Poročilo projekta - Aproksimacija ploščine in obsega konveksne lupine

Nina Velkavrh in Ajda Majhenič

26.11.2021

Uvod - Definicija konveksne lupine

Podmnožica T v $\mathbb{R}^2$ je konveksna, če lahko poljubni točki A in B v T povežemo z daljico, ki je v celoti vsebovana v množici T.

Naj bo S končna množica točk v ravnini. Konveksna lupina je najmanjša konveksna množica, ki vsebuje S.

Z drugimi besedami mejo konveksne lupine tvori konveksni poligon, katerega ogljišča so točke množice S, robovi pa so odseki, ki povezujejo pare točk množice S. Označimo konveksni mnogokotnih s $CH(S)$.

Opis problema

V projektni nalogi sva se ukvarjali z zelo pogostim problemom geometrije in sicer konstruiranje konveksne lupine s končno množico točk v ravnini. Osredotočili sva se predvsem na izračun njene ploščine in obsega. Predpostavljali sva, da imamo konveksno lupino množice P, znotraj katere je n točk. Sprva sva implementirali znan Jarvisovega algoritem za določitev konveksne lupine množice S z n točkami, nato sva izračunali eksakten obseg in ploščino. V glavnem delu projekta sva modelirali 2 metodi za njuno aproksimacijo. Dobljene ploščine in obsege, sva med sabo primerjali in zelo hitro ugotovili, da je 2. metoda precej slabša kot 1., saj je le ta vračala precej večje napake kot 1.. Doblejene rezultate sva primerjali tudi z dejanskimi vrednostmi obsega in ploščine. Zanimalo naju je koliko elementov v vzorcu (podmnožici množice S) potrebujeva, da dobiva približek z natančnostjo 90% , 99% in 99,9% ter ali ima pri tem število točk kakšno vlogo.

Opis algoritma za določitev konveksne lupine množice S z n točkami

Ob začetku najnega projekta sva v najin program implementirali zelo znan in intuitive algoritem za določanje konveksne lupine imenovan Jarvisov algoritem. Ta algoritem je poznan tudi po imenu algoritem zavijanja daril, saj se premika od enega oglišča konveksne lupine do drugega kot, da ovijamo kos papirja okoli množice točk.

Uporabljen algoritem deluje po naslednjem postopku:

- Začnemo s tem, da najdemo točko p_1 iz množice S, ki je oglišče konveksne lupine. Za p_1 vzamemo najnižjo točko množice S. V kolikor množica S vsebuje več točk z minimalno y koordinato, vzamemo najbolj levo točko med njimi (točko z najmanjšo x koordinato).
- Predpostavili smo, da je p_1 oglišče konveksne lupine, posledično mora obstajati tudi taka točka p_2 iz množice S, da je tudi p_2 oglišče konveksne lupine in da je p_1p_2 rob konveksne lupine. Točki, ki bi ustrezali omenjenemu pogoju sta dve. Za p_2 izberemo tisto točko, ki naredi p_1p_2 za rob, ki se bo gibal v smer urinega kazalca.
- Kako najdemo p_2?

Naj bo l_1 vodoravna premica skozi p_1. Potem je p_2 prva točka, ki jo "zadanemo", če vrtimo l_1 okrog p_1 v smer urinega kazalca. Naj bo α_q kot med l_1 in premico p_1q za vsak $q \in S \setminus \{p_1\}$. Potem je α_q kot po katerem moramo vrteti l_1 okrog p_1 v smeri urinega kazalca, dokler ne "zadanemo" p_1q. Upoštevamo, da je $0 \leq \alpha_q \leq 2\pi$. Torej je p_2 tista točka množice $S \setminus \{p_1\}$, za katero je ta kot minimalen. Če obstaja več točk, za katere je kot α minimalen, potem je p_2 tista izmed teh točk, ki ima maksimalno razdaljo od p_1.

Torej lahko za dano točko p_1 najdemo naslednje oglišče konveksne lupine p_2 tako, da preverimo vse točke $q \in S \setminus \{p_1\}$ in izberemo tisto, za katero je kot α_q minimalen.

- Nadaljujemo seveda po enakem postopku: Naj bo l_2 premica skozi točki p_1 in p_2. Za vsako točko $q \in S \setminus \{p_2\}$ naj bo α_q kot med l_2 in premico p_2q. (Sedaj je α_q kot po katerem moramo vrteti l_2 okrog p_2 proti smeri urnega kazalca, dokler ne "zadanemo" p_2q.) Potem je p_3 - naslednje oglišče konveksne lupine - tista točka, katere kot α je minimalen.

Nadaljujemo z izračunavanjem oglišč p_4, p_5, ..., dokler se ne vrnemo v p_1. Bolj natančno: Če $p_{h+1} = p_1$, smo končali in imamo seznam oglišč kompleksne lupine $(p_1, p_2, ..., p_h)$.

Po imlplementaciji zgoraj opisanega algoritma sva definirali obseg in ploščino. Obseg sva izračunali tako, da sva računali razdalje med točkami konveksne lupine in vrednosti med seboj seštevali. Ploščino pa sva izračunali tako, da sva implementirali znano formulo za določitev ploščino poljubnega lika imenovano Shoelace formula.

Seveda sva za določitev konveksne lupine, izračun ploščine in obsega potrebovali množico S z n točkami. Zato sva definirali naključno množico, ki sva ji določili parametre a, b in qty. V implementaciji Jarvisovega algoritma sva že definirali točko, zato sva tu potrebovali le še naključno izbiro za koordinato x in y. Parameter a določi, da koordinate x izbiramo iz intervala [o,a], parameter b pa, da koordinato y izbiramo iz območja [0,b]. Parameter qty določi koliko naključnih točk iz območja [0,a]x[0,b] želimo. Sicer sva tu definirali 2 parametra za določitev območja, a sva kasneje v algoritmu za aproksimacijo ploščine in obsega uporabljali le še kvadrate, torej bi lahko definirali tudi samo parameter a.

Kasneje sva se lotili še glavnega dela projektne naloge. Za algoritem, ki bi lahko aproksimiral ploščino in obseg območja konveksne lupine sva imeli 2 ideji. Oba algoritma sta spodaj podrobneje opisana.

Opis 1. metode

Pri prvi metodi začnemo s praznim seznamom pravokotnikov. Privzamemo, da je vrednost a/m dolžina, vrednost b/m pa višina celice mreže na katero razdelimo začetno območje. Slednje nam torej območje [0, a]x[0, b] razdeli na pravokotnike (kvadrate) velikosti m x m. Iz vsakega dobljenega pravokotnika (celice) izberemo kvečjemu eno točko (če je v pravokotniku več točk množice S, naključno izberemo eno izmed njih, če pa v njem ni nobene točke iz množice S, potem ta pravokotnik »preskočimo«). Izbrane točke zbiremo v seznam, ki sedaj predstavlja podmnožico osnovne množice S in v najnem alogritmu en vzorec. Dobljenemu vzorcu (podmnožici) določimo konveksno lupino, njeno ploščino in obseg.

Opis 2. metode

Najina druga ideja za algoritem, ki bi aproksimiral območje konveksne lupine je bila, da bi konveksno lupino aproksimirali z najmanjšim krogom, ki bi vseboval vse točke množice S. Torej pri temu algoritmu, bi en vzorec predstavljal celotno množico S. Omenjena ideja se je skozi analizo podatkov izkazala za precej slabo. V algoritmu najprej določimo težišče (povprečje x in y koordinat) vzorca množice S. (Vzamemo vse x koordinate točk iz množice S in izračunamo njihovo povprečje. Enako ponovimo za y koordinate. Dobimo torej povprečje x in y koordinat.) Nato določimo še polmer kot dolžino od težišča do najbolj oddaljene točke množice S. Definiramo ploščino in obseg kroga po znanih formulah.

Skozi celotno analizo podatkov sva kot omenjeno ugotovili, da je izbira vzorca celotna množica S precej slaba, zato sva poskusili tudi ta algoritem izboljšati in ga približati točnosti prvega algoritma. Zato sva definirali

podmnožico naključne množice, ki iz osnovne množice S vzame k tock.Po izbiri podmnožice algoritem deluje kot zgoraj opisano, le da izbere težišče podmnožice in polmer kot razdaljo najbolj oddaljene točke od težišča iz podmnožice.

Statistično modeliranje v programu r

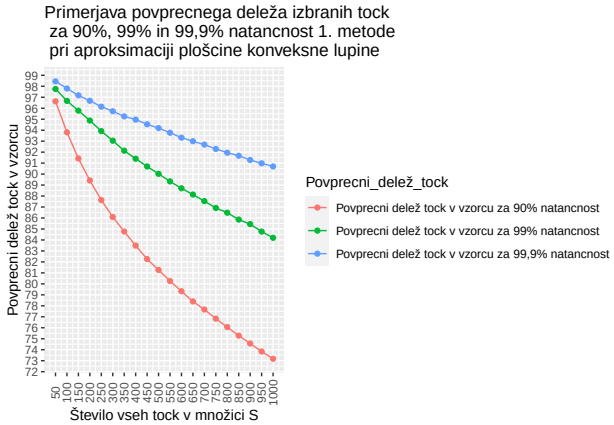
Obdelava podatkov v programu r Podakte sva uvozili v program r in jih statistično obdelali. Med analiziranjem rezultatov naju je zanimalo:

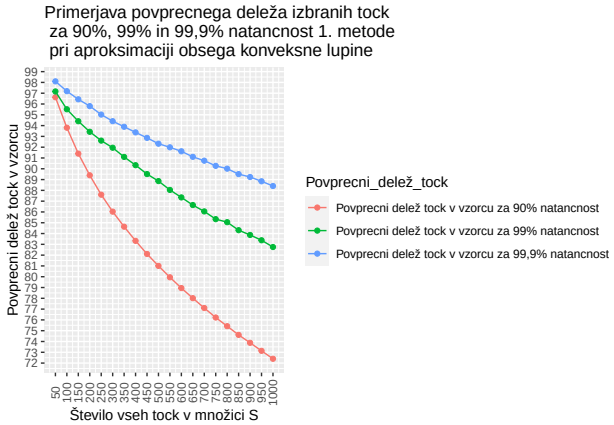
- Koliko je povprečni delež točk v vzorcu 1. modela, ki ga potrebujeva za 90% , 99% in 99,9% natančnost pri različnih velikostih osnovne množice S,
- kako na rezultate modela 1.vpliva različna razdelitev območja pri fiksnem številu točk,
- kako se razlikujejo rezultati posamezne metode glede na različno število točk v množici S,
- ali in kdaj je mogoče tudi z 2. metodo dobiti 90% , 99% in 99,9% natančnost,

Rezultati:

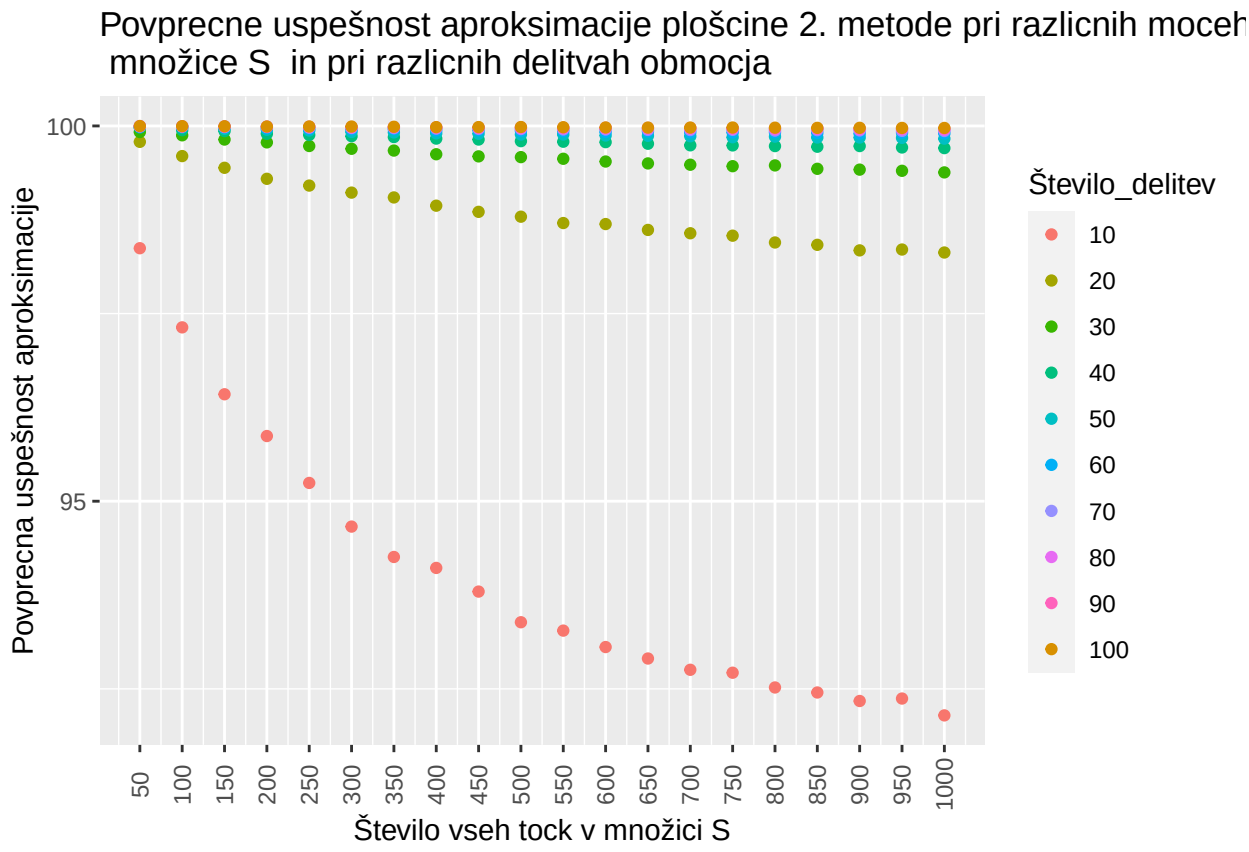
Povprečni delež točk za 90% , 99% in 99,9% natančnost 1. modela pri izračunu ploščine in obsega pri različnih močeh množice S Ob začetku analize podatkov sva si odgovorili na vprašanje kakšen delež točk je potreben za 90% , 99% in 99,9% natančnost 1. modela. Program je zgeneriral točke na območju [0,10] x [0,10], pri 100 delitvah enega intervala - torej mreža 10000 kvadratov, pri različnih močeh množice S. Sprva sva podatke generirali tako, da se je za vsako moč množice S algoritem ponovil stokrat, ker sva pri takem številu ponovitev ugotovili, da obstajajo odstopanja pri deležih, sva se odločili, da algoritem poženeva raje tisočkrat za vsako moč množice S. Generiranje podatkov je sicer trajalo kar nekaj časa, a sva kasneje z analizo ugotovili, da je tovrstn način bolj reprezentativen. Rezultate zadnjega poskusa prikazujeta spodja grafa. Opazi se, da funkciji nimata nenavadnih odstopanj. To, da ne opazimo odstopanj se zdi logično, saj se pri večjem številu iteracij izračunana povprečna vrednost deležev točk, ki je potrebna za željeno natančnost bolj približuje pravi povprečni vrednosti. Iz spodnjih grafov je moč opaziti, da se povprečni delež točk potrebnih za željeno natančnost, zmanjšuje z večanjem moči množice S. Videno je verjetno posledica tega, da je območje glede na količino točk v večjih množicah precej majhno in so posledično točke zelo gosto razporejene po območju. Kot omenjeno model 1 iz vsakega dela mreže na katero je razdeljen kvadrat [0,10] x [0,10], vzame le 1 točko oziroma nobene, če celica nima točke množice S. Posledica tega je, da so neizbrane točke verjetno zelo blizu izbranim in se rezultati aproksimacije ne razlikujejo bistveno od eksaktnih vrednosti ploščin in obsega.

Grafa prikazujeta potrebne deleže točk za željene natančnosti aproksimacije obsega in ploščine. Vidimo lahko, da so potrebni deleži za obseg le nekaj odstotnih točk manj kot potrebni deleži za ploščino.





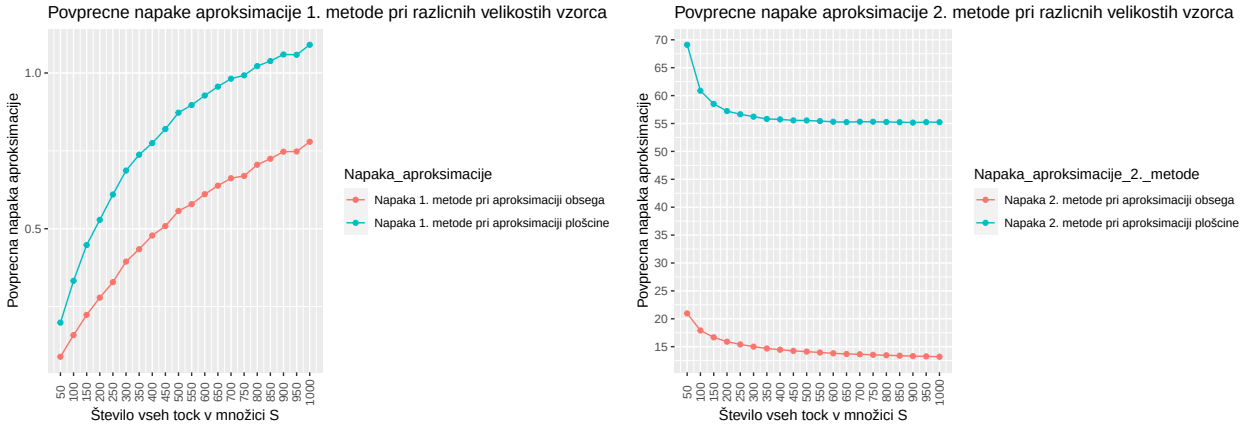
Vpliv razdelitve območja na rezultate 1. metode Zelo pomemben del 1. metode je to, da razdelimo območje na več delov. Posledično naju je zanimalo, kako delitev območja dejansko vpliva na rezultate metode pri različnih močeh množice S. Ponovno sva vzeli območje [0,10] x [0,10] in ponovili algoritem tisočkrat za vsako moč množice S. Ugotovitve sva prikazali na spodnjem grafu. Vidimo lahko, da pri delitvi na 10 delov območja [0,10] x [0,10], povprečna natančnost aproksmacije ploščine pri večanju moči množice S pada, podobno na rezultate vpliva delitev na 20 in 30. Kasneje, ko delimo območje še na več delov pa opazimo, da se rezultati pri različnih delitvah ne razlikujejo več toliko. Torej bi lahko rekli, da od delitve 40 naprej pri takem območju, z povečanjem delitev ne izboljšujemo natančnosti algoritma. V programu r sva si narisali tudi graf, ki predstavlja povprečne napake aproksimacije obsega pri različnih delitvah, a sta si grafa med seboj zelo podobna, zato v poročilo nisva vključili obeh.



Primerjava rezultatov prve različice 1. metode in 2. metode pri različnih močeh množice S Spodnja grafa prikazujeta povprečne rezultate obeh metod pri tisočih ponovitvah metod za vsako moč množice S. Tudi pri spodnjih grafih sva sprva poskusili algoritem ponoviti le stokrat za vsako moč množice, a je prišlo tudi tu do precejšnih odstopanj. Razlog odstopanj je podoben kot pri izračunu potrebnih deležev. Torej, da se pri večjem številu ponovitev modela izračunana povprečna vrednost napake za vsako moč množice bolj približuje pravi napaki. Opazimo lahko, da so napake prve različice 2. metode bistveno večje, kot napake 1. metode. Razlog za to je, da prva različica 2. metode vzame za polmer kroga točko, ki je najbolj oddaljena od težišča (povprečja koordinat točk), ta razdalja je lahko v primeru, da je večina točk zgoščena okoli neke vrednosti in le majhen delež točk popolnoma drugje, precej velika. Ta majhen delež točk, ki se nahaja stran od večine ne prispeva veliko pri izračunu povprečja koordinat, zato se težišče nahaja bližje območju kjer je večji delež točk množice S. Posledično bo krog vzel precej večje območje, kot je konveksna lupina te množice in bojo napake aproksimacije zelo velike. Podoben razmislek, kot smo ga uporabili pri razlagi, zakaj je za večje množice potreben manjši delež točk za željeno točnosti, lahko uporabimo tudi tu, ko opazimo, da se napake prve različice 2. metode manjšajo z večjim številom točk množice S. Na območju

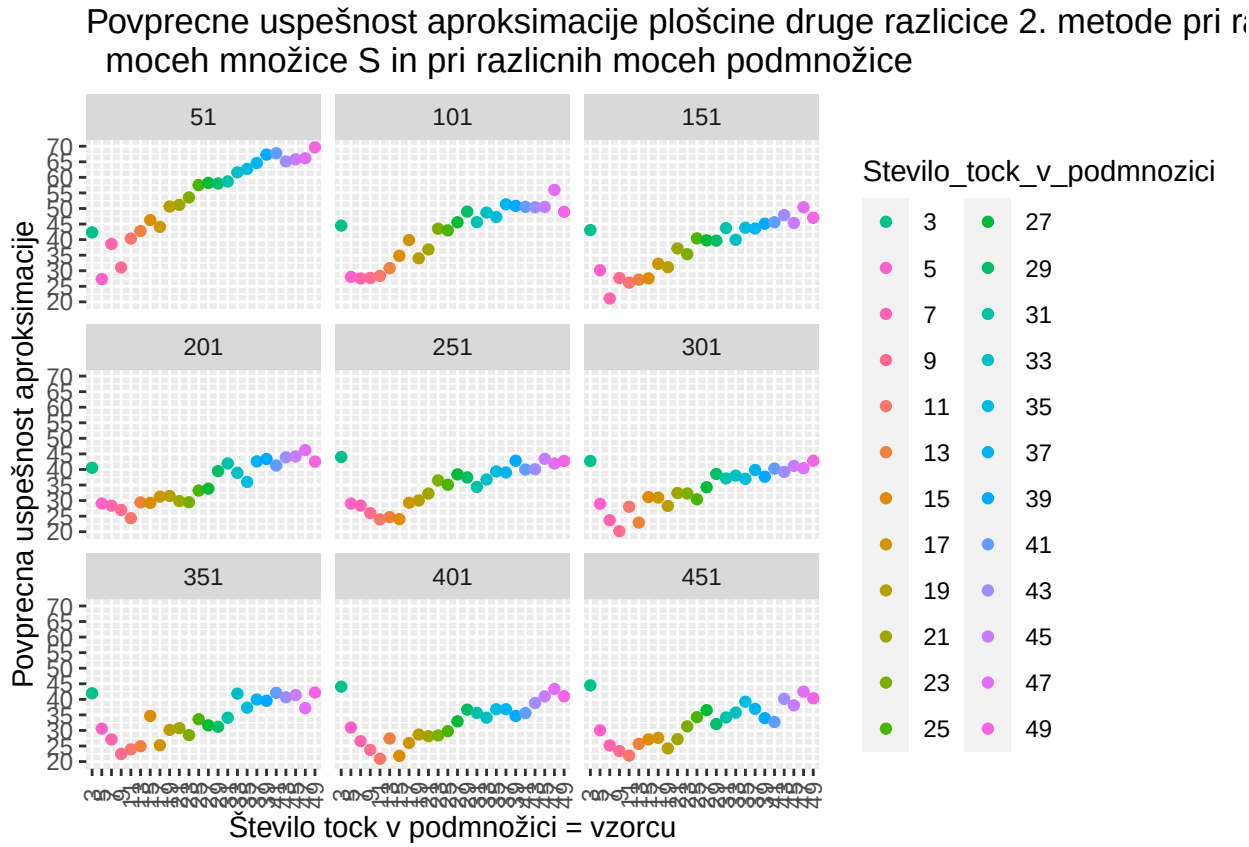
[0,10] x [0,10] je namreč zelo verjetno, da bojo točke večjih množic bolj enakomerno porazdeljene po območju in bo krog posledično manj odstopal od dejanske konveksne lupine. Zanimiv podatek, ki sva ga opazili je, da na neki točki napaka ne pade pod 55% za ploščino in 10% za obseg. Razlog tega je verjetno ta, da se z večanjem vzorca povečuje verjetnost, da bo najbolj oddaljena točka od težišča blizu roba, težišče pa blizu sredini kvadrata in bo s tem krog očrtan krog kvadrata.

Pri grafu, ki prikazuje povprečne napake 1. metode lahko opazimo, da so napake med 0 in 0.15, a se z večanjem vzorca povečujejo. Slednje se dogaja zato, ker je pri izbiri točk pri manjših močeh množice bolj verjetno, da so izbrane točke ravno točke, ki sestavljajo konveksno lupino. Medtem, ko gostota razporeditve točk večjih množic S vpliva na to, da je verjetnost, da izberemo ravno točke ovojnice manjša.



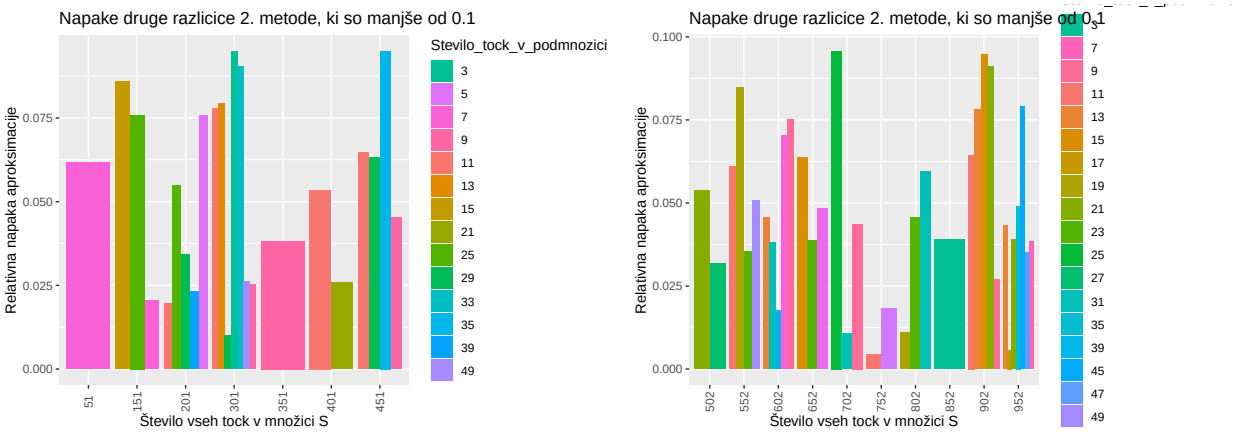
Seveda z rezultati, ki jih predstavljata zgornja grafa nisva bili zadovoljni in sva tudi hitro ugotovili, da ideja, da za vzorec vzamemo celotno množico S ni najboljša. Algoritem sva izboljšali, tako da sva 2. metodo implementirali na podmnožici množice S. Spodaj so prikazani rezultati, ki sva jih ob analiziranju te različice dobili.

Relativne napake aproksimirane ploščine pri drugi različice 2. metode pri različnih močeh množice S in pri različnih močeh podmnožice Ob novi različici 2. metode naju je najprej zanimalo, kako različno število točk podmnožice vpliva na rezultate metode. Ponovno sva algoritem ponovili tisočkrat za vsako moč množice S na območju [0,10] x [0,10]. Iz spodnjih grafov lahko vidimo, da so napake, v kolikor izberemo preveč točk v podmnožici S, podobno visoke kot pri prvi različici. Razlog tega je verjetno to, da je lahko pri velikem številu točk polmer kroga ponovno zelo velik, zaradi gostote porazdelitve točk in najbolj oddaljene točke. Hkrati pa lahko pri določenih močeh podmnožice povprečno dosegamo boljše rezultate, kot smo jih dosegali pri prvi različici 2. metode, ko smo računali ploščino in obseg kroga, ki zajame vse točke množice S.

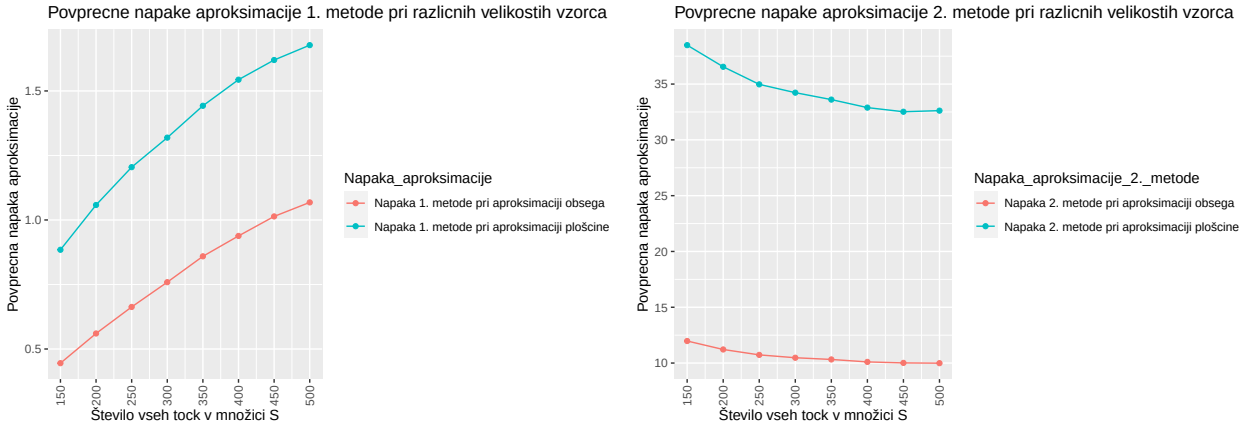


Najboljše uspešnosti 2. metode pri aproksimaciji ploščine

Zanimalo naju je tudi ali lahko druga različica 2. metode kdaj dosega napake aproksimacije pod 0.1. Z analizo sva ugotovili, da so napake v posameznih primerih res lahko zelo majhne. Primeri, ko se pri različnih močeh množice S zgodi, da je napaka aproksimacije manj kot 0.1 so prikazani spodaj. Vendar se nama zdi, da so ti posamezni primeri zelo naključni in iz njih ne moremo sklepati, kaj bi bilo optimalno število točk v podmnožici za doseganje najboljših rezultatov.



Primerjava rezultatov druge različice 1. metode in 2. metode pri različnih močeh množice S



Zaključek in možne izboljšave

Obe prikazani metodi dobro delujeta, vendar sta zelo časovno zahtevni. Na rezultate pri obeh algoritmih sva čakali dolgo časa, kar potrjuje, da je algoritem precej počasen. Poleg izboljšave algoritma na področju časovne zahtevnosti bi bilo zanimivo pogledati, kako bi se na rezultatih prve metode izražalo to, da bi območje razdelili na dejanske pravokotnike in ne na kvadrate. Poleg tega pa bi bilo zanimivo videti rezultate druge različice 2. metode pri vplivu na izbiro točk v podmnočici. Zdi se nama, da je druga različica 2. metode še vedno preveč odvisna od naključne izbire točk v podmnožico. Zato so lahko rezultati tudi pri enakih močeh podmnožice popolnoma različni.