Nelinearna minimizacija

Miha Čančula

4. november 2011

1 Orodja

Nalogo sem reševal s prostim programom `GNU Octave`, ki za nelinearno optimizacijo ponuja funkcijo `sqp`.

2 Thompsonov problem

2.1 Izpeljava

Položaj diskretnih točkastih nabojev na krogli lahko opišemo z dvema koordinatama, $\vartheta \in [0, \pi]$ in $\varphi \in [0, 2\pi]$. Potencialna energija takih nabojev je odvisna le od medsebojne razdalje, zato moramo najprej izraziti razdaljo med dvema nabojema z njunima koordinatama. Kot med dvema točkama izračunamo kot skalarni produkt med njunima krajevnima vektorjema in znasa

$$\cos \alpha_{ij} = \cos(\varphi_i - \varphi_j) \sin \vartheta_i \sin \vartheta_j + \cos \vartheta_i \cos \vartheta_j \tag{1}$$

dejansko razdaljo med točkama pa po kosinusnem izreku

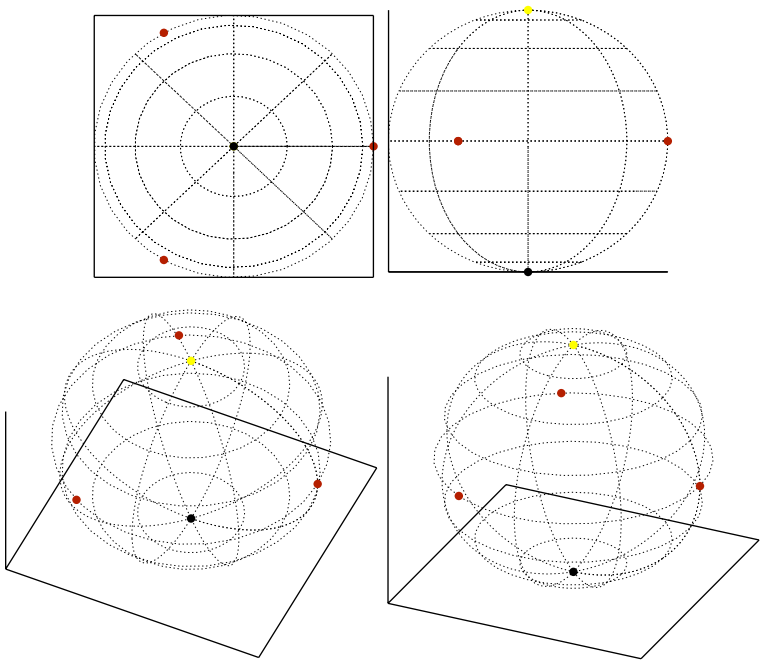
$$d_{ij} = \sqrt{2R^2(1 - \cos \alpha_{ij})} \tag{2}$$

Skupna potencialna energija sistema je vsota potencialnih energij vseh parov nabitih delcev na krogli

$$E = E_0 \sum_{i<j} \frac{\sqrt{2R^2}}{d_{ij}} = E_0 \sum_{i<j} \left[1 - \cos(\varphi_i - \varphi_j) \sin \vartheta_i \sin \vartheta_j - \cos \vartheta_i \cos \vartheta_j\right]^{-1/2} \tag{3}$$

Ker multiplikativna konstanta v energiji ne spremeni optimalne razporeditve nabojev po krogli, lahko izraz pretvorim v brezdimenzijske enote. Energija $E_0 = \frac{e^2}{4\pi\varepsilon_0 R}$ nas bo zanimala le, če bomo želeli izračunati vrednosti energije v minimumu, ne pa samega položaja minimuma.

$$y = \sum_{i<j} \left[1 - \cos(\varphi_i - \varphi_j) \sin \vartheta_i \sin \vartheta_j - \cos \vartheta_i \cos \vartheta_j\right]^{-1/2} \tag{4}$$



Slika 1: Optimalna razporeditev $N = 5$ nabojev po krogli

2.2 Reševanje

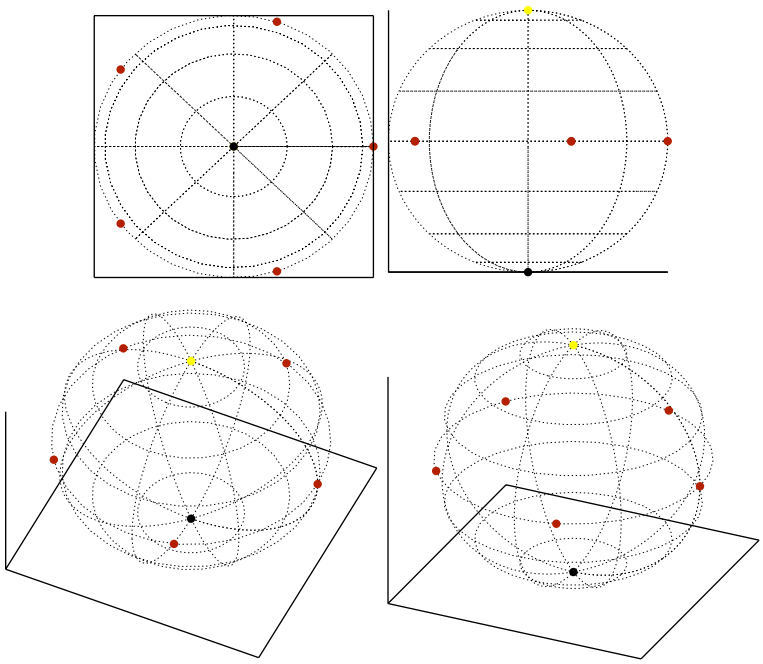
Brez izgube splošnosti lahko fiksiramo tri koordinate: $\vartheta_1 = 0$, $\varphi_1 = 0$, $\varphi_2 = 0$. Na ta načim preprečimo vrtenje okrog krogle med reševanje, hkrati pa pospešimo reševanje, saj zmanjšamo število prostostnih stopenj. Za problem z N naboji nam ostane $2N - 3$ prostih koordinat.

Pri prvem poskusu reševanja sem pogosto naletel na lokalne minimume, ko se je program ustavil v ne-optimalnem položaju. Zato sem uporabil preprost trik, da sem celoten račun ponovil večkrat, vsakič z naključno začetno porazdelitvijo, in vsakič shranil le tisti rezultat, ki je dal najmanjšo vrednosti potencialne energije. Ker je rezultat zanimiv predvsem za majhno število delcev, je bil izračun hiter in sem si lahko privoščil veliko število ponovitev, v mojem primeru sem jih uporabil 20.

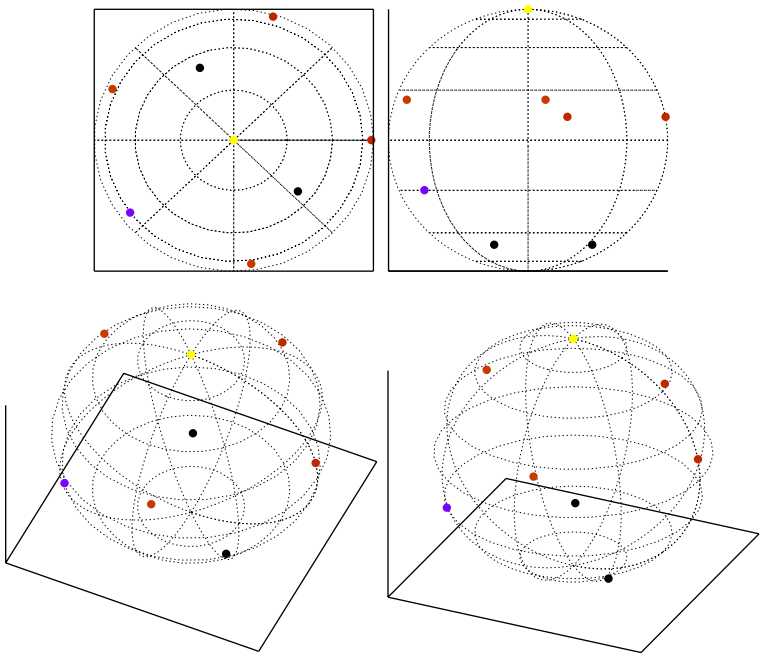
2.3 Rezultati

Porazdelitve sem izračunal za vse N od 2 do 12, posebej pa še za $N = 20$, vendar sem v poročilo vključil le tiste, za katere ne obstajajo platonska telesa s takim številom oglišč in jih ne znamo določiti na pamet. Predvsem zanimiv se mi zdi rezultat pri $N = 8$, saj optimalna porazdelitev ni niti malo podobna ogliščem kocke. Za lažjo predstave sem točke obarval glede na njihovo koordinato z, tako da so točke na zgornji polobli svetlejše, na spodnji pa temnejše.

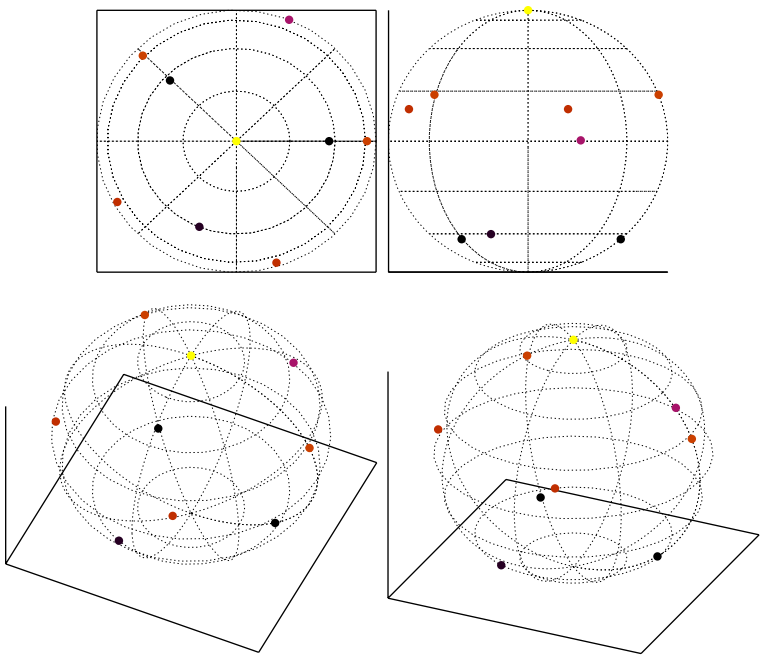
Pri $N \in \{3, 4, 6, 12, 20\}$ se naboji razporedijo v oglišča ustreznega platonskega telesa. Račun je možen tudi z večjim številom nabojev, vendar jih je težko prikazati v dveh dimenzijah.



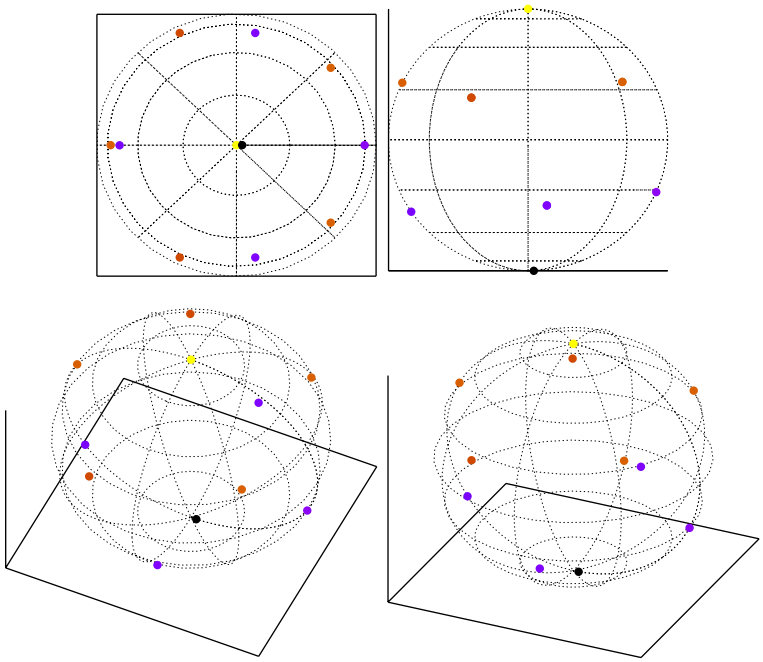
Slika 2: Optimalna razporeditev $N = 7$ nabojev po krogli



Slika 3: Optimalna razporeditev $N = 8$ nabojev po krogli



Slika 4: Optimalna razporeditev $N = 9$ nabojev po krogli



Slika 5: Optimalna razporeditev $N = 11$ nabojev po krogli

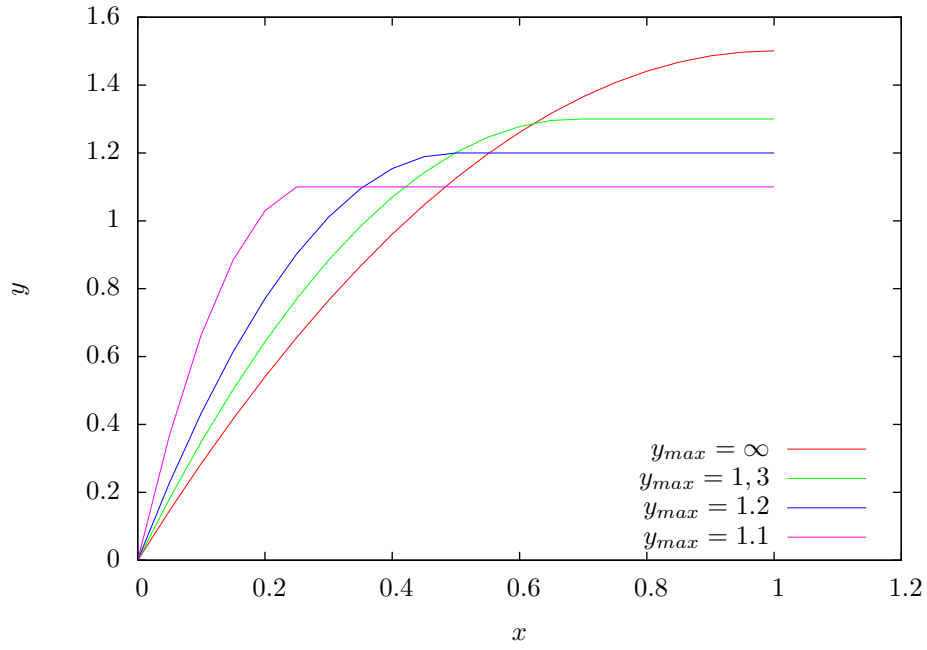
3 Voznja do semaforja

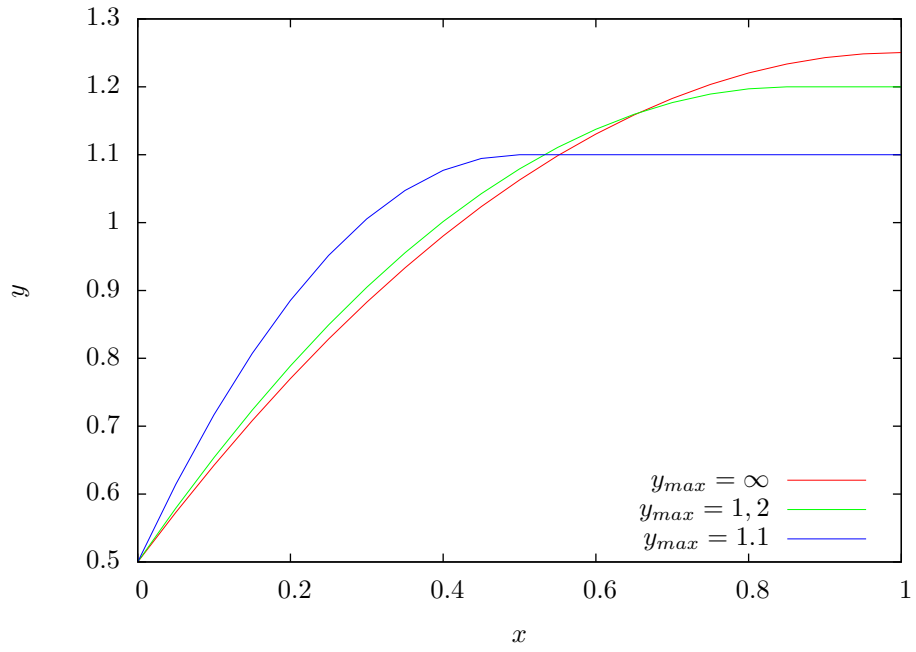
Na enak način sem reševal tudi problem iz prejšnje naloge. Časovni interval sem razdelil na N podintervalov, hitrost vožnje sem omejil med 0 in y_{max} in dodal omejitev, da je skupna pot enaka 1.

Izbral sem metodo, ki je sama določila smer najugodnejšega premikanja, brez da bi ji moral podati gradient akcije. Zato ni bilo potrebe po analitičnosti funkcije in sem lahko uporabil ostro omejitev tako da hitrost kot za pospešek. Izračun je bil dovolj hiter tudi za večje število intervalov (nekaj 100), zato ni bilo potrebe po optimizaciji, saj se večja natančnost računanja ne pozna na grafu.

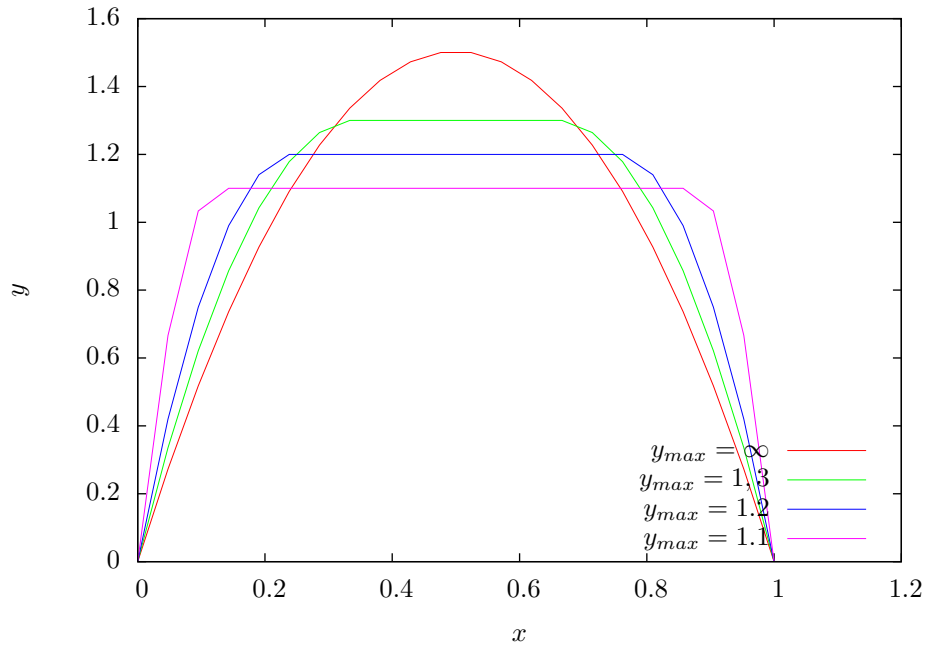
V primerjavi s prejšnjo nalogo dodatne omejitve prinesejo kar nekaj novih parametrov, zato sem narisal več grafov.

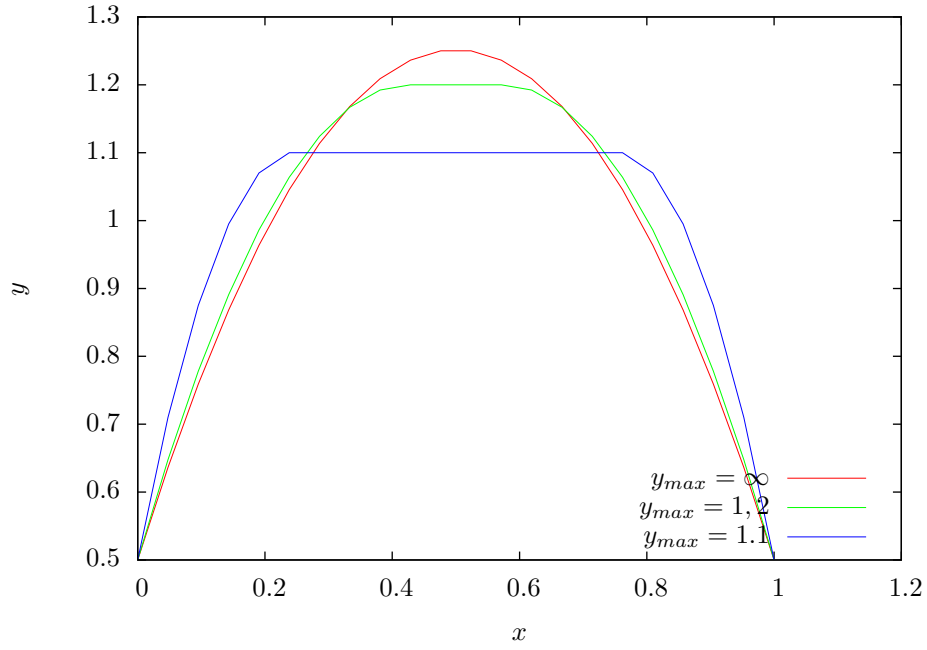
3.1 Brez končnega robnega pogoja

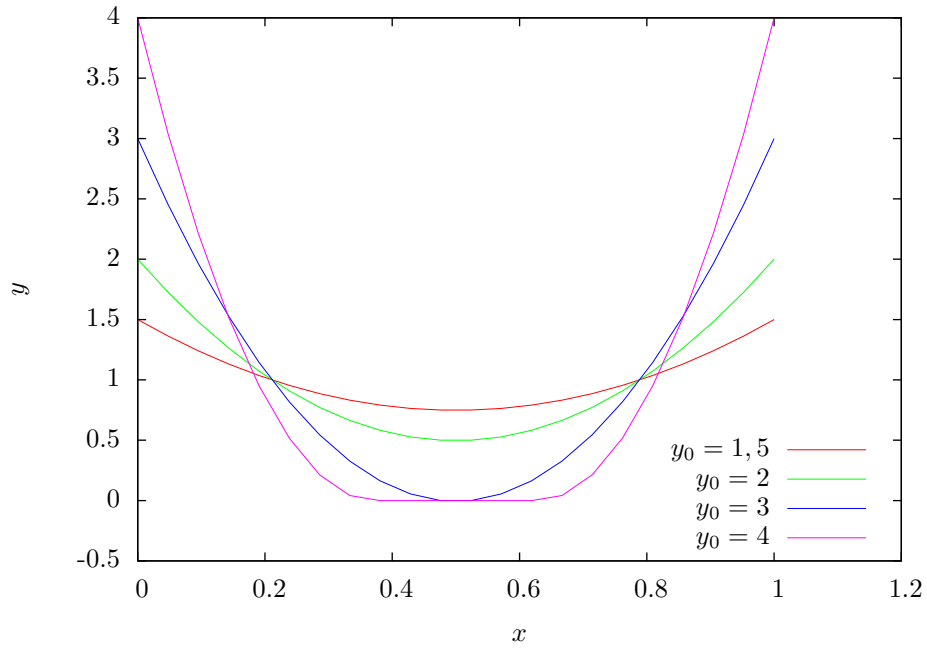




3.2 Periodična vožnja







3.3 Omejitev pospeška

Upošteval sem tudi asimetrično omejitev pospeška $a_{min} < a < a_{max}$, kjer je $a_{min} < 0$ največji pojemek pri zaviranju in $a_{max} > 0$ največji pospešek pri pospeševanju. Običajno so avtomobili sposobni hitrejšega zaviranja kot pospeševanja. S to dodatno omejitvijo moramo biti bolj previdni pri izbiri robnih

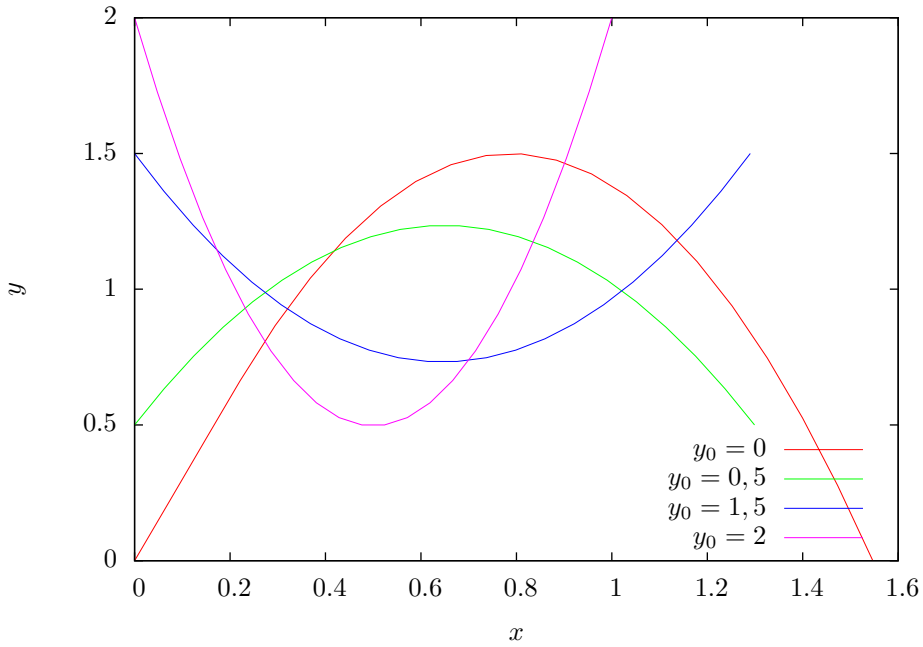
pogojev, saj moramo paziti, da rešitev se vedno obstaja.

Poleg tega rešitev postane manj zanimiva, saj avto večino časa pospešuje ali pa zavira s polno močjo. Zato sem raje upošteval neenakost v času, tako da sem dopustil možnost, da avto v križišče zapelje po tem ko se prižge zelena luč. Pri pogojih, ko je začetna hitrost prevelika, da bi avto uspel upočasniti pred prihodom v križišče, pa še vedno ne moremo najti rešitve.

Če bi dopustil poljuben čas do križišča, bi bila najugodnejša pot seveda s konstantno hitrostjo, saj bi bila v tem primeru akcija enaka 0. Zato sem moral upoštevali čas (oz prepotovano pot v fiksnem času) tudi v funkciji, ki jo želimo minimizirati. Ta funkcija je tako postala

$$S = \sum_{i=0}^{N} y_i - e^{\beta(1-s)} \tag{5}$$

Prehiter prihod do križišča sem omejil z ostro mejo, medtem ko je prepozen prihod kaznovan le z eksponentno funkcijo.



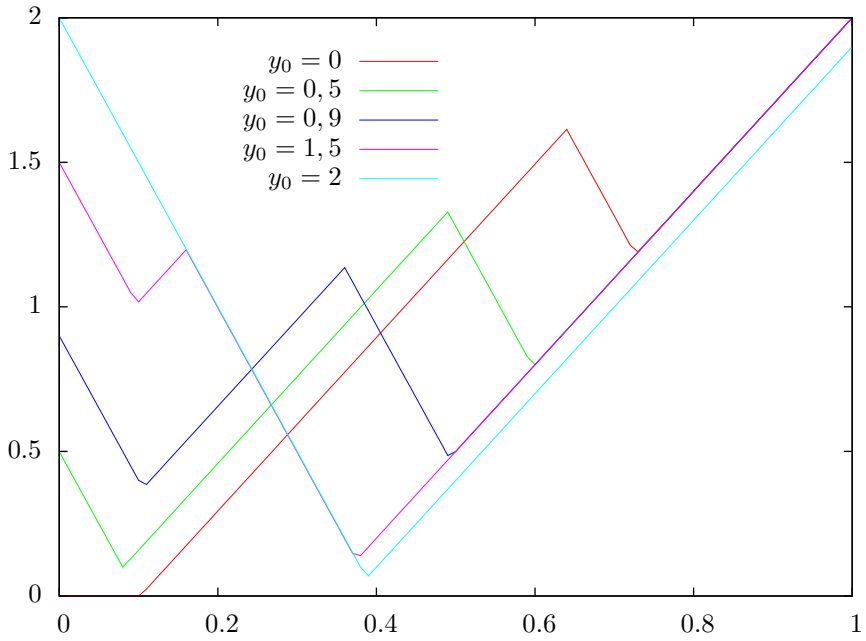
Asimetrična omejitev pospeška pride do izraza pri majhnih začetnih in končnih hitrostih, ko je treba najprej pospeševati in nato zavirati.

4 Linearno programiranje

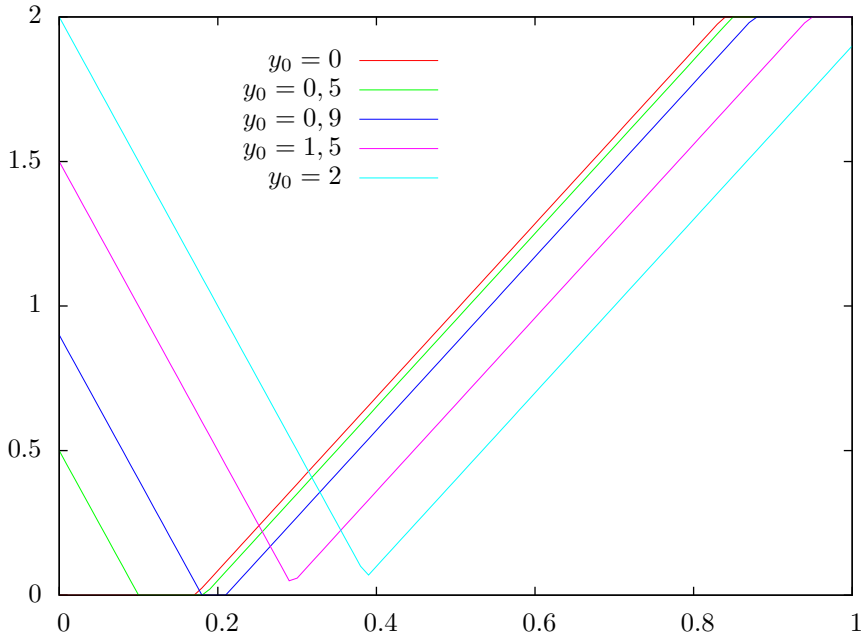
Problem vožnje do semaforja sem obravnaval tudi z uporabo linearnega programiranje. V ta namen ima Octave vključeno funkcijo `glpk`, ki najde optimalno rešitev matričnega sistema enakosti in neenakosti. Lagranžijan iz prejšnje naloge tu ne pride v poštev, saj imamo na voljo le linearno operacije, zato kvadrata popeška ne moremo upoštevati. Namesto tega sem iskal rešitev, pri kateri

semafor prevozimo s čim večjo hitrostjo, upošteval pa sem naslednje linearne omejitve:

- Prepotovana pot mora biti v brezdimenzijskih enotah enaka 1
- Pospešek med dvema točkama ne sme preseči določene vrednosti a_{max}
- Pospešek ne sme biti manjši od a_{min}, $a_{min} < 0$
- Hitrost v vsaki točki je med 0 in y_{max}



Zaradi enostavnosti problema in omejitev je tudi rešitev enostavna: Najprej močno zaviranje da pridobimo čas, nato pa pospeševanje do največje hitrosti. V primeru, da bi lahko največjo dovoljeno hitrost dosegli ze pred semaforjem, obstaja celo več načinov vožnje ki dajo isti rezultat. Ker nimamo nobene uteži, ki bi kaznovala pospeševanje in zaviranje, dobimo rešitve, kjer se pospeševanje in zaviranja izmenjujeta. Takšna vožnja ni preveč udobna, zato sem v funkcijo cene dodal člene, ki želijo čimvečjo hitrost ob vseh časih, ne samo na koncu. Da sem ohranil našo začetno željo, da je hitrost ob prehodu skozi križišče čim večja, so časi proti koncu obteženi z večjim koeficientom.



Rešitev se v omenjenih primerih spremeni, ampak spet ne opazimo ničesar nepričakovanega: Avto pospešuje z največjim možnim pospeškom do omejitve, nato pa vozi enakomerno do semaforja. V primeru, da omejitve hitrosti ne dosežemo, je rezultat enak kot na prejšnjem grafu.