UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika – 2. stopnja

Anja Plesec

Gradientni spust

Končno poročilo pri predmetu Matematika z računalnikom

Mentor: prof. dr. Sergio Cabello Justo,
asist. Gašper Domen Romih

Ljubljana, 2022

1 Uvod

V projektni nalogi predstavim metodo gradientnega spusta. Na enostaven način bi lahko to metodo opisala takole: stojimo na vrhu hriba in si želimo priti do doline. Ker hriba ne poznamo se odpravimo po poti navzdol, ki je najbolj strma. Na začetku delamo velike korake, ko pa pot postaja bolj položna začnemo delati manjše korake, saj bomo kmalu prispeli na cilj - v dolino. Na vsakem koraku je naslednji korak odvisen od lokacije, kjer se nahajamo in strmine hriba. Na isti način deluje tudi metoda gradientnega spusta, ki jo bom predstavila skozi nalogo. Projekt sem izdelala v programu Matlab, kjer sem si naprej pogledala uporabo gradientnega spusta v primeru linearne regresije, nato pa na funkcijah ene in več spremenljivk.

2 Teoretična izhodišča

Gradientni spust je iterativna metoda za iskanje lokalnega minimuma funkcije, ki se lahko uporablja na različnih področjih. Ideja metode je, da najdemo minimum funkcije:

$$\min_{x \in \mathbb{R}^n} f(x),$$

kjer predpostavimo, da je funkcija f konveksna in diferenciabilna v vsaki točki. Naj obstaja optimalna točka y^* in predpostavimo, da obstaja tak x^*, da je $f(x^*) = y^*$.

Splošen algoritem:

1. Izberemo začetno točko $x_0 \in \mathbb{R}^n$.
2. Za $t \geq 0$ predpostavimo, da že poznamo $x_0, ..., x_t$. Člen x_{t+1} izračunamo kot linearno kombinacijo x_t in $\nabla f(x_t)$.
3. Algoritem se konča, ko je dosežen eden izmed kriterijev in vrne rezultat zadnje iteracije.

V zgornjem algoritmu x_t predstavlja točko, ki smo jo izračunali v drugem koraku tega algoritma v n-ti iteraciji. Na tem mestu se pojavi vprašanje zakaj se potem ta algoritem imenuje gradientni spust. To ime je povezano z drugim korakom, torej z izračunom točke x_{t+1}. Točke se pomikajo v smeri **najbolj strmega spusta**. Radi bi izbrali vektor u, ki

$$\max_{||u||=1} \left[\lim_{\delta \to 0} \frac{f(x) - f(x + \delta u)}{\delta} \right].$$

Izraz znotraj oklepaja je pravzaprav limita negativnega odvoda funkcije f v točki x, kar lahko zapišemo kot:

$$\max_{||u||=1} \left(-\frac{\partial f}{\partial u}(x) \right) = \max_{||u||=1} \left(-\langle f(x), u \rangle \right).$$

Maksimum je dosežen pri:

$$u^* := -\frac{\nabla f(x)}{||\nabla f(x)||}.$$

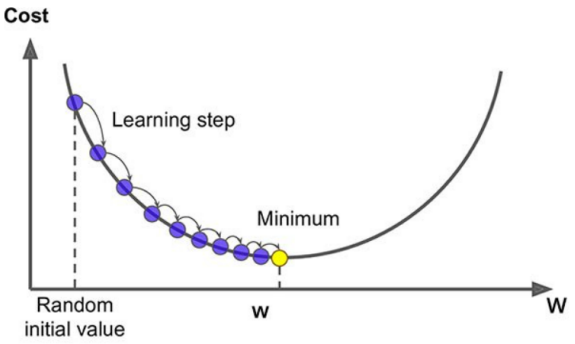
Iz tega sledi, da je naravna oz. Eulerjeva diskretizacija sledeča:

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x)}{||\nabla f(x)||},$$

kjer je $\alpha > 0$ dolžina koraka - kako daleč od u^* se premaknemo. Faktor $\frac{1}{||\nabla f(x)||}$ lahko izpustimo in dobimo:

$$x_{t+1} = x_t - \eta \nabla f(x),$$

kjer je $\eta > 0$ parameter, ki se imenuje **učna stopnja** (ang. learning rate). Izpeljano formulo upoštevamo v drugem koraku algoritma in s pomočjo tega poračunamo minimum.



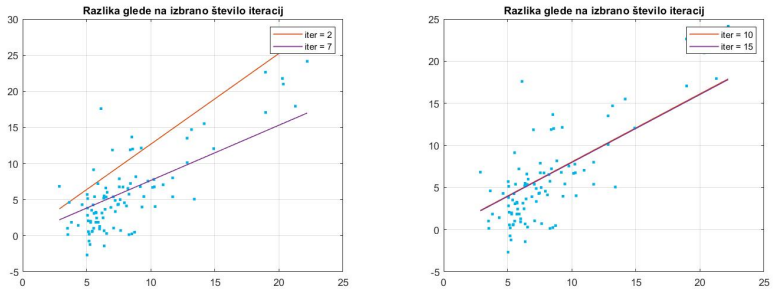
Slika 1: Prikaz delovanja metode gradientnega spusta

Težave, ki se pri tem algoritmu pojavijo, so ustrezna izbira vhodnih podatkov. Eden izmed teh je učna stopnja, saj bi radi delali velike korake, kar bi pomenilo manjše število iteracij, vendar se nam lahko zgodi, da s tem zgrešimo našo rešitev in sam algoritem ne vrne prave rešitve, če pa delamo premajhne korake pa je potrebno veliko število iteracij, da pridemo do rešitve.
Naslednji izmed vhodnih podatkov je začetna točka x_0. Želimo si jo izbrati tako, da je čim bližje optimalni rešitvi, saj bomo zaradi tega potrebovali manj iteracij.

3 Primer uporabe pri linearni regresiji

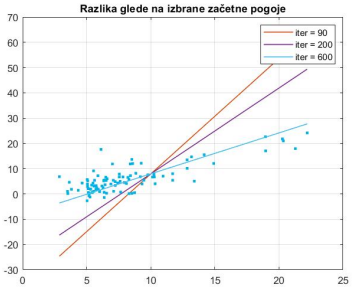
Za začetek obravnave algoritem gradientnega spusta sem si pogledala linearno regresijo. Podatki, ki sem jih uporabila so izmišljeni. Opazovala sem predvsem kako se algoritem odziva ob spreminjaju vhodnih podatkov.

Vemo, da gradientni spust doseže optimalno vrednost tako, da na začetku algoritma dela velike korake, ko pa se bliža optimalni vrednosti (minimumu) pa so koraki vedno manjši. To lastnost algoritma lahko potrdi slika 2. Na prvi sliki torej vidimo, da pet dodatnih iteracij zelo vpliva na premico (razlika med pet in deset iteracij), med tem ko na drugi sliki, kjer gledamo razliko med 95 in 100 iteracij, skoraj ni razlike, saj so koraki, ki jih delamo vedno manjši.

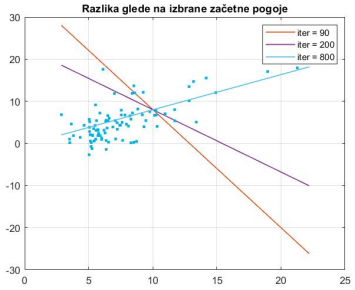


Slika 2: Razlike v številu iteracij

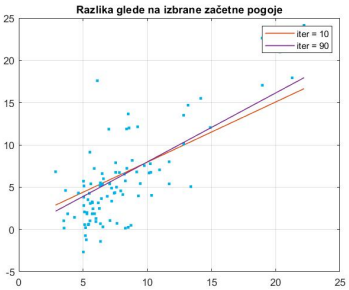
V naslednjem poskusu, sem spreminjala začetne pogoje in opazovala koliko iteracij je potrebno, da se premica približa optimalni rešitvi. V primeru a) sem za naklon vzela 20, za konstanto (začetno vrednost) pa -50. Opazimo, da algoritem potrebuje približno 600 iteracij, da se dobro prilega danim podatkom, med tem ko v naslednjem primeru, kjer je naklon enak -20, konstanta pa 50, potrebuje približno 800 iteracij. Sam vzrok takšne razlike je v napačnem predznaku začetnega naklona. V primeru c) sem vzela za začetni naklon in konstanto kar vrednost 1. Opazimo, da sta blizu optimalnemu naklonu in konstanti, zato se že pri desetih iteracijah premica dobro prilega točkam. V tem primeru je torej dobro pogledat podatke in na podlagi tega določit približen naklon in konstanto, saj bo zaradi tega potrebnih manj iteracij.



(a) naklon = 20, konst = -50



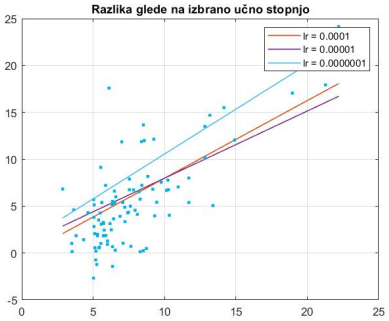
(b) naklon = -20, konst = 50



(c) naklon = 1, konst = 1

Slika 3: Razlika pri različnih začetnih pogojih

Zanimalo me je tudi, kako se sama premica spreminja pri različno izbranih učnih stopnjah. Naredila sem 100 iteracij za vsako učno stopnjo in ugotovila, da manjša je ta stopnja dlje smo od optimalne rešitve po vseh iteracijah. Kar je seveda smiselno, saj manjša kot je učna stopnja, manjše korake delamo in dlje časa potrebujemo, da se približamo optimalni premici.



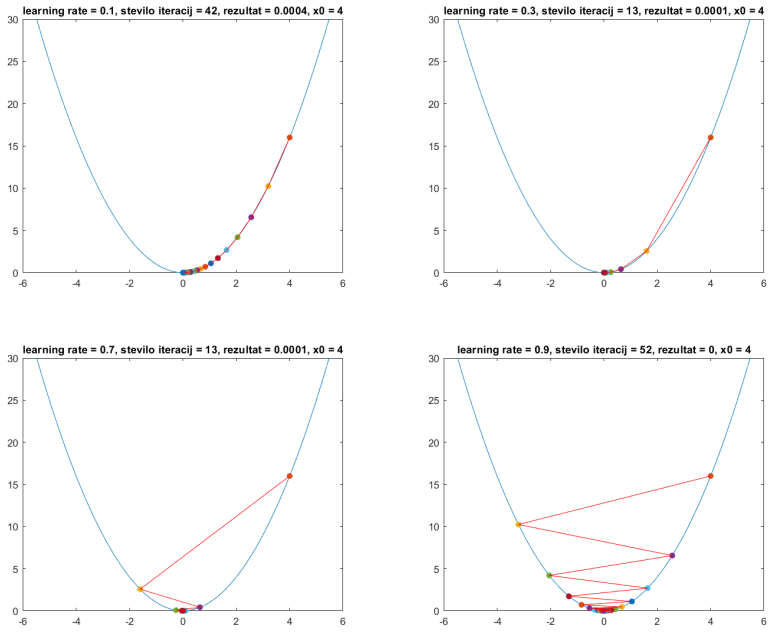
Slika 4: Razlika pri različnih učnih stopnjah

4 Iskanje minimuma podane funkcije

V tem poglavju bom predstavila iskanje minimuma funkcije ene in več spremenljivk s pomočjo metode gradientnega spusta. Pogledala sem si enostavne funkcije iz katerih se lepo vidijo prednosti in slabosti te metode.

4.1 Iskanje minimuma funkcije ene spremenljivke

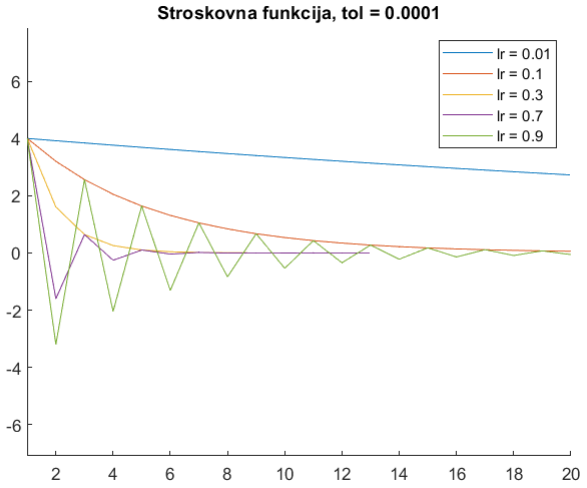
Prva težava gradientnega spusta, kot sem že omenila v teoretičnem delu, je izbira učne stopnje. Želimo si izbrati čim večjo, da bo potrebnih čim manj iteracij, ampak se nam lahko zgodi, da bomo minimum preskočili. Pogledala sem si funkcijo x^2. Iz spodnjih slik se lepo vidi, kako različne učne stopnje vplivajo na iskanje minimuma in koliko iteracij je potrebnih. Na grafu levo zgoraj, kjer je učna stopnja enaka 0.1, lahko vidimo, da po 42 iteracijah dosežemo neko vrednost dovolj blizu 0. Če za učno stopnjo izberemo 0.3 ali 0.7 potrebujemo 13 iteracij. Opazimo pa tudi da pri učni stopnji 0.7 začnejo vrednosti skakati iz ene strani minimuma na drugo stran, kar je posledica prevelike stopnje. V primeru, ko pa je učna stopnja enaka 0.9 je teh skokov še več in je vrednost dovolj blizu 0 šele pri 52 iteraciji.



Slika 5: Razlike glede na izbrano učno stopnjo

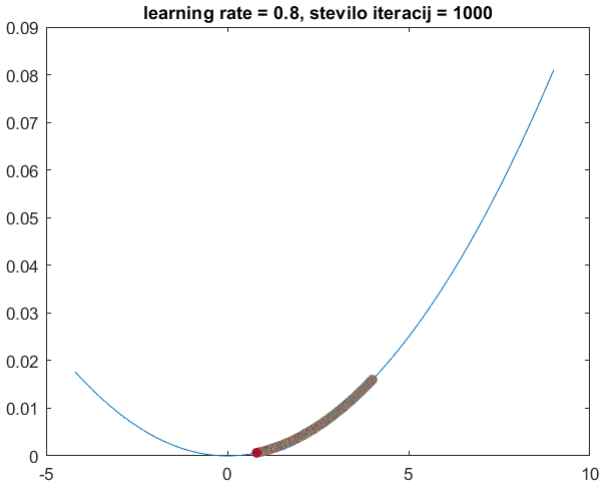
Za določitev ustrezne učne stopnje si lahko pomagamo z grafom. V primeru

funkcije x^2 sta to vrednosti 0.3 in 0.7, ki se dovolj približata ničli v trinajstih iteracijah. Torej za ustrezno učno stopnjo lahko vzamemo vrednosti med 0.3 in 0.7. Tiste, ki pa niso v tem intervalu bodo potrebovale več iteracij, da bodo dovolj blizu minimuma.



Slika 6: Izravnana funkcija

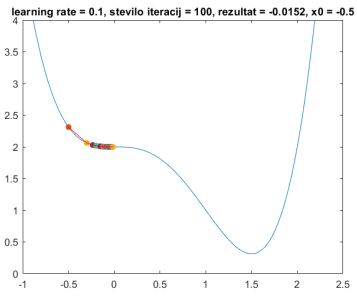
Naslednja stvar, ki sem jo opazila je, da sam algoritem ni primeren za funkcije, ki so nekaj časa skoraj konstantna oz. je naklon tangente skoraj ničelni. Pri takih funkcijah sam algoritem potrebuje veliko več iteracij, da pride do minimuma, saj dela majhne korake. Na spodnjem grafu funkcije $\frac{x^2}{1000}$ vidimo, da je kljub dobro izbrani učni stopnji končna vrednost približka po 1000 iteracijah za minimum enaka 0.8066, kar je še vedno daleč stan od samega minimuma.

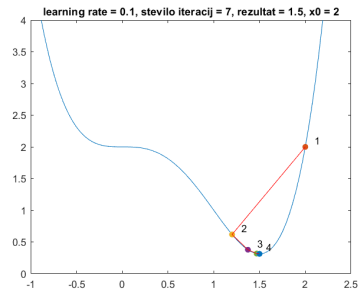


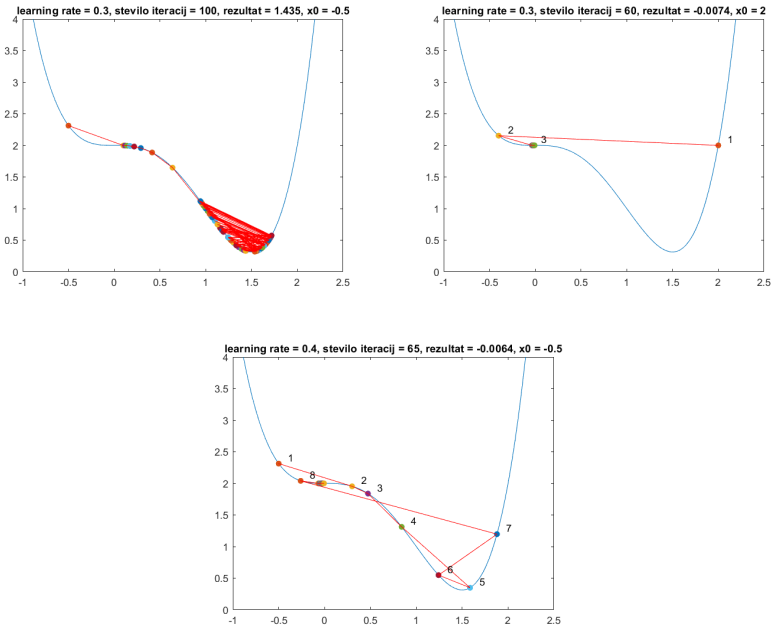
Slika 7: Izravnana funkcija

Zanimiv primer za obravnavo je naslednja funkcija: $x^4 - 2x^3 + 2$. Težave algoritmu povzroči sedlo, torej funkcija ni strogo konveksna. To je lep primer iz katerega so vidne težave algoritma. Vrednost, ki nam jo vrne metoda gradientnega spusta je zelo odvisna od izbire začetne točke in učne stopnje.

V primeru, ko za začetni približek vzamemo $x_0 = 2$, lahko opazimo, da nam pri učni stopnji 0.1 algoritem vrne globalni minimum, pri učni stopnji 0.3 pa lokalni minimum, saj delamo velike korake in s tem preskočimo globalni minimum. Prav tako je zanimiva situacija v primeru, ko za x_0 vzamemo vrednost -0.5. V primeru učne stopnje 0.1 algoritem vrne lokalni minimum, v primeru 0.3 globalni minimum, v primeru 0.4 pa obišče tako globalni kot lokalni minimum in potem obstoji v lokalnemu, ne glede na to koliko iteracij naredimo. Iz tega primera se lahko torej naučimo, da pri tem algoritmu nikoli ne vemo ali nam vrne globalni ali lokalni minimum.

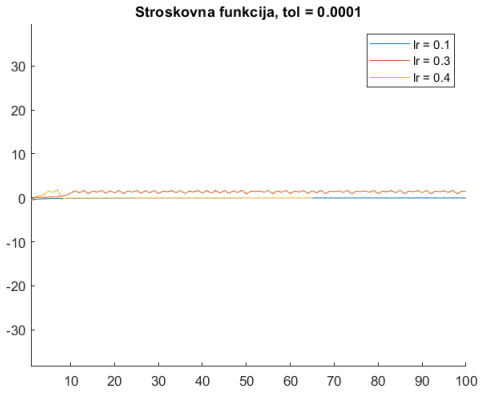






Slika 8: Razlike glede na izbrano učno stopnjo in začetni približek

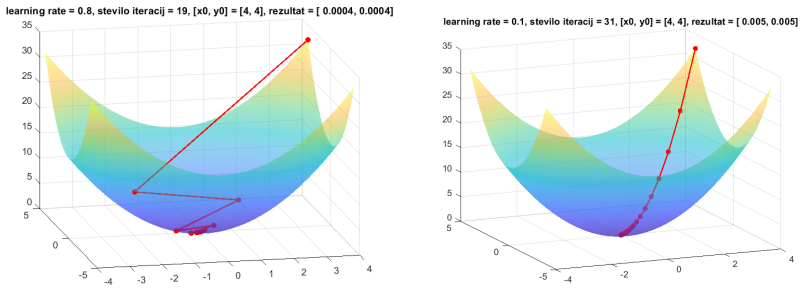
Tudi če pogledamo graf za lažje določanje ustrezne učne stopnje nam le ta vrne, da pri začetni točki $x_0 = 2$ je najbolj ustrezna učna stopnja 0.4. Ampak iz zgornjih grafov vidimo, da nam le ta ne vrne globalnega minimuma, ampak lokalni minimum.



Slika 9: Izravnana funkcija

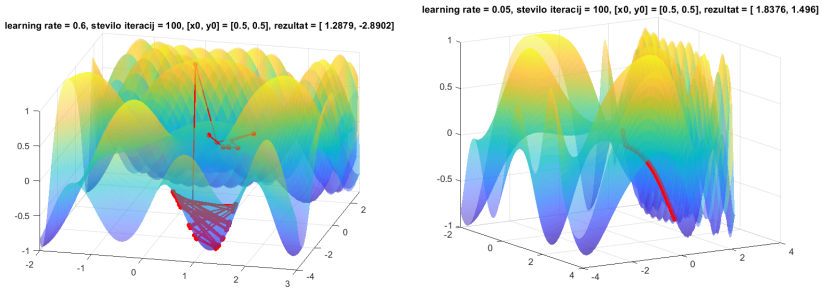
4.2 Iskanje minimuma funkcije več spremenljivk

Podobno kot pri iskanju minimuma funkcije ene spremenljivke si lahko za začetek pogledamo funkcijo $x^2 + y^2$. V primeru, ko je učna stopnja enaka 0.8 lahko opazimo, da tako x koordinate kot y koordinate skačejo iz pozitivnih v negativna števila in dosežejo minimum v 19 iteracijah, med tem ko pri učni stopnji 0.1 so vrednosti obeh koordinat pozitivne in lepo padajo k minimumu. V tem primeru pa je potrebnih 31 iteracij.



Slika 10: Razlike glede na izbran learning rate

Če si pogledamo funkcijo, ki ima sedla in več lokalnih minimumov opazimo podobne stvari, kot v primeru funkcij ene spremenljivke. Samo iskanje minimuma je zelo odvisno od izbire začetnega približka in učne stopnje. Na spodnjih grafih opazimo, da pri različnih učnih stopnjah metoda najde različne minimume. Prav tako je razlika v konvergenci, nekje lepo pada proti minimumu, nekje pa skače iz ene strani minimuma na drugo stran.



Slika 11: Razlike glede na izbran learning rate

5 Zaključek

Iz grafov in izračunov sem ugotovila, da je sam algoritem najbolj odvisen od učne stopnje in začetnega približka. Od tega je odvisno število iteracij. Sama učna stopnja vpliva tudi na konvergenco/divergenco algoritma ter na hitrost konvergence. Algoritem, kot sem že omenila, je primeren samo za konveksne in diferenciabilne funkcije, saj drgače konvergenca ni zagotovljena, kot smo lahko tudi videli v četrtem poglavju.
To projektno nalogo bi bilo smiselno tudi razširit, saj poznamo različne metode in variacije gradientnega spusta, ki so uporabne. Ena izmed njih, ki sem jo velikokrat zasledila, je gradientni spust z momentom, ki reši težavo, ko se vrednost ujame v lokalni minimum. Enačba je v tem primeru sledeča

$$x_{t+1} = \lambda x_t - \eta \nabla f(x).$$

Smiselno bi bilo tudi pogledati še za različne funkcije, tudi v višjih dimenzijah. Lahko bi tudi trigonometričnim funkcijami poiskali polinome, ki se jim najbolje prilegajo. Mislim, da je možnosti za razširitev zelo veliko.

Literatura

[1] Nisheeth K. Vishnoi (2020) *Algorithms for Convex Optimization*. Pridobljeno 24. 4. 2022 iz `https://convex-optimization.github.io/`.

[2] Robert Kwiatkowski (2021) *Gradient Descent Algorithm — a deep dive*. Pridobljeno 30. 4. 2022 iz `https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21`.