103: Nelinearna opimizacija

Peter Rupnik

24. oktober 2018

1 Thomsonov[1] problem

1.1 Naloga

Na prevodno kroglo nanesemo N enakih (klasičnih) nabojev. Kako se razmestijo po površini? Zahtevamo seveda minimum elektrostatične energije. Uporabi katero od minimizacijskih metod, npr. Powellovo ali n-dimenzionalni simpleks (amebo).

1.2 Reševanje z ugibanjem

Ko sem pripravil programsko infrastrukturo za optimizacijo energije, sem iz radovednosti naprej poskusil z zaporednim ugibanjem začetnih pozicij in opazoval, kakšne rezultate dobim. Če zaporedoma ugibamo začetne lege nabojev in si v vsaki iteraciji zapomnimo rezultat, če je manjši od prejšnjega najmanjšega, lahko načeloma z neskočno časa ali neskončno sreče[2] zadanemo minimum energije.

Za ilustracijo naivnega iskanja najnižje energije sem se omejil na 1000 poskusov in za različno število točk pogledal, kako nizko energijo sem v tem omejenem času sposoben doseči. Rezultati so na sliki 1. Poskusil sem tudi z različico, kjer dodamo bodisi 1, 5, 10 ali 50 dodatnih nabojev, rezultati pridejo zelo podobni, saj v vsaki iteraciji le ugibamo začetne lege, zaradi česar se časovna odvisnost ne povečuje. Za 1000 iteracij sem potreboval $26.2s \pm 1.07s$. Primerjava s sliko 2, kjer lahko na levi strani odčitamo minimalne energije, pokaže, da smo pri vseh sistemih daleč od optimuma.

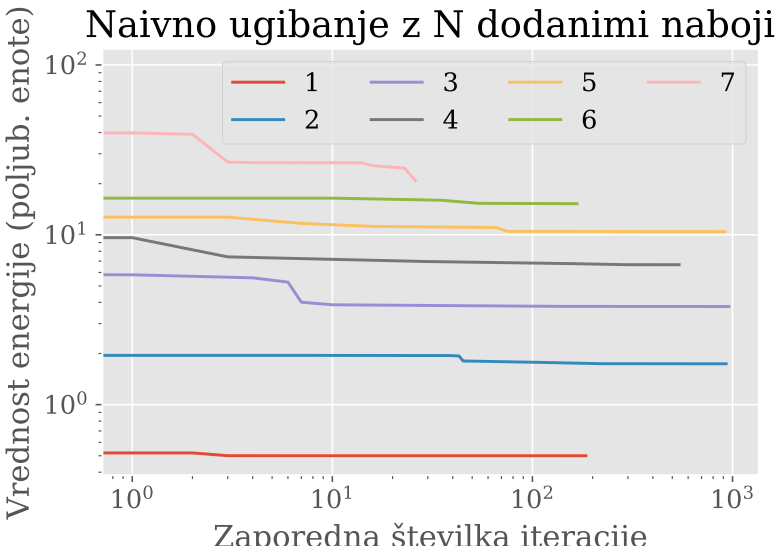
1.3 Nelinearna optimizacija

Naslednji korak je bil seveda uporaba istega mehanizma še z orodjem za minimizacijo energije. S funkcijo `scipy.optimize.minimize` sem minimiziral energijo z algoritmom *Nelder-Mead*, dobljene vektorje $[x, y, z]$ pa sem narisal na enotski krogli. Da bi se prepričal, da je konfiguracija res blizu konfiguracije, ki minimizira energijo, sem bolj kot sliko opazoval doseženo energijo in povprečje $[\langle x\rangle, \langle y\rangle, \langle z\rangle]$. Odvisno od simetrije je lahko to povprečje ničelni vektor, lahko pa ima neničelno komponento v smeri z.

Večkrat sem ponovil isti minimizacijski problem in opazil, da pri $N > 10$ pričnem dobivati vedno bolj različne rezultate, včasih se je algoritem zataknil na lokalnem minimumu red velikosti nad drugimi, kar je pričakovati, saj nam fazni prostor raste eksponentno kot

[1] Vedno sem mislil, da se je gospod pisal *Thompson*...

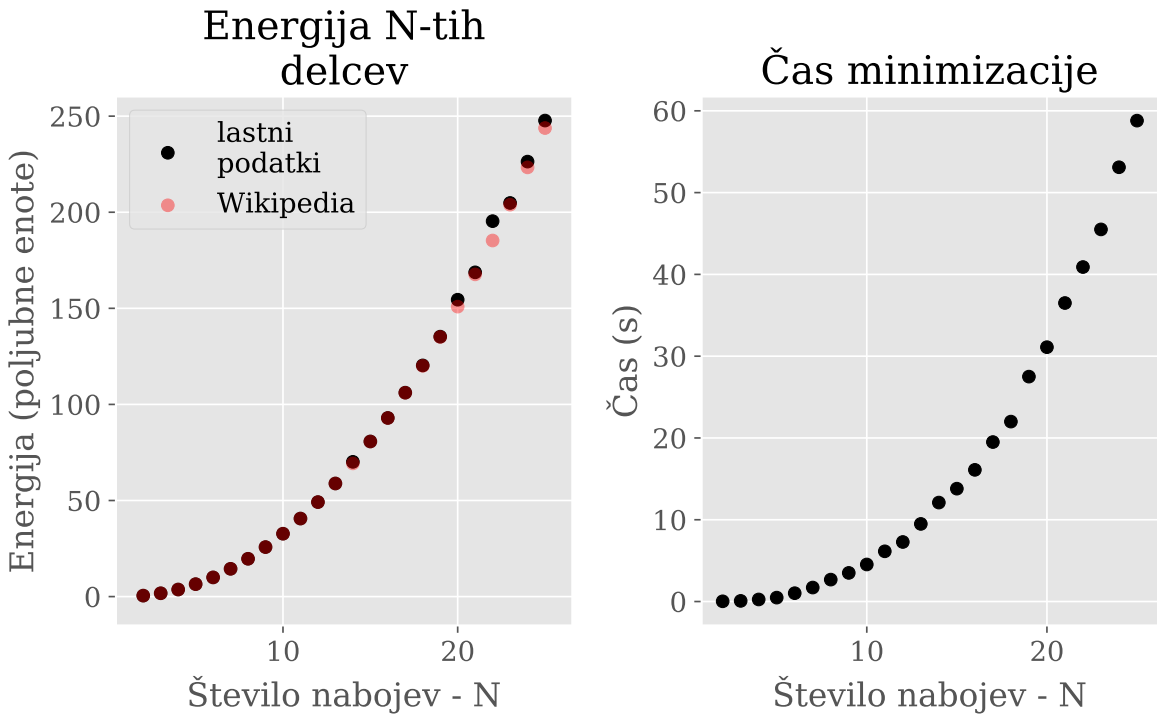
[2] Pri čemer seveda privzamemo, da imamo dovolj dober generator psevdonaključnih števil in dovolj dobro strojno natančnost...



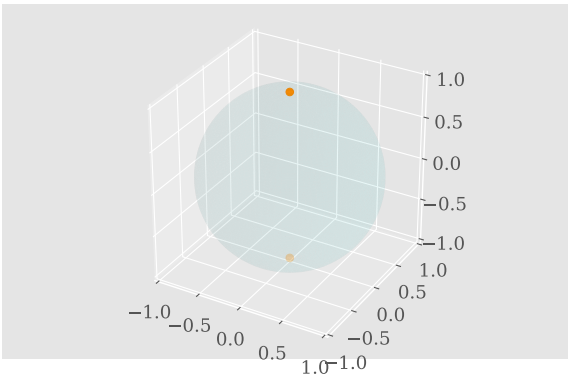
Slika 1: Prikaz poteka naivnega ugibanja minimalne energije v 1000 poskusih. Obe osi sta logaritmični, kar upravičim z dvema argumentoma; na vodoravni osi se dogajanje hitro ustali, sprva hitro najdemo boljše kandidate, nato pa je le redkokateri še boljši od predhodnikov, zato se vrednost energije izravna. Na navpični osi pa imamo širok razpon energije, zato bi linearna skala povsem zakrila dogajanje z energijo sistemov, ki jim dodamo samo en ali dva naboja. Kakšna izmed krivulj se konča že po nekaj deset iteracijah, ker od tistega koraka dalje nismo uganili nobene konfiguracije, ki bi zmanjšala energijo.

2^N in problem minimizacije postaja vedno bolj kompleksen. Večinoma pa je vrednost opletala okrog 1% nad minimalno.

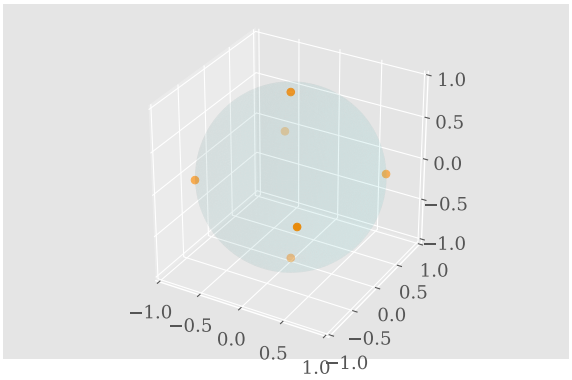
Nekaj konfiguracij prilagam (slike 3, 4, 5, 6).



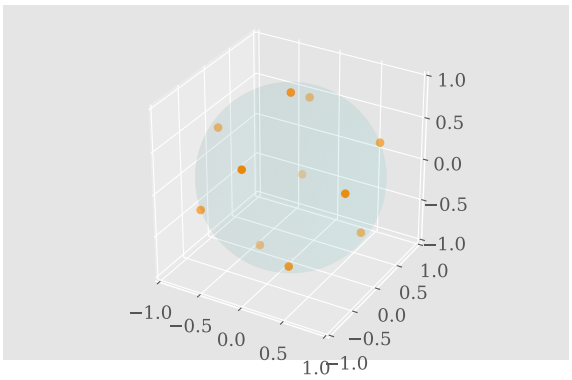
Slika 2: Energija, ki jo dosežem z minimizacijo energije pri Thomsonovem problemu ter čas, ki je potreben za minimizacijo. Pri $N < 15$ sem minimizacijo ponavljal po sedemkrat, kar mi je dalo več podatkov o času minimizacije in nižjo vrednost energije, kasneje, ko je čas pričel naraščati, pa sem minimizacijo za vsak N ponovil le dvakrat. Poleg svojih minimalnih energij prilagam še podatke iz Wikipedie (`https://en.wikipedia.org/wiki/Thomson_problem`).



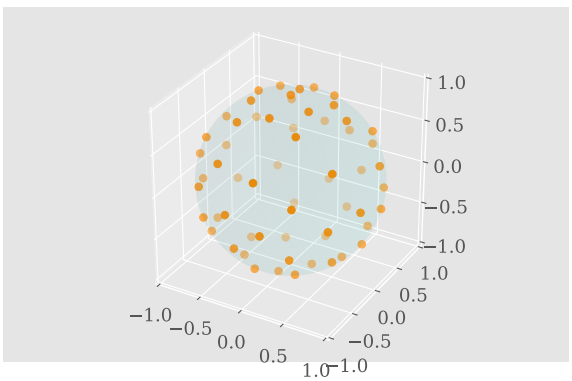
Slika 3: Konfiguracija 2 nabojev, ki minimizira energijo.



Slika 4: Konfiguracija 6 nabojev, ki minimizira energijo.



Slika 5: Konfiguracija 11 nabojev, ki minimizira energijo.



Slika 6: Konfiguracija 60 nabojev, ki minimizira energijo.

2 Optimalna vožnja skozi semafor

2.1 Naloga

Problem optimalne vožnje skozi semafor, ki smo ga spoznali pri nalogi 1, lahko rešujemo tudi z numerično minimizacijo, če časovno skalo diskretiziramo.

Lagrangianu $\int(\mathrm{d}v/\mathrm{d}t)^2\mathrm{d}t - \lambda\int v\mathrm{d}t$ lahko dodamo omejitev hitrosti v obliki členov $\exp\beta(u-u_{lim})$, če hočemo (približno) zagotoviti u_{lim}. Izpolnitev pogoja je toliko ostrejša, kolikor večji β vzamemo. Poskusiš lahko tudi druge omejitvene funkcije, na primer kakšno funkcijo s polom.

Za iskanje Lagrangeovega multiplikatorja lahko uporabiš bisekcijo ali kakšno drugo vgrajeno metodo za iskanje ničel na funkciji $l(\lambda) = \int v(\lambda, t)\mathrm{d}t$, kjer je $v(\lambda t)$ rezultat minimizacije pri izbranem λ. V tem primeru je enakost izpolnjena eksaktno.

2.2 Postopek

Enačbo bom tudi tokrat reševal v brezdimenzijski obliki s spremenljivkami:

$$x = \frac{t}{t_0}, \tag{1}$$

kjer je t_0 čas do prižiga zelene luči, in brezdimenzijsko hitrost:

$$y = v \cdot \frac{t_0}{L} = \frac{v}{\langle v\rangle}. \tag{2}$$

Količina, ki jo bomo minimizirali, bo

$$\int_0^1 (\frac{\partial y}{\partial x})^2\mathrm{d}x, \tag{3}$$

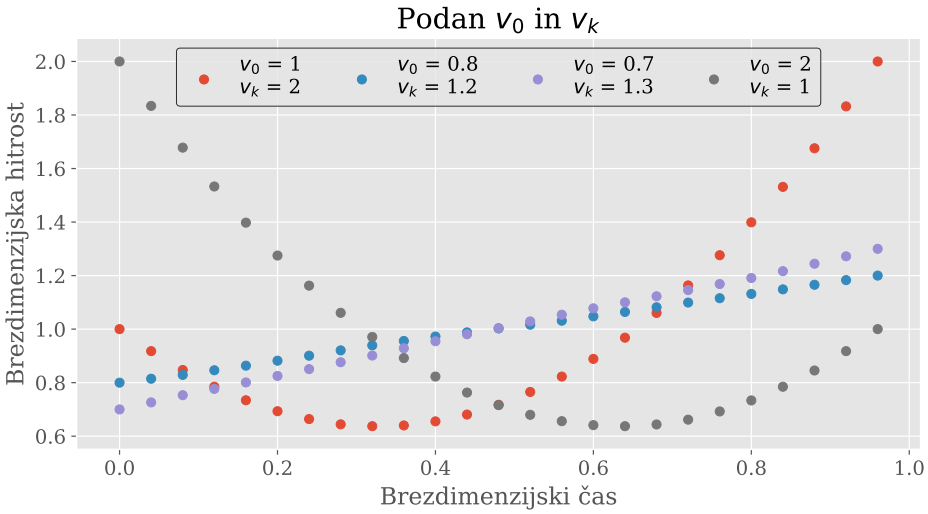
aproksimiral pa jo bom z vsoto

$$\frac{1}{N}\left[0,5(y_0-y_{-1})^2 + (y_1-y_0)^2 + \cdots + 0,5(y_N-y_{N-1})^2\right] \tag{4}$$

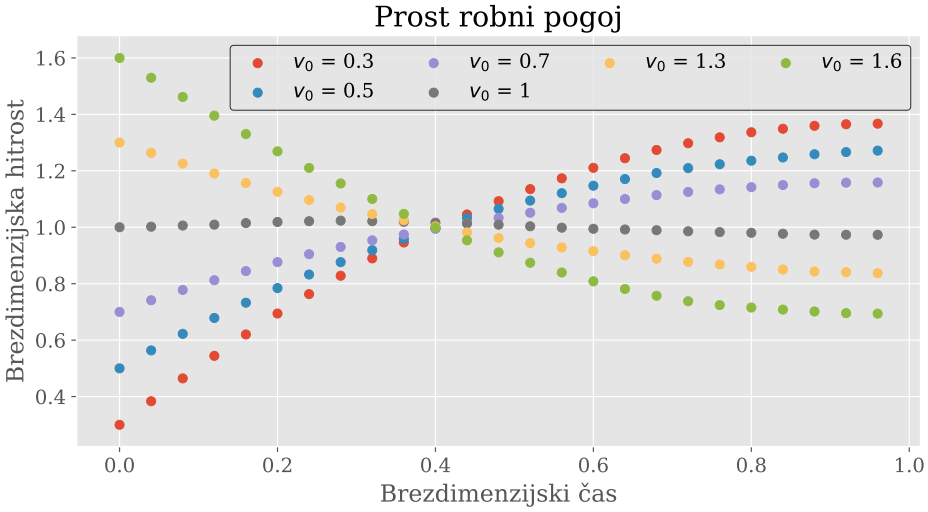
Časovno skalo sem diskretiziral in minimiziral hitrosti v 25 točkah. Uporabil sem algoritem SLSQP:

> Method SLSQP uses Sequential Least SQuares Programming to minimize a function of several variables with any combination of bounds, equality and inequality constraints. The method wraps the SLSQP Optimization subroutine originally implemented by Dieter Kraft. (`https://docs.ascipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`)

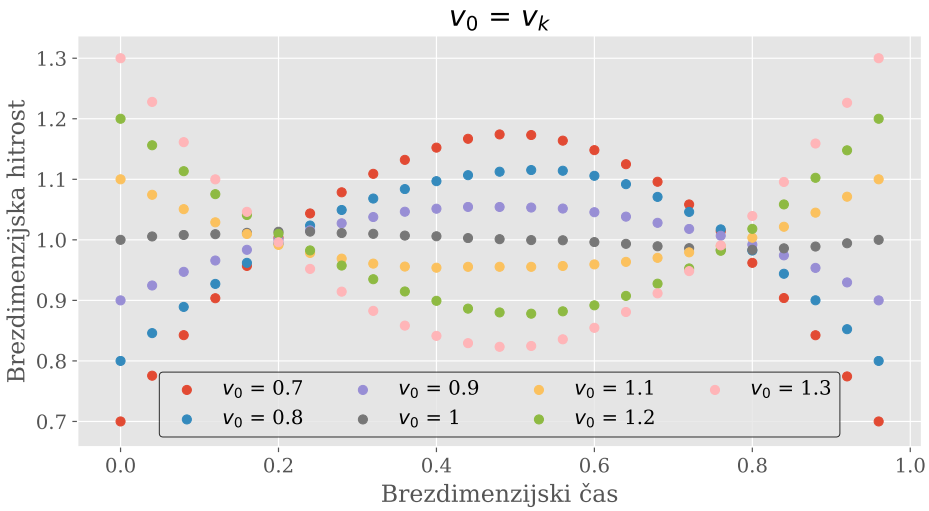
Ta algoritem omogoča udobno zastavitev vezi in robnih pogojev, v primer ponujam kodo 1. S takim formalizmom sem napadel problem semaforja in pogledal rešitve istih podnalog, ki smo jih obravnavali v prvi nalogi. Dodatno sem obravnaval primer, ko omejimo hitrost z zgornjo mejo.



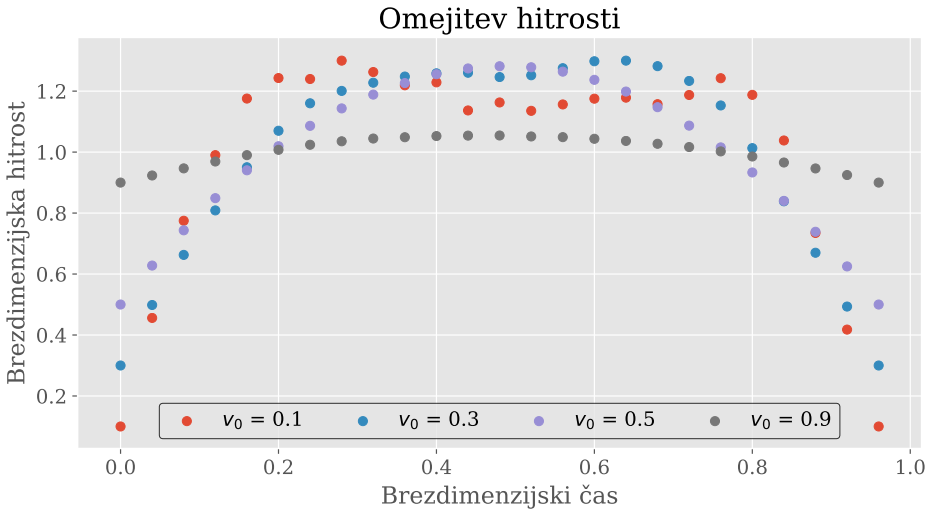
Slika 7: Predpisal sem začeten in končen pogoj, hitrost ob začetku opazovanja in ko dosežemo semafor. Seveda sem predpisal tudi skupno dolžino, ki naj pri času, ko dosežemo semafor, doseže točno L. Na sliki prepoznamo parabole, ki smo jih izpeljali v prvi nalogi.



Slika 8: Predpisal sem samo en robni pogoj, začetno hitrost. Pri prvi nalogi smo odsotnost drugega robnega pogoja premostili z matematiko, tu nam algoritem sam ponudi rešitve, ki se pri $x = 1$ izravnajo v vodoravno črto.



Slika 9: Kot pri prvi nalogi sem v tem podproblemu zahteval enakost končne in začetne hitrosti in dobil enake rezultate in pričakovano simetrijo okrog sredine časovnega intervala. Zaradi primerjave s prvo nalogo nisem obravnaval rešitev za zvezen prehod pri zaporednih M semaforjih.



Slika 10: Pri omejevanju maksimalne hitrosti sem poskusil z dodajanjem pogojev v python-constraints slovar minimizacijskega algoritma, a z majhnim uspehom. Več uspeha sem imel s sledenjem navodilom: pripravil sem dodatno funkcijo, ki je s faktorjem 10 000 penalizirala seštevek vseh hitrosti, ki so presegale 1.3. Do neke mere mi je uspelo, a pri nizkih začetnih hitrostih namesto pričakovane sploščitve na vrhu parabole dobim prenihavanja. Pri minimizaciji sem predpisal začetno in njej enako končno hitrost.

```python
def zminaj1(v0):
    x0 = np.random.uniform(size=N)

    cons = ({"type": "eq", "fun" : lambda arej: 1-np.sum(arej)/arej.shape[0]},
            {"type": "eq", "fun": lambda arej: arej[0]-v0},
            {"type": "eq", "fun": lambda arej: arej[0]-arej[-1]}
            )

    res = minimize(F, x0 = x0, method="SLSQP", constraints=cons)
    return res.x
```

Koda 1: Metoda SLSQP omogoča, da algoritmu, ki minimizira parametre stroškovne funkcije, ki jo minimiziramo, dodamo robne pogoje in še več. Konkretno v tem primeru zahtevam, da je integral hitrosti enak točno L oziroma 1 v brezdimenzijskih količinah, poleg tega predpišem začetno hitrost v_0 in enakost končne in začetne hitrosti, $v_0 = v_k$.