije uvrštena relativistika u konačnici daje dobre rezultate jer ne pokazuje odstupanje veće od 35% od generiranog radijusa, međutim uz određene postavke može postići odstupanje od 1% što je vrlo zadovoljavajući rezultat. Nedostatak algoritma je što njegova preciznost ovisi o omjeru iznosa komponenti količine gibanja, što je njihova razlika veća to je algoritam precizniji. Dakle za razlike od nekoliko redova veličine između komponenti dobivamo približno delta funkciju što znači da su svi rekonstruirani radijusi gotovo pa jednaki generiranom radijusu. Također, treba naglasiti da radijus koji računamo bez uvrštavanja relativistike odstupa za nekoliko redova veličine od vrijednosti iz literature što je bila glavna motivacija za uključenje relativistike u algoritam. Kod u kojemu je uvrštena relativistika ne razlikuje se značajno od početnog koda, ali daje značajno drugačije rezultate. Rekonstruirani radijus vrlo malo odstupa od generiranog radijusa, za manje od 0.01% i to neovisno o omjeru komponenti količine gibanja. Također, uočavamo da razdioba rekonstruiranih radijusa prati normalnu razdiobu, što je dobar rezultat. Rekonstruirani radijus slaže se sa vrijednostima iz literature što nam govori da bez relativističkih zakona ne možemo objasniti gibanje elektrona unutar detektora.

Rad se može unaprijediti tako da se dodatno analizira zašto početni kod ne prati normalnu razdiobu te zašto njegova preciznost ovisi o omjeru komponenti količine gibanja. Mogući problem je što zadajemo količinu gibanja od $p = 10$ GeV/c što je nedostižna količina gibanja za elektron ukoliko se ne uvede relativistika. Unatoč navedenim nedostacima smatram da rad dobro prikazuje osnove silicijskih piksel detektora te ideju analiziranja elementarnih čestica. Također, vrlo je dobro što možemo uočiti doprinos koji imaju relativistički zakoni pri brzinama koje su bliske brzini svjetlosti te kako ih je nužno koristiti pri tim brzinama.

6 Literatura

[1] J. Gaarder, Sofijin svijet, Znanje, Zagreb 1997.

[2] CERN, Successful installation of the CMS Pixel Tracker,
URL:
https://home.cern/news/news/experiments/successful-installation-cms-pixel-tracker
(7. 9. 2022.)

[3] R. Pleština, Potencijal CMS detektora za potragu za Higgsovim bozonom kroz kanal raspada H $\rightarrow$ ZZ* $\rightarrow$ 4e$\pm$, diplomski rad, 22-25 (2008)

[4] CERN, How CMS works,
URL: https://cms.cern/index.php/detector/identifying-tracks (5. 9. 2022.)

[5] M. Thomson, Modern Particle Physics, Cambridge University Press, Cambrige, 2013.

[6] Lorentz force, Wikipedia,
URL: https://en.wikipedia.org/wiki/Lorentz_force (5. 9. 2022.)

[7] V. Horný, Relativistic motion equation solver, 1-2 (2016).

[8] L. Svilan, Numeričko određivanje putanje naboja u elektromagnetskom polju, Završni rad, 24 (2021)
URL: https://urn.nsk.hr/urn:nbn:hr:166:479817 (05.09.2022.)

[9] Runge-Kutta 4, Wikipedia,
URL: https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods (5.9.2022.)

[10] Funkcija random.uniforme(), NumPy,
URL:
https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html
(5.9.2022.)

A Kod bez relativistike

```
from cmath import sqrt
from turtle import color
import numpy as np
from scipy.constants import elementary_charge, electron_mass, ↩
    speed_of_light
import random

e = elementary_charge
m = electron_mass
c = speed_of_light
Bz = 3.8 #T

#definiramo Lorentzovu silu, odnosno dv/dt

def dvdt(_E,_B,_v):
    return e/m*(_E+np.cross(_v,_B))

def drdt(_v):
    return _v


def momentum_to_speed_electron(_p):     #kolicina gibanja u GeV/c
    _m = 0.51099895 #masa elektrona u MeV/c^2
    _v2 = 1/((_m*_m)/(_p*1000*_p*1000) + 1)*c*c
    return sqrt(_v2)

def Euler(_h,N,_r,_v,_E,_B):
    _position = []
    _velocity = []
    _position.append(_r)
    _velocity.append(_v)
    for i in range(1,N):
        _r = _r + drdt(_v)*_h
        _v = _v + dvdt(_E,_B,_v)*_h
        _position.append(_r)
        _velocity.append(_v)
    return np.array(_position)

def RK4(_h,_N,_r,_v,_E,_B):
    _position = []
    _velocity = []
    _position.append(_r)
    _velocity.append(_v)
    for i in range(1,_N):
```

```
44
45          kr = drdt(_v)
46          sum_kr = kr
47          kv = dvdt(_E,_B,_v)
48          sum_kv = kv
49
50          r_temp = _r + kr*_h/2.0
51          v_temp = _v + kv*_h/2.0
52
53          kr =drdt(v_temp)
54          sum_kr = sum_kr + 2.0*kr
55          kv = dvdt(_E,_B,v_temp)
56          sum_kv += 2.0*kv
57
58          r_temp = _r + kr*_h
59          v_temp = _v + kv*_h
60
61          kr = drdt(v_temp)
62          sum_kr += kr
63          kv = dvdt(_E,_B,v_temp)
64          sum_kv += kv
65
66          _r = _r + sum_kr*_h/6.0
67          _v = _v + sum_kv*_h/6.0
68
69          _position.append(_r)
70          _velocity.append(_v)
71
72      return np.array(_position)
73
74
75  def IsOnCircle(_position, _R1, _R2, _R3, _R4):
76      _targetPoints = []
77      _counter0 = 0
78      _counter1 = 0
79      _counter2 = 0
80      _counter3 = 0
81      _counter4 = 0
82      _numOfDec = 9
83
84      for i in range(0,np.shape(_position)[0]):
85
86          _temp = round( (pow(_position[i,0].real,2)+pow(_position[i,1].real↩
                ,2)) ,_numOfDec)
87
88          if(_temp == 0.0 and _counter0 == 0):
89              _counter0+=1
```

```
90              _targetPoints.append(_position[i,:])
91
92          if(_temp == pow(_R1,2) and _counter1 == 0):
93              _counter1+=1
94              _targetPoints.append(_position[i,:])
95
96          if(_temp == pow(_R2,2) and _counter2 == 0):
97              _counter2+=1
98              _targetPoints.append(_position[i,:])
99
100         if(_temp == pow(_R3,2) and _counter3 == 0):
101             _counter3+=1
102             _targetPoints.append(_position[i,:])
103
104         if(_temp == pow(_R4,2) and _counter4 == 0):
105             _counter4+=1
106             _targetPoints.append(_position[i,:])
107
108     return np.array(_targetPoints)
109
110
111 def FindRadius(_xLimit):
112
113     #generirajmo nasumicne kolicine gibanja za x i y smjer
114     p = 10 #ukupna kolicina gibanja iznosi 10GeV
115     px = random.uniform(0,_xLimit)
116     py = sqrt(p*p-px*px)
117
118
119     #definirajmo vektore polja, brzne i polozaja
120     E = np.array([0,0,0])
121     B = np.array([0,0,Bz])
122     v = np.array([momentum_to_speed_electron(px),↩
            momentum_to_speed_electron(py),0])
123     r = np.array([0,0,0])
124
125     #radijusi detektora
126     R1 = 0.0001
127     R2 = 0.00015
128     R3 = 0.0002
129     R4 = 0.00025
130
131     #position=Euler(1e-14,3000,r,v,E,B)
132     position=RK4(1e-14,1000,r,v,E,B)
133
134     # print('\nPositions:\n', position.real)
135
```

```
136     targetPoints = IsOnCircle(position, R1, R2, R3, R4)
137 
138     # print('\nIntersections: \n', targetPoints.real)
139 
140     #trazenje radijusa preko matrica
141     A = np.array([ [-2*targetPoints[1,0], -2*targetPoints[1,1]] , [-2*↩
            targetPoints[2,0], -2*targetPoints[2,1]] ])
142     B = np.array([-1*pow(targetPoints[1,0],2)-1*pow(targetPoints[1,1],2), ↩
            -1*pow(targetPoints[2,0],2)-1*pow(targetPoints[2,1],2)])
143     X = np.linalg.inv(A).dot(B)
144 
145     x0 = X[0]
146     y0 = X[1]
147 
148     R = sqrt(X[0]*X[0]+X[1]*X[1]).real
149 
150     #trazenje radijusa preko kopleksnih br
151     # x, y, z = complex(targetPoints[1,0].real,targetPoints[1,1].real), ↩
            complex(0,0), complex(targetPoints[2,0].real,targetPoints[2,1].↩
            real)
152     # w = z-x
153     # w /= y-x
154     # c = (x-y)*(w-abs(w)**2)/complex(0,2)/w.imag-x
155 
156     # R = abs(c+x)
157 
158     # print('\np , q:', X.real)
159 
160     # print('RADIUS:  ', R, '\n\n')
161 
162     return R
163 
164 #racunamo analiticki radijus
165 R_gen = (m*momentum_to_speed_electron(10))/(e*Bz)
166 R_gen = R_gen.real
167 #print('\nR analiticki:', R_gen, '\n')
168 
169 #racunamo (analiticku) kolicinu gibanja
170 p_gen = e*Bz*R_gen    #nece bit 10Gev
171 p_gen = p_gen.real
172 
173 import multiprocessing
174 
175 def thread_function(index):
176     #Radii = []
177     Momenta = []
178 
```

```
179     for i in range(1,300):
180         R_reco = FindRadius(0.0000001*i)
181         p_reco = e*Bz*R_reco
182         #Radii.append(R_reco/R_gen)
183         Momenta.append(p_reco/p_gen)
184     f = open("plot_data"+str(index)+".txt","w")
185     #for val in Radii:
186     for val in Momenta:
187         f.write(str(val)+"\n")
188     f.close
189 
190 
191 def FileToArray(file_name):
192     array_from_file = []
193     with open(file_name,"r") as f:
194         for line in f:
195             array_from_file.append(float(line[:-1]))
196     return array_from_file
197 
198 m1 = multiprocessing.Process(target=thread_function, args=(1,))
199 m2 = multiprocessing.Process(target=thread_function, args=(2,))
200 m3 = multiprocessing.Process(target=thread_function, args=(3,))
201 m4 = multiprocessing.Process(target=thread_function, args=(4,))
202 
203 m1.start()
204 m2.start()
205 m3.start()
206 m4.start()
207 
208 m1.join()
209 m2.join()
210 m3.join()
211 m4.join()
212 
213 # allRadii = []
214 # allRadii = FileToArray("plot_data1.txt")+FileToArray("plot_data2.txt")+↩
    FileToArray("plot_data3.txt")+FileToArray("plot_data4.txt")
215 
216 # print(allRadii)
217 
218 allMomenta = []
219 allMomenta = FileToArray("plot_data1.txt")+FileToArray("plot_data2.txt")+↩
    FileToArray("plot_data3.txt")+FileToArray("plot_data4.txt")
220 
221 #print(allMomenta)
222 
223 
```

```
224 #racunanje broja slucajeva
225 counter1 = 0
226 counter10 = 0
227 counterELSE = 0
228
229 for i in range(0, len(allMomenta)):
230     if allMomenta[i] <= 1.01:
231         counter1 +=1
232     elif allMomenta[i] <= 1.1:
233         counter10 +=1
234     else:
235         counterELSE +=1
236
237 #print(counter1*100/len(allMomenta),counter10*100/len(allMomenta),↩
    counterELSE*100/len(allMomenta))
238
239 perc1 = round(counter1*100/len(allMomenta),2)
240 perc10 = round(counter10*100/len(allMomenta),2)
241 percELSE = round(counterELSE*100/len(allMomenta),2)
242
243 string1 =   "Slucajevi unutar 1%: " + str(perc1)
244 string10 =  "Slucajevi unutar 10%: " + str(perc10)
245 stringELSE =   "Slucajevi izvan 10%: " + str(percELSE)
246
247 print(string1 + '\n' + string10 + '\n' + stringELSE)
248
249
250 #histogram
251 import matplotlib.pyplot as plt
252
253 #_ = plt.hist(np.array(allRadii), bins=60, color='orange')
254 _ = plt.hist(np.array(allMomenta), bins=60, color='orange')
255 plt.title("Histogram")
256 #plt.xlabel('R_{RECO}/R_{GEN}')
257 plt.xlabel('p_{RECO}/p_{GEN}')
258 plt.ylabel('N')
259 plt.show()
260
261
262
263 # #graficki prikaz putanje
264
265 # import matplotlib.pyplot as plt
266
267 # fig, ax = plt.subplots()
268
269 # #imenovanje osi
```

```
# ax.set_xlabel('x')
# ax.set_ylabel('y')

# ax.set_aspect(aspect=1)

# ax.plot(position[:, 0], position[:, 1], linewidth=1.0, color='orange')


# circle1 = plt.Circle((0, 0), 0.0001, color='r', fill=False)
# circle2 = plt.Circle((0, 0), 0.00015, color='r', fill=False)
# circle3 = plt.Circle((0, 0), 0.0002, color='r', fill=False)
# circle4 = plt.Circle((0, 0), 0.00025, color='r', fill=False)

# THEcircle = plt.Circle((x0, y0), R, color = 'g', fill=False )

# ax.add_patch(circle1)
# ax.add_patch(circle2)
# ax.add_patch(circle3)
# ax.add_patch(circle4)
# ax.add_patch(THEcircle)

# plt.show()
```