SEMINARSKI RAD
IZ PREDMETA
NUMERIČKI ALGORITMI

Newton-ov metod za minimizaciju funkcija više nezavisnih promjenljivih, uz razne varijante izbora koraka duž odabranih pravca

---

Autori:
Anel MANDAL
Lejla MEHMEDAGIĆ

Profesor:
prof. dr. Željko JURIĆ, dipl.ing.el.

Asistent:
Šeila BEČIROVIĆ, BA ing.

ELEKTROTEHNIČKI FAKULTET
SARAJEVO

7. februar 2019.

Sažetak

U ovom radu prezentirati ćemo jednu od više metoda za višedimenzionalnu minimizaciju, tačnije Newton-ov metod koji pripada skupini metoda pretraživanja po pravcu. Kod ovih metoda se, polazeći od neke tačke bira neki pravac nakon čega se traži minimum duž tog pravca. Osvrnut ćemo se na klasični Newton-ov metod a finalna implementacija prezentirati će također kvazi-Newton metod koji predstavlja značajno poboljšanje klasičnog, s obzirom da isti nije primjenjiv u praksi. Cilj je dakle ostvariti nešto bržu a i precizniju aproksimaciju u odnosu na klasični metod. Poglavlje "Uvod" sadrži opis algoritma te probleme koje je potrebno riješiti prilikom konstrukcije istog. "Implementacija algoritma" sadrži sam pseudokod, koji će u ovom radu biti direktno implementiran u paketu numeričkog računanja Scilab. Rad završavamo sa nekim testiranjem u cilju utvrđivanja efikasnosti i preciznosti samog algoritma te se kratko osvrćemo i na samo primjene istog.

Abstract

In this paper, we will present one of several methods for multi-dimensional minimization, specifically Newton's method, which belongs to a group of line search methods. In these methods, we start from a specific point, choose a direction, which we move along to find a minimum. We will discuss the classic Newton method, and the final implementation will also present a quasi-Newton method that represents a significant improvement of the classic one, since it is not applicable in practice. Therefore, the goal is to achieve a faster and more precise approximation. Chapter "Uvod" contains a description of the algorithm and problems that need to be solved when constructing it. "Implementacija algoritma" contains the pseudocode itself, which in this paper will be directly implemented in the Scilab numerical computational package. We end the paper with some tests in order to determine the efficiency and precision of the algorithm itself, we also briefly look at its applications.

Sadržaj

1. Uvod

1.1. Minimizacija metodom pretraživanja po pravcu

Optimizacioni algoritmi su iterativni. Dakle, započinje se sa nekom početnom pretpostavkom varijable x te se generiše sekvenca poboljšanjih pretpostavki, dok se ista ne terminira, nadajmo se u rješenju. Kod metoda pretraživanja po pravcu se, polazeći od neke tačke $\mathbf{x}^*$ bira neki pravac, odnosno vektor $\mathbf{p}$, nakon čega se traži minimum duž tog pravca.

Tačke duž tog pravca imaju oblik $x^* + h\mathbf{p}$ s obzirom da taj pravac prolazi kroz tačku x^* a određen je vektorom $\mathbf{p}$ gdje h predstavlja predstavlja dužinu koraka duž odabranog pravca ($h \in \mathbb{R}$).

Ciljna funkcija f je, duž odabranog pravca, funkcija samo jedne realne promjenljive h, koju ćemo označiti sa $\varphi(h) = f(\mathbf{x}^* + h\mathbf{p})$. Iz ovoga neposredno slijedi da se za minimizaciju funkcije $\varphi(h)$ mogu koristiti neke od tehnika za nalaženje minimuma funkcije jedne realne promjenljive, kao što je recimo Brentov[1] algoritam.

Nakon pronalaska koraka h^* koji minimizira funkciju φ, tačka $\mathbf{x}^* + h\mathbf{p}$ se uzima za novu tačku, nakon čega se bira novi pravac te se postupak ponavlja dok se ne utvrdi da se nalazimo u minimumu, ili barem njegovoj okolini, što sve zavisi od same implementacije algoritma.

Prirodno se postavlja pitanje određivanja koraka h. Strategija kod koje se korak h bira tako da se minimizira funkcija $\varphi(h)$, kao što je to navedeno iznad, naziva se *egzaktno pretraživanje po pravcima*.

Međutim, ova strategija se ne primjenjuje u praksi s obzirom da nalaženje minimuma funkcije $\varphi(h)$ samo za sebe može biti dosta zahtjevno. Stoga se javlja potreba za drugačijom strategijom, kao što je recimo *neegzaktno pretraživanje po pravcima* kod koje se zadovoljavamo sa tačkom u kojoj je funkcija *dovoljno opala* u odnosu na trenutnu tačku.

Ipak, potrebno je i dalje postaviti određene uvjete na korak jer se može desiti da se konvergencija ne postigne prilikom lošeg izbora. S tim na umu, postoji nekoliko uvjeta, koji mogu garantirati validan izbor koraka h a koje ćemo diskutovati dalje u nastavku.

Međutim, izbor koraka nije najveći problem ovih metoda već izbor pravca. Većina metoda bira silazne pravce, odnosno pravce duž kojih ciljna funkcija opada od polazne tačke. S obzirom da se ovaj rad bavi Newtonovim[2] metodama za minimizaciju nećemo se upuštati u mnogobrojne načina izbora pravca. Neki od metoda, kao što je upravo i Newtonov, u svakoj iteraciji predlažu ne samo pravac nego čak i korak. Treba naglasiti da nemamo garanciju da je taj pravac odnosno korak ujedno i validan. O ovome ćemo također diskutovati u nastavku.

[1] Richard Peirce Brent (1946-), australijski matematičar i informatičar

[2] Isaac Newton (1642 – 1717), engleski fizičar, matematičar i astronom

1.2. Newton-ov metod

Razmotrimo problem minimizacije bez ograničenja

$$min\ f(x)$$
$$x \in \mathbb{R}^n,\ n = 1$$

Neka je $f : R \to R$ dva puta diferencijabilna funkcija. Tada se ista može razviti u Taylorov[3] polinom drugog stepena u okolini trenutne aproksimacije x^* kao

$$f(x) \approx f(x^* + (x - x^*)) \approx f(x^*) + f'(x^*)(x - x^*) + \frac{1}{2}f''(x^*)(x - x^*)^2,\ \ k \in \mathbb{N}$$

Nova iteracija može se izračnati kao tačka u kojoj izvod aproksimativne funkcije ima vrijednost 0, pri čemu se dobija

$$0 = f'(x^*) + f''(x^*)(x - x^*)$$

odnosno, imamo Newton-ov iterativni proces

$$x_i = x^*_{x-1} - \frac{f'(x^*_{x-1})}{f''(x^*_{x-1})},\ \ i \in \mathbb{N}$$

Lahko se primijeti da ovo nije ništa drugo nego Newtonov metod za rješavanje nelinarne jednačine $f'(x) = 0$. S tim na umu možemo zaključiti da ovaj metod ima kvadratnu konvergenciju.
S obzirom da nas zanimaju funkcije više nezavisnih promjenljivih nećemo se upuštati u algoritam za ovaj specijalan slučaj, odnosno kada je $n = 1$.

Razmotrimo, dakle, generalniji slučaj, odnosno

$$min\ f(x)$$
$$x \in \mathbb{R}^n,\ \ n \geq 2$$

Oko tačke x^* (početna aproksimacija minimuma), funkcija f se može aproksimirati sa Taylorovim polinomom drugog reda na sljedeći način:

$$f(x) \approx f(\mathbf{x}^* + (\mathbf{x} - \mathbf{x}^*)) \approx f(\mathbf{x}^* + h\mathbf{p}) \approx f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T h\mathbf{p} + \frac{1}{2}h\mathbf{p}^T \nabla^2 f(\mathbf{x}^*)h\mathbf{p}$$

Ako uzmemo $h\mathbf{p} = \mathbf{t} = \mathbf{x} - \mathbf{x}^*$

$$f(\mathbf{x} + \mathbf{t}) = q(\mathbf{t}) = f(\mathbf{x}^* + (\mathbf{x} - \mathbf{x}^*)) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T \cdot \mathbf{t} + \frac{1}{2}\mathbf{t}^T \nabla^2 f(\mathbf{x}^*)\mathbf{t}$$

[3]Brook Taylor (1685–1731) je bio engleski matematičar, najbolje poznat po Taylorovom teoremu i Taylorovom redu

Pri čemu vrijedi sljedeće:

- $h \in \mathbb{R}$, je korak duž odabranog pravca
- $\mathbf{p} \in \mathbb{R}^n$, je pravac (odnosno vektor) duž kojeg tražimo minimum
- $\nabla f(\mathbf{x}^*)$ - gradijent funkcije u tački $\mathbf{x}^*$
- $\nabla^2 f(\mathbf{x}^*)$ - Hessian funkcije u tački $\mathbf{x}^*$

Narednu aproksimaciju minimuma funkcije f ćemo aproksimirati sa minimumom model funkcije $q(\mathbf{t})$. Navesti ćemo prvo nekoliko teorema diferencijalnog računa koji će nam poslužiti za nalaženje izvoda ove model funkcije. Potrebno je dakle naći izvod po $\mathbf{t}$.
Napomenimo da, ukoliko su $\frac{\partial^2 f}{\partial x_1 \partial x_j}$ neprekidni u okoličini tačke $\mathbf{a}$,
za $\forall i, j = 1, ..., n$, vrijedi da je Hessian simetrična matrica.

Teorema 1: Neka je $\mathbf{a} \in R^n$ konstantan vektor i neka je funkcija $\mathbf{f} : R^n \to R$ definisana kao:

$$\mathbf{f}(\mathbf{x}) = \mathbf{a},\ \mathbf{x} \in R^n$$

Tada važi:

$$\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \mathbf{0}, \in R^n$$

Teorema 2: Neka je $\mathbf{a} \in R^n$ konstantan vektor i neka je funkcija $\mathbf{f} : R^n \to R$ definisana kao:

$$\mathbf{f}(\mathbf{x}) = \mathbf{a}^T\mathbf{x},\ \mathbf{x} \in R^n$$

Tada važi:

$$\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \mathbf{a}, \mathbf{x} \in R^n$$

U teoriji optimizacije vrlo bitnu ulogu imaju simetrične matrice, tj. primjenjivat ćemo sljedeću teoremu

Teorema 3: Neka je $\mathbf{A} \in R^n$ simetrična matrica i neka je funkcija $\mathbf{f} : R^n \to R$ definisana kao:

$$\mathbf{f}(\mathbf{x}) = \mathbf{x}^T A\mathbf{x},\ \mathbf{x} \in R^n$$

Tada važi:

$$\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = 2A\mathbf{x}, \mathbf{x} \in R^n$$

Imajući ove teoreme na umu lahko se nalazi izvod model funkcije koji glasi:

$$\nabla q(\mathbf{t}) = \nabla f(x^*) + \nabla^2 f(x^*) \cdot \mathbf{t}$$

Prepostavimo sada da je nova tačka $\mathbf{t} = \mathbf{x}^* + h\mathbf{p}$ upravo tražena tačka minimuma. U toj tački mora vrijediti $\nabla f(\mathbf{x}^* + h\mathbf{p}) = 0$, onda imamo

$$0 \approx \nabla f(x^*) + \nabla^2 f(x^*) \cdot h\mathbf{p}$$

Sada, uz pretpostavku da je Hessian simetrična matrica, imamo

$$h\mathbf{p} \approx -\nabla^2 f(\mathbf{x}^*)^{-1} \nabla f(\mathbf{x}^*)$$

odnosno, koristeći matrično lijevo dijeljenje,

$$h\mathbf{p} \approx -\nabla^2 f(\mathbf{x}^*) \backslash \nabla f(\mathbf{x}^*)$$

Ako prihvatimo konveciju po kojoj je gradijent vektor red a ne vektor kolona onda je ovo ekvivalentno

$$h\mathbf{p} \approx -\nabla^2 f(\mathbf{x}^*) \backslash \nabla f(\mathbf{x}^*)^T$$

Dakle dobili smo čitav pomak, odnosno i pravac i dužinu tog pravca $h\mathbf{p}$ koji nas iz tekuće tačke $\mathbf{x}^*$ vodi do minimuma.

Na osnovu svega ovog, formira se Newton-ov iterativni proces

$$\mathbf{x}_i = \mathbf{x}_{i-1} - \nabla^2 f(\mathbf{x}_{i-1}) \backslash \nabla f(\mathbf{x}_{i-1})^T$$

Iako je veoma privlačno u jednom potezu dobiti i pomak i pravac, Newtonov metod ipak posjeduje i nekoliko problema/nedostataka. Prvi je recimo što nemamo garanciju da će Hessian biti pozitivno definitan odnosno dobijeni pravac nije nužno silazni. Također, veliki nedostatak je što je u svakoj iteraciji potrebno izračunati taj Hessian kao i njen inverz.

U suštini, klasični Newton-Raphson[4] metod sam za sebe ne nalazi velike primjene u praksi. Moguće je isti modifikovati da se osigura globalna konvergencija npr. koristeći algoritme linijskog pretraživanja koje ćemo uskoro ukratko opisati. Međutim postoje i bolje metode, kao što su Quasi-Newton metode koje predstavljaju modifikacije klasičnog algoritma a kojim se ujedno i rješavanju većina nedostataka klasičnog metoda.

1.3. Quasi-Newton metod

Unatoč širokom spektru metoda za minimizaciju bez ograničenja, metodi se mogu široko kategorisati u smislu informacija izvoda koje koriste, odnosno ne koriste. Metode pretraživanja koje koriste samo vrijednosti funkcija (npr. simplex pretraga Neldera[5] i Meada[6]) najprikladnije su za probleme koji nisu glatki ili imaju niz diskontinuiteta. Gradijentne metode su općenito učinkovitije kada je funkcija koja se minimizira neprekidna u njenom prvom izvodu. Metode višeg reda, kao što je Newtonova metoda, prikladne su najčešće samo kada se izvod drugog reda lahko računa jer je računanje istog, koristeći numeričku diferencijaciju, računski skupa operacija. Metode gradijenta koriste informacije o nagibu funkcije kako bi diktirali smjer pretraživanja gdje se smatra da je minimum. Najjednostavniji od njih je metoda najstrmijeg pada u kojoj se pretraživanje vrši u smjeru negativne vrijednosti gradijenta funkcije, s obzirom da je poznato da gradijent funkcije pokazuje u smjeru najbržeg rasta funkcije.

Od metoda koje koriste informacije o gradijentu, najomiljeniji a ujedno i najčešće korišteni su kvazi-Newton metodi. Klasični Newton metod računa Hessian direktno i nastavlja silaznim pravcem locirati minimum. Računanje Hessiana numerički generalno uključuje veliki broj izračunavanja. Kod kvazi-Newton metoda pokušava se izbjeći računanje Hessian matrice. Drugim riječima, inverzna matrica Hessiana aproksimira se nekom matricom $\mathbf{H}_i$ koja se ažurira svaku iteraciju pri čemu aproksimacije postaju sve bolje prilikom svake iteracije.

Sa $\mathbf{H}_i$ označavamo aproksimaciju inverza Hessiana matrice $\nabla^2 f(\mathbf{x}^*)^{-1}$ u i-toj iteraciji.

[4]Joseph Raphson (1648-1715), ruski matematičar
[5]John Ashworth Nelder FRS (1924–2010), engleski statističar i
[6]Roger Mead (1938 - 2015), engleski statističar

Kod kvazi-Newton metoda zahtjeva se zadovoljavanje nekoliko uslova:

- $\forall i \in \mathbb{N}$ vrijedi da je matrica $\mathbf{H}_i$ pozitivno definitna.
- $\forall i \in \mathbb{N}$ pravac $\mathbf{p}_i = -\mathbf{H}_i \nabla f(\mathbf{x}_i)$ je silazni pravac, odnosno pravac opadanja ciljne funkcije.
- $\forall i \in \mathbb{N}$ ažuriranje ove matrice nije isuviše složeno (po mogućnosti ne više od vremena reda n^2)

Prirodno se nameće sljedeće pitanje. Kako obaviti ažuriranje/korekciju matrice koja je aproksimacija Hessiana (odnosno inverz Hessiana) a da pri tom budu ispunjeni navedeni uslovi?

Postoji nekoliko metoda, od kojih su najpoznatiji **DFP (Davidon[7]-Fletcher[8]-Powell[9]) algoritam** i **BFGS (Broyden[10]-Fletcher-Goldfarb[11]-Shanno[12]) algoritam**. U ovom radu koristiti će se **BFGS** metod koji se danas smatra najboljom od svih kvazi-Newton metoda. Teorijski, BFGS posjeduje sve dobre osobine DFP ažuriranja, s tim da je još ovaj metod u kombinaciji sa netačnim linijskim pretraživanjem (recimo Armijo[13] ili Wolfe[14]-Powell[15], a koji su pritom vrlo dobro implementirani) globalno konvergentan.

Uglavnom, matrice $\mathbf{H}_i$ se u ovom algoritmu ažuriraju po formuli

$$\mathbf{H}_i = \mathbf{H}_{i-1} + \frac{\boldsymbol{\delta}_i \boldsymbol{\delta}_i^T}{\boldsymbol{\gamma}_i^T \boldsymbol{\delta}_i} + \frac{(\boldsymbol{\gamma}_i^T \mathbf{H}_{i-1} \boldsymbol{\gamma}_i)(\boldsymbol{\delta}_i \boldsymbol{\delta}_i^T)}{(\boldsymbol{\gamma}_i^T \boldsymbol{\delta}_i)^2} - \frac{\mathbf{H}_{i-1} \boldsymbol{\gamma}_i \boldsymbol{\delta}_i^T + \boldsymbol{\delta}_i \boldsymbol{\gamma}_i^T \mathbf{H}_{i-1}}{\boldsymbol{\gamma}_i^T \boldsymbol{\delta}_i}$$

gdje je

$$\boldsymbol{\delta}_i = \mathbf{x}_i - \mathbf{x}_{i-1}, \quad \boldsymbol{\gamma}_i = \nabla f(\mathbf{x}_i)^T - \nabla f(\mathbf{x}_{i-1})^T$$

Dakle, umjesto formule

$$p_i = -\nabla^2 f(\mathbf{x}_i) \backslash \nabla f(\mathbf{x}_i)^T)$$

koristi se formula

$$p_i = -\mathbf{H}_i \nabla f(\mathbf{x}_i)^T)$$

Ovo naravno nije jedini način ažuriranja matrice kod ovog metoda ali se najčešće koristi.

[7] William Cooper Davidon (1927–2013), bio je američki profesor fizike i matematike
[8] Roger Fletcher FRS FRSE (1939–2016), bio je britanski matematičar i profesor
[9] Michael James David Powell FRS FAA (1936–2015), bio je britanski matematičar
[10] Charles George Broyden (1933-2011), bio je engleski matematičar i fizičar
[11] Donald Goldfarb (1941-), američki matematičar
[12] David F. Shanno (1938-), američki matematičar
[13] Larry Armijo (1938-), američki matematičar
[14] Philip Starr "Phil" Wolfe (1927-2016), bio je američki matematičar
[15] Michael James David Powell (1936-2015), bio je britanski matematičar

1.4. Metode linijske pretrage

Razvoj algoritma za probleme minimizacije se često temelji na metode linijskog pretraživanja. Linijska pretraga se koristi kao dio većeg algoritma za optimizaciju. Metode tačnog (egzaktnog) linijskog pretraživanja se ne primjenjuju u praksi iz već navedenih razloga, ali imaju veliki teorijski značaj. Svaka iteracija metoda linijske pretrage računa pravac p i odlučuje koliko daleko da se kreće duž tog pravca. Uspjeh linijske pretrage ovisi o efiktivnom izboru i pravca i dužine koraka. Većina metoda zahtijeva da pravac p bude silazni kao što smo već spomenuli u nekoliko navrata, s obzirom da ovaj uvjet garantira da se funkcija može *reducirati* duž ovog pravca.

Pri računanju dužine koraka h, suočavamo se sa kompromisom. Cilj je izabrati takav korak koji će dati znatnu redukciju funkcije ali nije poželjno potrošiti previše vremena na pronalazak istog. Kao što se to radi kod egzaktnog linijskog pretraživanja, idealno bi bilo izabrati korak koji će minimizirati funkciju φ ali je to kao što smo već rekli generalno skupa operacija. Tipični algoritam linijskog pretraživanja se obavlja u dvije faze:

- faza ograđivanja pronalazi interval koji sadrži poželjne korake
- faza bisekcije (metod polovljenja) ili interpolacije računa dobar korak unutar ovog intervala

Postoje razni uvjeti terminiranja za algoritam linijskog pretraživanja. Za metod koji će ovdje biti opisan (BFGS) najčešće se koriste Wolfe-Powell uvjeti ili njihova strožija varijanta. Ovaj uvjet, koji je zapravo kombinacija dva druga uvjeta propisuje da korak h bi prije svega trebao dati *dovoljan pad* ciljne funkcije, mjereno sljedećom nejednakošću

$$f(\mathbf{x}^* + h\mathbf{p}) \leq f(\mathbf{x}^*) + c_1 * h * \nabla f(\mathbf{x}^*) * \mathbf{p}$$

za neku konstantu $c_1 \in (0, 1)$ pri čemu se najčešće uzima $c_1 = 10^{-4}$. Drugim riječima, redukcija funkcije f trebala bi biti proporcionalna i sa h i $\nabla f(\mathbf{x}^*)\mathbf{p}$. Ova nejednakost je zapravo Armijov uvjet.

Međutim, ovaj uvjet sam za sebe nije dovoljan jer dopušta i premale vrijednosti koraka h, s obzirom da je isti zadovoljen za dovoljno male vrijednosti h. Da bi se isključili neprihvatljivo mali koraci, uvodi se dodatan uvjet zvani uvjet zakrivljenosti koji zahtijeva da h zadovoljava

$$\nabla f(\mathbf{x} + h\mathbf{p})\mathbf{p} \geq c_2 \nabla f(\mathbf{x}^*)\mathbf{p}$$

za neku konstantu $c_2 \in (c_1, 1)$ pri čemu se najčešće uzima $c_2 = 0.9$.

Ukoliko su u pitanju strogi Wolfe-Powell uvjeti, ovaj uvjet se mijenja za sljedeći

$$|\nabla f(\mathbf{x} + h\mathbf{p})\mathbf{p}| \leq c_2 |\nabla f(\mathbf{x}^*)\mathbf{p}|$$

Bitno je naglasiti da postoje dužine koraka h koje zadovoljavaju Wolfe-Powell uvjete za svaku funkciju koja je *glatka* i ograničena odozdo.

2. Implementacija algoritma

2.1. Klasični Newtonov (Newton-Raphson) metod

Ulazni parametri su:

- funkcija **f** čiji minimum tražimo
- početna aproksimacija **x0** $\in \mathbb{R}^n$
- željena tačnost **eps**
- maksimalni broj iteracija **maxiters**

Prilikom opisivanja ovog metoda, već smo naveli da je potrebno računanje Hessiana u trenutnoj tački x, pa ćemo kao zaustavni kriterij uzeti uvjet $|\nabla f(x)| < \epsilon$ (linija 9). Razlog je priroda predefinisane funkcije *numderivative* u Scilabu koja vraća Jakobijan i Hessian funkcije u datoj tački (linija 8), a Jakobijan u ovom slučaju zapravo predstavlja gradijent $\nabla f(x)$.

Ukoliko je minimum pronađen u toku maxiters iteracija, funkcija vraća tu vrijednost (linija 10). U suprotnom, ispisuje poruku 'Minimum cannot be found within given number of iterations or it does not exist!' (linija 14). Implementacija ovako opisanog algoritma u programskom paketu Scilab glasi:

Newton-Raphson

```
1   function x = Newton_Raphson(f, x0, eps, maxiters)
2       if ( eps<=0 )
3           error('Invalid parameters!');
4           return;
5       end
6       x = x0(:);
7       for i = 0 : maxiters
8           [J, H] = numderivative(f, x, [], [], 'hypermat');
9           if (norm(J) <eps)
10              return;
11          end
12          x = x - H\J.';
13      end
14      disp('Minimum cannot be found within given number of iterations
15          or it does not exist!';
16  endfunction
```

Linija 12 predstavlja iterativni proces kojim računamo novu tačku, sve dok eventualno ne dostignemo minimum. Vidimo da uzimamo transponovanu vrijednost gradijenta jer funkcija *numderivative* vraća gradijent kao vektor red. Također, linija 6 dozvoljava da tačka x bude unesena i kao vektor red, ali mi je posmatramo kao vektor kolonu.

2.2. Broyden-Fletcher-Goldfarb-Shanno (BFGS) algoritam

Ulazni parametri su isti kao i u prethodnom metodu. Ova funkcija također vraća tačku x u kojoj se nalazi minimum ukoliko je pronađen, odnosno ispisuje poruku 'Minimum cannot be found within given number of iterations or it does not exist!'. Zaustavni kriterij je također isti zbog relativno lahkog računanja gradijenta, ali umjesto lijevog matričnog dijeljenja Hessiana i gradijenta, u ovom slučaju korak računamo tako da zadovoljava stroge Wolfe-Powellove uvjete. Ovo smo izveli pozivajući funkciju *strong_wolfe_powell* čiju implementaciju ćemo prikazati u nastavku. Prikažimo implementaciju ovog algoritma u programskom paketu Scilab:

BFGS

```
1   function[x] = BFGS(f, x0, eps, maxiters)
2       if (eps <= 0)
3           error('NInvalid parameters!');
4           return;
5       end
6       n = length(x0);
7       H = eye(n, n);
8       x = x0(:);
9       dx = numderivative(f, x);
10      for i = i : maxiters
11          if norm(dx) <eps
12              return;
13          end
14          p = -H * dx.';
15          h = strong_wolfe_powell(f, x, f(x), dx, p);
16          x_new = x + h*p;
17          dx_old = dx;
18          dx = numderivative(f, x_new);
19          if (isnan(dx))
20              disp('NaN value detected');
21              return;
22          end
23          s_i = x_new - x;
24          y_i = dx.' - dx_old.';
25          rho_i = 1/(y_i.'*s_i);
26          if (i==1)
27              H = (y_i.'*s_i)/(y_i.'*y_i) * eye(n, n);
28          end
29          E = eye(n, n);
30          H = (E-rho_i*s_i*y_i')*H*(E-rho_i*y_i*s_i.') + rho_i*s_i*s_i.';
31          x = x_new;
32      end
33      disp('Minimum cannot be found within given number of iterations
34          or it does not exist!';
35  endfunction
```

Ovim metodom izbjegli smo računanje Hessiana i njegovog inverza, te osigurali da je on pozitivno definitna matrica tako što je inicijalno postavljen na jediničnu matricu **I** (linija 7), a zatim modificiran formulom koju smo naveli prilikom ranijeg opisivanja ovog metoda (linija 30).

Bitno je istaći da izbor inicijalne vrijednosti Hessiana kao jedinične matrice nije najefikasnija izvedba ovog algoritma. Kao alternativa može se uzeti neki umnožak jedinične matrice, međutim ne postoji dobra strategija za odabir ove vrijednosti. Jedan od efikasnijih, često korištenih načina je modifikacija jedinične matrice I nakon računanja koraka, a prije glavnog ažuriranja matrice, na sljedeći način:

$$H_0 \leftarrow \frac{y_k^T s_k}{y_k^T y_k} I$$

gdje je $y_k = \nabla f(x_k) - \nabla f(x_{k-1})$ i $s_k = x_k - x_{k-1}$. Ova formula skalira vrijednost H_0 tako da je približi vrijednosti inverza Hessiana u početnoj tački, bez potrebe da računamo isti. Upravo ovo smo izveli u liniji 27.

Također, algoritam zaustavlja rad i ispisuje poruku 'NaN value detected' u slučaju da gradijent, ili čak i Hessian, u trenutnoj tački primi besmislenu vrijednost. Kako dalje računanje ovisi o ovoj vrijednosti, nema smisla nastavljati sa izvršavanjem.

2.3. Strogi Wolfe-Powellovi uvjeti

BFGS algoritam je globalno konvergentan u kombinaciji sa nekim neegzaktnim pretraživanjem po pravcu, pri čemu najbolje radi upravo sa korakom koji zadovoljava Wolfe-Powellove uvjete, odnosno stroge Wolfe-Powellove uvjete.

Potrebni ulazni parametri su:
- funkcija **f** čiji minimum tražimo
- trenutna aproksimacija **x0** $\in \mathbb{R}^n$
- vrijednost funkcije u trenutnoj tački **f0**
- vrijednost gradijenta u trenutnoj tački **g0**
- pravac pretrage **p**

Strogi Wolfe-Powellovi uvjeti su zadovoljeni ukoliko je zadovoljen i Armijov uvjet (linija 16) i uvjet ograničenja brzine pada/porasta funkcije f u smjetu p (linija 21). Iako pronaći korak h takav da zadovoljava ove uvjete nije lagan posao, oni obezbjeđuju globalnu konvergentnost ovog metoda. Sve dok oba kriterija nisu zadovoljena, odnosno dok ne dostignemo *maxiters* broj iteracija, poziva se funkcija *zoom*, druga faza algoritma linijskog pretraživanja, koju ćemo opisati u nastavku. Posljednji uslov (linija 25) provjerava nalazimo li se iza minimuma.

Dakle, radimo sa vrijednostima h (tražena vrijednost koraka), h_max (maksimalna vrijednost koraka), h_i i h_im1 (vrijednost koraka u trenutnoj, odnosno prethodnoj, iteraciji), te dphi i dphi0 (umnožak gradijenta u trenutnoj, odnosno početnoj, tački i pravca kretanja). Sve dok odgovarajući korak nije pronađen, ažuriramo ga tako što biramo vrijednost h iz intervala (h_im1, h_max) (linija 31).

Prikažimo implementaciju ovog algoritma u programskom paketu Scilab:

Strogi Wolfe-Powell

```
1   function h = strong_wolfe_powell(f, x0, f0, g0, p)
2       c1 = 1e-4;
3       c2 = 0.9;
4       h_max = 2.5;
5       h = 1;
6       h_im1 = 0;
7       h_i = h;
8       f_im1 = f0;
9       dphi0 = g0*p;
10      i = 1;
11      max_iters = 20;
12      while 1
13          x = x0 + h_i*p;
14          f_i = f(x);
15          g_i = numderivative(f, x);
16          if (f_i >f0 + c1*dphi0*h_i) | | ((f_i >=f_im1) && (i >1))
17              h = zoom(f, x0, f0, g0, p, h_im1, h_i);
18              break;
19          end
20          dphi = g_i*p;
21          if (abs(dphi) <= -c2*dphi0)
22              h = h_i;
23              break;
24          end
25          if (dphi >= 0)
26              h = zoom(f, x0, f0, g0, p, h_i, h_im1);
27              break;
28          end
29          h_im1 = h_i;
30          f_im1 = f_i;
31          h_i = h_i + 0.8*(h_max - h_i);
32          if (i >max_iters)
33              h = h_i;
34              break;
35          end
36          i = i + 1;
37      end
38  endfunction
```

Konačno, funkcija *zoom* prima parametre:

- funkcija **f** čiji minimum tražimo
- trenutna aproksimacija $\mathbf{x0} \in \mathbb{R}^n$
- vrijednost funkcije u trenutnoj tački **f0**
- vrijednost gradijenta u trenutnoj tački **g0**
- pravac pretrage **p**
- donju (h_low) i gornju (h_high) granicu intervala

Traženu vrijednost koraka h tražimo metodom polovljenja intervala (linija 8) i zatim provjeravamo već navedene kriterije strogih Wolfe-Powell uvjeta (linija 14 i 18). Ovaj interval (h_low, h_high) šaljemo iz prethodne funkcije i osiguravamo da se unutar njega nalazi korak koji zadovoljava date uvjete. Vrijednosti c1, c2, dphi i h_i imaju isto značenje kao i za prethodnu funkciju. Ovakva implementacija glasi:

Zoom

```
1   function h = zoom(f, x0, f0, g0, p, h_low, h_high)
2       c1 = 1e-4;
3       c2 = 0.9;
4       i = 1;
5       max_iters = 20;
6       dphi0 = g0*p;
7       while 1
8           h_i = 0.5*(h_low + h_high);
9           x = x0 + h_i*p;
10          f_i = f(x);
11          g_i = numderivative(f, x);
12          x_low = x0 + h_low*p;
13          f_low = f(x_low);
14          if ((f_i >f0 + c1*dphi0*h_i) || (f_i >=f_low))
15              h_high = h_i;
16          else
17              dphi = g_i*p;
18              if (abs(dphi) <= -c2*dphi0)
19                  h = h_i;
20                  return;
21              end
22              if (dphi * (h_high - h_low) >= 0)
23                  h_high = h_low;
24              end
25              h_low = h_i;
26          end
27          if (i >max_iters)
28              h = h_i;
29              break;
30          end
31          i = i + 1;
32      end
33  endfunction
```

Ukoliko se nađemo iza minimuma (linija 22), funkcija reguliše situaciju tako da za gornju granicu intervala postavlja trenutnu donju, a pronađeni korak prilikom te iteracije postavlja za donju granicu. Algoritam se završava ukoliko se pronađe odgovarajući korak (linija 20) ili ukoliko prođe kroz *max_iters* broj iteracija (linija 29). Ovo se dešava kada se približimo rješenju, te razlika između dvije susjedne tačke postane neprimjetna.

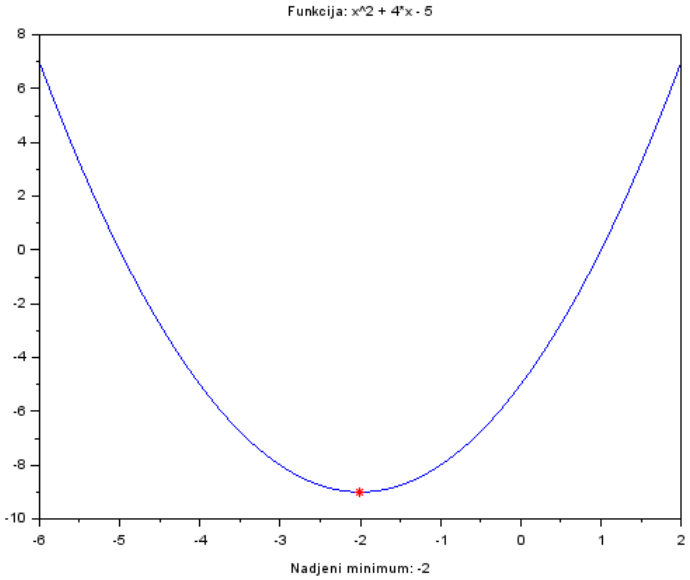
2.4. Rezultati testiranja algoritma

Opisani algoritam je globalno konvergentan, odnosno, ukoliko minimum postoji, algoritam ga pronalazi u određenom broju iteracija, međutim nemamo garanciju da će pronađen minimum biti globalni s obzirom da je naše poznavanje o funkciji najčešće lokalno. Prikažimo to na par primjera.

2.4.1 Jednodimenzionalne funkcije

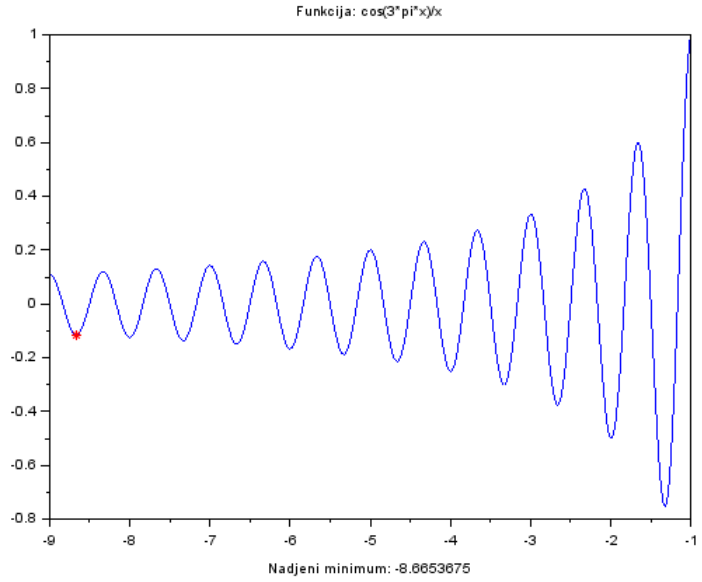
BFGS algoritam se skoro nikada ne koristi za pronalazak minimuma ovih funkcija jer se u ovom slučaju svodi na metod sekante primijenjen na rješavanje jednačine $f'(x) = 0$. Pored toga, inicijalna vrijednost matrice H uzeta kao jedinična matrica stvara probleme u prvoj iteraciji, gdje se pravac zapravo računa po formuli $p = -H * \nabla f(x) = -\nabla f(x)$ što nas može odvesti daleko od početne tačke. Ovaj problem se javlja i pri razmatranju višedimenzionalnih funkcija, ali ga je u ovom slučaju najlakše primijetiti.

Uzimo običnu parabolu, funkciju oblika $y = x^2 + 4x - 5$ čiji se jedini, ujedno i globalni, minimum nalazi u tački x = -2. Krenemo li iz tačke x = 13, algoritam će pronaći ovu vrijednost u tačno dvije iteracije. Isti rezultat imamo krenemo li iz tačke x = 133, te x = 1333, itd. Prva iteracija ne daje rezultat jer se provjerava minimum u početnoj aproksimaciji, ali već u drugoj iteraciji daje rezultat sa traženom tačnosti. Prikažimo ovo grafički:



Ukoliko uzmemo navedenu funkciju, samo prvi koeficijent pomnožimo sa -1, dobijemo funkciju $y = -x^2 + 4x - 5$ koja nema minimum, algoritam će proći kroz zadani broj *maxiters* iteracija i ispisati poruku 'Minimum cannot be found within given number of iterations or it does not exist!'.

Dalje, uzmimo funkciju $y = \frac{cos(3\pi \cdot x)}{x}$ koja ima beskonačno mnogo minimuma. Ukoliko kao početnu aproksimaciju uzmemo tačku $x = -1.1$ algoritam nakon 6 iteracija pronalazi minimum u tački -8.6653675, iako postoji minimum u tački -1.324854. Prikažimo ovo grafički:



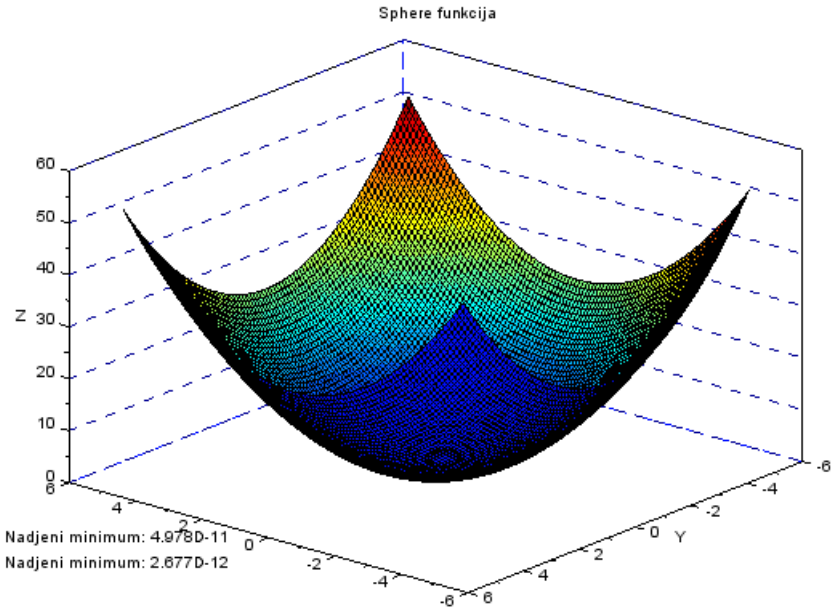
Ovo je posljedica odabira jedinične matrice kao inicijalne vrijednosti Hessiana.

2.4.2 Dvodimenzionalne funkcije

Funkcije na koje ćemo se osvrnuti u ovom odjeljku su poznate funkcije korištene za testiranje algoritama za optimizaciju. I ovu skupinu funkcija započnimo sa najjednostavnijom između njih, odnosno **sfernom funkcijom** čija je formula

$$f(x, y) = x^2 + y^2$$

i grafik:

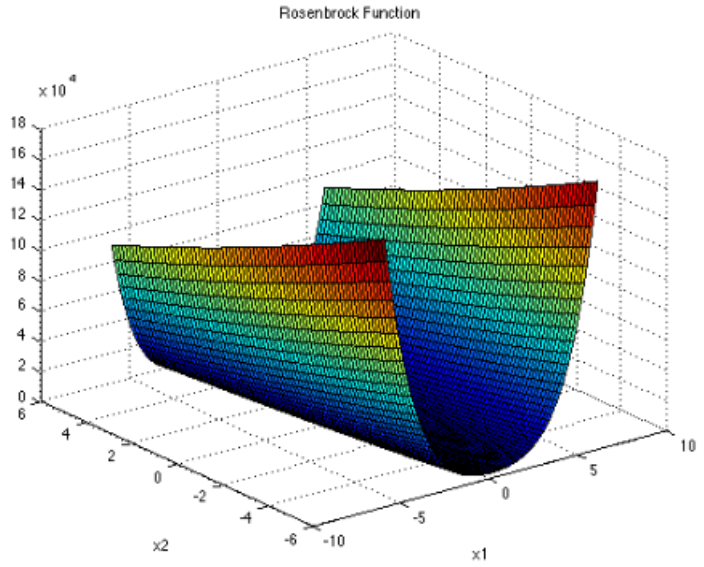


Vidimo da se globalni minimum ove funkcije nalazi u tački (0, 0), koju algoritam vrlo brzo pronalazi. Slično kao i za parabolu, BFGS daje tačan rezultat krenuvši iz ma koje tačke već u drugoj iteraciji.

Testirajmo sada **Rosenbrockovu funkciju** koja predstavlja standardni test za testiranje algoritama za optimizaciju, a čija formula glasi:

$$f(x, y) = b * (y - x^2)^2 + (a - x)^2, \quad a, b \in \mathbb{R}$$

Naziva se još i banana funkcija zbog oblika nivo linija. Uzmemo li vrijednosti a=1 i b=100 njen grafik ima oblik:



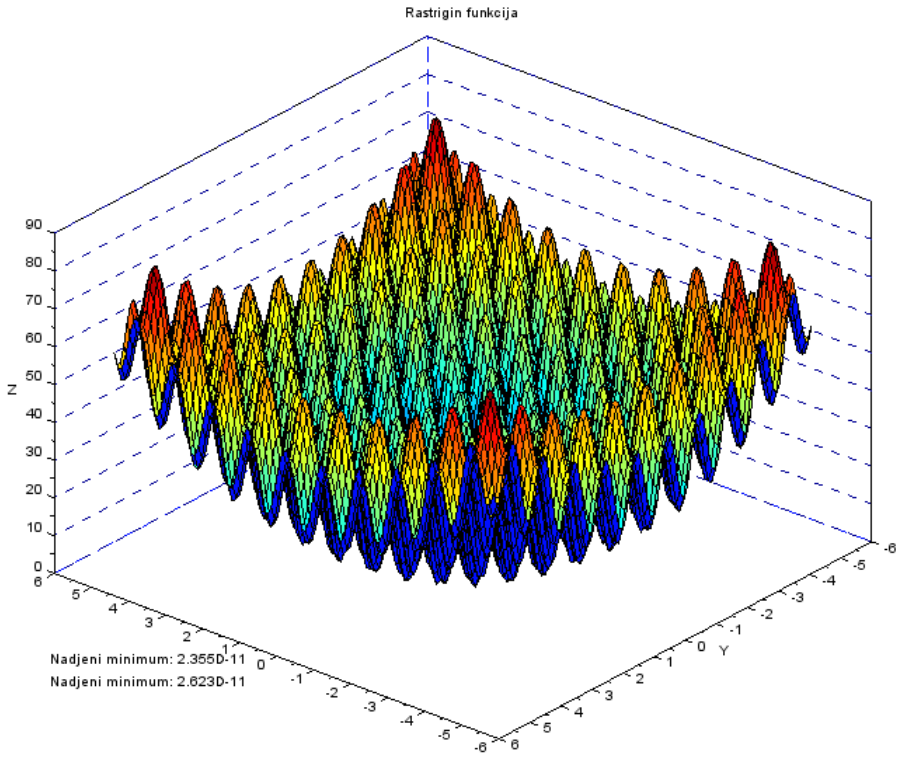
Kao što vidimo, njen globalni minimum (generalno tačka (a, a), konkretno (1, 1)) se nalazi u uskoj, paraboličnoj dolini koju je lako pronaći, ali konvergenciju prema minimumu je vrlo teško postići. Uzmimo nekoliko različitih tačaka i prikažimo broj iteracija do minimuma za svaku u sljedećoj tabeli:

Tačka (x, y)	Broj iteracija
(2, 2)	42
(-3, -3)	40
(22, 54)	87
(-72, 83)	101
(8, -13)	38
(110, 130)	105
(112, 11)	67
(544, 999)	202

Još jedna poznata funkcija za testiranje ovih algoritama je **Rastriginova funkcija**. Za razliku od Rosenbrockove, ova funkcija ima veliki broj lokalnih i jedan globalni minimum u tački (0, 0). Njena formula glasi:

$$f(x, y) = 20 + x^2 + y^2 - 10 * (cos(2\pi \cdot x) + cos(2\pi \cdot y))$$

a grafik:



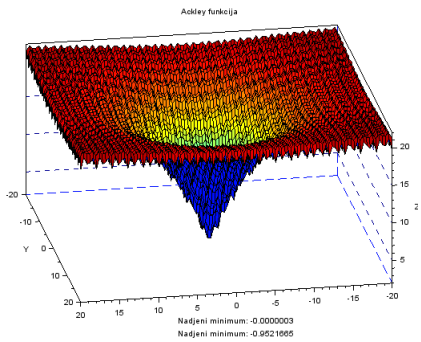
Iako ih ima mnogo, ovi minimumi su ravnomjerno raspoređeni i algoritam veoma brzo konvergira ka jednom, iako često dosta udaljenom od početne aproksimacije.

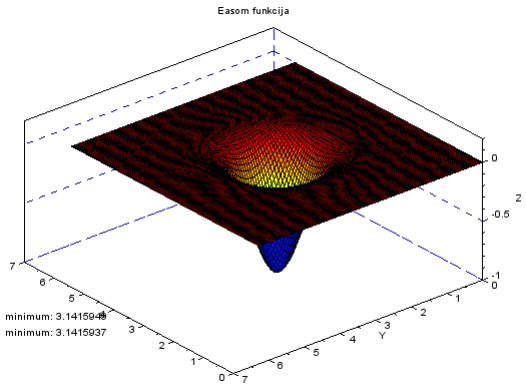
Prikažimo sada dvije dosta slične funkcije, Ackleyevu i Easomovu. Obje su neprekidne, nekonveksne i multimodalne. Formule im respektivno glase:

$$f(x,y) = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(cos(2\pi\cdot x)+cos(2\pi\cdot y))} + 20 + e$$

$$f(x,y) = -cos(x)cos(y)e^{-(x-\pi)^2-(y-\pi)^2}$$

a grafici im respektivno imaju oblik:



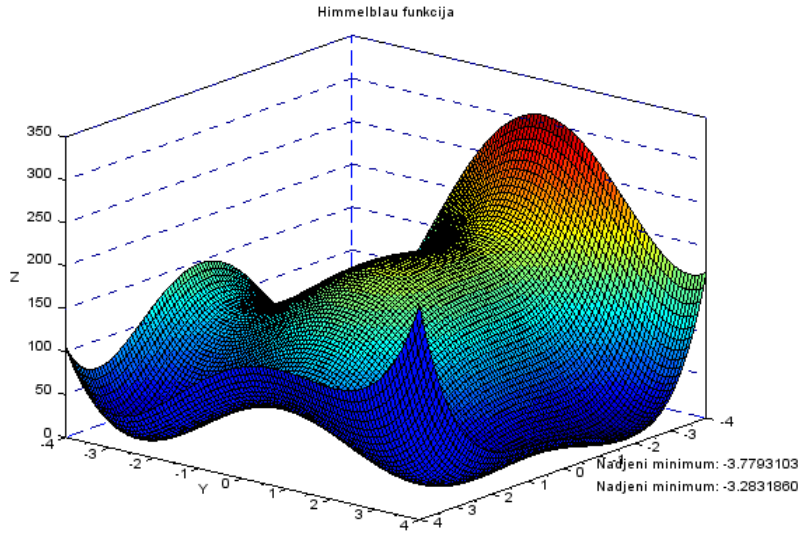


Globalni minimum Ackleyeve funkcije nalazi se u tački (0, 0), a Easomove u (π, π). Pored ovoga, funkcije imaju veliki broj lokalnih minumuma koji predstavljaju veliki problem svim algoritmima za optimizaciju, pa i našem. Pored ovih minimuma, dio funkcija oko globalnog minimuma je prilično ravan, što dodatno otežava pretragu. Za obe ove funkcije algoritam se zaustavi u nekom od lokalnih minimuma blizu početne tačke, i to odmah u drugoj iteraciji, osim u slučaju kada se kao početna aproksimacija da tačka u blizini globalnog minimuma. U ovom slučaju, Wolfe-Powellovi uvjeti daju dobar korak koji često dovede do globalnog minimuma.

Prikažimo još neprekidnu, nekonveksnu i multimodalnu **Himmelblauevu funkciju** čija formula glasi:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

a grafik joj ima oblik:



Funkcija ima 4 globalna minimuma u tačkama (3, 2), (-2.8051181, 3.131312), (-3.779310, -3.283186) i (3.584428, -1.848126). Algoritam daje veoma dobre rezultate i dosta brzo konvergira ka nekom od ova 4 minimuma. Također, veoma je teško predvidjeti u kojem tačno minimumu će algoritam prekinuti izvršavanje, osim ako kao početnu aproksimaciju damo tačku dosta blizu jednom od njih.

U prilogu se nalaze rezultati testiranja još nekoliko sličnih funkcija, koje ovdje nećemo opisivati.

2.4.3 Višedimenzionalne funkcije

Većina prethodno spomenutih dvodimenzionalnih funkcija se može generalizirati u višedimenzionalni slučaj, za koje algoritam daje slične rezultate, proporcionalne s brojem promjenljivih. Kako smo ih već opisali, u ovom odjeljku razmotrimo drugu, karakterističnu funkciju za testiranje problema optimizacije.

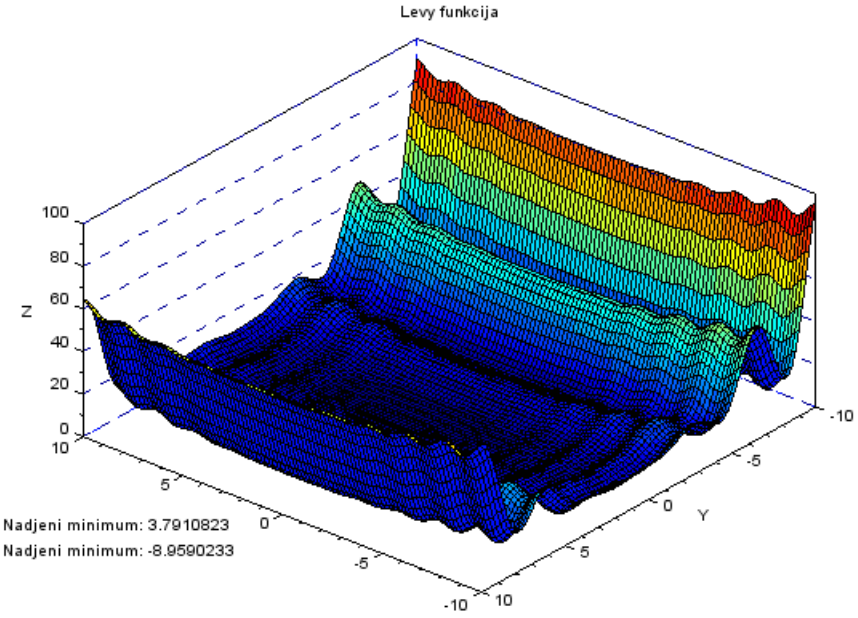
Formula **Levyjeve funkciju** glasi:

$$f(x) = sin^2(\pi w_i) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10sin^2(\pi w_i + 1)] + (w_d - 1)^2[1 + sin^2(2\pi w_d)]$$

gdje je d broj promjenljivih i

$$w_i = 1 + \frac{x_i - 1}{4}, \quad i = 1, 2, ..., d$$

Kako ne možemo prikazati ovu funkciju u višedimenzionalnom slučaju, prikažimo njen grafik u slučaju dvije promjenljive:



Ova funkcija ima globalni minimum u tački **x** = (1, 1, ..., 1), ali i dosta lokalnih minimuma zbog kojih algoritam teško završi u globalnom.

Još jedna poznata višedimenzionalna funkija za testiranje je **Hartmann-ova 6D funkcija**, koja ima 6 lokalnih minimuma, pri čemu je globalni

$$f(\mathbf{x}^*) = -3.32237,$$

gdje je

$$\mathbf{x}^* = (0.20169, 0.150011, 0.476874, 0.275332, 0.311652, 0.6537)$$

Navedeni BFGS algoritam uspije naći globalni minimum u 30 iteracija za sljedeću početnu tačku:

$$\mathbf{x}^* = (-1, 0.33, 0.8, -0.53, 0.22, 1)$$

Naravno, u ovisnosti od početne tačke broj iteracija može biti znatno manji odnosno veći.

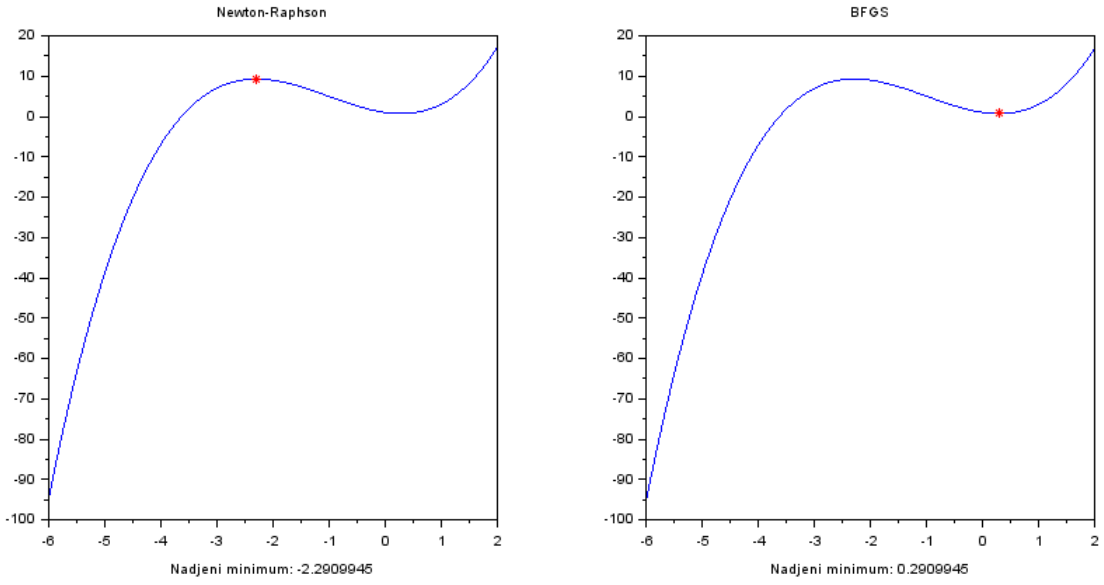
3. Primjene algoritma u praksi

Generalno, problem minimizacije ima velike primjene u praksi. Investitori, proizvođači i inženjeri teže maksimalnoj efikasnosti u njihovom radu. Ovo postižu tako što odrede neki cilj koji može biti profit, vrijeme ili neka druga veličina. Dostizanje ovog cilja mora ovisiti o jednom ili nekoliko faktora, koje posmatramo kao varijable. Sve ovo se svodi na problem optimizacije n-dimenzionalne funkcije, s tim da se problem maksimizacije vrlo jednostavno svodi na minimizaciju.

Pored toga, ove probleme možemo primijetiti i u prirodnim procesima. Zakon o minimalnoj potencijalnoj energiji govori o tome da svaki sistem čestica, ili skupina istih, teži da zauzme takav položaj i raspored da njegova ukupna potencijalna energija bude minimalna. Svjetlosne zrake putuju onim putem kojim stižu u minimalnom intervalu vremena. Molekule u izoliranom sistemu međusobno reaguju tako da im ukupna potencijalna energija bude minimalna. Ovakvih primjera ima još mnogo, sama priroda optimizira.

Konkretno, opisani algoritam rješava dosta navedenih problema, međutim, klasični Newtonov algoritam se vrlo rijetko koristi u praksi zbog svojih mnogih nedostataka. Jedan od glavnih razloga je taj što nemamo nikakvu garanciju da radimo sa silaznim pravcem, pa algoritam često terminira u nekom od lokalnih maksimuma, ili u nekoj sedlastoj tački.

Razmotrimo naizled jednostavnu funkciju $y = x^3 + 3x^2 - 2x + 1$ koja ima jedan lokalni minimum u tački x = 0.2909944, te jedan lokalni maksimum u tački x = -2.2909944. Ukoliko funkciji *Newton_Raphson* proslijedimo neku tačku iz intervala (-2.2909944, 0.2909944) algoritam nam kao rezultat u većini slučajeva vrati upravo lokalni maksimum. S druge strane, modificirani metod BFGS daje lokalni minimum. Prikažimo ovo grafički:



Zanimljivo je spomenuti još neke probleme s ovom funkcijom. Naime, zbog naglog rasta, čak i metod BFGS za svaku vrijednost početne aproksimacije ($x > 0.8$) "preskoči" lokalni minimum i nastavi pretragu iza lokalnog maksimuma. Međutim, funkcija nakon ove vrijednosti samo opada, tako da algoritam terminira. Ovo je još jedan od razloga zbog kojeg se ovaj metod rijetko koristi sa jednodimenzionalnim funkcijama.

S druge strane, većina softverskih paketa za minimizaciju koristi upravo ovaj algoritam, ili se korisniku barem daje mogućnost njegovog korištenja u slučaju da isti nije postavljen kao *default* način. Konkretan primjer je programski paket korišten za implementaciju, Scilab, koji posjeduje predefinisani funkciju *optim* u ovu svrhu.

Vidjeli smo da i ovaj algoritam zakaže u nekim situacijama, pa postavljamo pitanje postoji li univerzalan algoritam koji daje optimalne rezultate. Odgovor je odričan. U praksi se često ne koristi konkretno jedan, nego optimizaciju vrši kombinacija većeg broja algoritama. Sve ovo ovisi o prirodi problema. Također, bitno je prepoznati ukoliko je došlo do greške, te iskorititi skupljeno znanje o problemu u svrhu poboljšanja rješenja.

4. Zaključak i diskusija

Klasični Newton metod, njegove modifikacije kao i Quasi-Newton metode (BFGS specifično) predstavljaju veoma efikasan način pronalaska minimuma za funkcije više nezavisnih promjenljivih. Ne postoji generalno dobra strategija kada koji algoritam koristiti prilikom rješavanja problema optimizacije s obzirom da postoji širok spektar istih a ovdje su razmotreni samo metodi Newtonovog tipa.

Uprkos tome pokazano je da su metodi ovog tipa veoma brzi i efikasni ukoliko se isti implementiraju na korektan način.

S obzirom da smo se bavili Newton-Raphson metodom i BFGS u nastavku navodimo potencijalne prednosti BFGS (u odnosu na klasični metod):

- potrebni su samo prvi izvodi (drugi)
- pozitivna definitnost Hessiana implicira silazni pravac (Hessian ne mora biti pozitivno definitan)
- Vremenska kompleksnost $O(n^2)$ ($O(n^3)$)

U algoritmima prezentiranim u ovom radu ostavljeno je dosta prostora za poboljšanje jer isti predstavljaju samo kostur za neku bolju i kompletniju implementaciju. Sam algoritam linijskog pretraživanja koji inkorporira sve nužne značajke za postizanje globalne konvergentnosti i bržeg terminiranja može biti poprilično težak zadatak. Za još bržu pretragu prikladnog koraka unutar intervala umjesto bisekcije preporučuje se i korištenje interpolacije drugog reda, eventualno trećeg. Također, može se desiti da, kako se približavamo rješenju, dvije uzastopne vrijednosti funkcija u dvije uzastopne iteracije mogu biti neprimjetne u aritmetici konačne preciznosti. Postoji još nekoliko stvari na koje treba obratiti pažnju za postizanje optimalnog rješenja. Jedna od najboljih implementacija algoritma linijskog pretraživanja je implementacija Moore-a i Thuente-a. Pogledati više na [3].

Postavlja se i pitanje koliko je formula za ažuriranje Hessiana pouzdana, odnosno da li ista može uzrokovati loše rezultate. S obzirom da se radi o aritmetici konačne preciznosti jasno je da ne možemo očekivati perfekciju. Pa tako i algoritam prezentiran iznad posjeduje liniju koda koja terminira isti ukoliko se detektuje NaN vrijednost što se može desiti recimo ukoliko unutrašnji produkt $y_i s_i$ postane veoma mal ali pozitivan pri čemu će tada Hessian sadržavani ekstremno velike vrijednosti. U takvim slučajevima treba se voditi logikom *"An ounce of prevention is better than a pound of cure"*. Također, performanse BFGS algoritma mogu poprilično degradirati ukoliko se linijska pretraga ne zasniva na Wolfe-Powell uvjetima. Za više informacija predlažemo [1].

Bibliografija

[1] Jorge Nocedal, Stephen J. Wright: *"Numerical Optimization"*, 2^{nd} *edition*, Springer, New-York, 2006.

[2] Željko Jurić: *"Numerički algoritmi"*, Elektrotehnički fakultet Univerziteta u Sarajevu, Sarajevo, 2018.

[3] Jorge J. More and David J. Thuente: *Line Search Algorithms with Guaranteed Sufficient Decrease*, ACM Transactions on Mathematical Software, Vol. 20, No. 3, September 1994.

[4] R.Fletcher: *Practical Methods of Optimization*, University of Dundee, Scotland, UK, 1980.

[5] J.E. Dennis, Jr., Robert B. Schnabel: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, 1996.

[6] Robert M. Freud: *Newton's Method for Unconstrained Optimization*, Massachusetts Institute of Technology, 2004.