Gradimir V. Milovanović
Predrag S. Stanimirović

SIMBOLIČKA IMPLEMENTACIJA NELINEARNE OPTIMIZACIJE

Predgovor

Motivacija za izradu ove knjige je odsustvo slične literature, kako na srpskom jeziku, tako i u svetskim razmerama. Glavni cilj ove knjige jeste simbolička implementacija osnovnih metoda nelinearnog programiranja. U navedenoj literaturi se može naći implementacija optimizacionih metoda koji su opisani u ovoj knjizi. Međutim, ti programi su pisani u proceduralnim programskim jezicima, najviše u jezicima FORTRAN *i* C*. U ovoj knjizi je pokazana efikasnost implementacije tih metoda pomoću neproceduralnih programskih jezika* MATHEMATICA *i* LISP*. Ideja da se minimum i maksimum ciljne funkcije izračunavaju u funkcionalnim programskim jezicima zaista je prirodna. Takođe, u funkcionalnim programskim jezicima odsustvuje implementacija većine metoda matematičkog programiranja.*

Prva glava je uvodnog karaktera i sadrži neophodne pojmove, definicije i poznate stavove. Druga glava sadrži implementaciju glavnih metoda bezuslovne optimizacije, kako negradijentnih, tako i gradijentnih metoda. Takođe, u drugoj glavi je opisana i implementacija osnovnih metoda globalne optimizacije. U trećoj glavi opisana je implementacija metoda uslovne nelinearne optimizacije i višekriterijumske leksikografske optimizacije. U četvrtoj glavi je opisana primena metoda gradijentne optimizacije prvog i drugog reda u izračunavanju najmanje-kvadratnog rešenja i najmanje-kvadratnog rešenja minimalne norme.

Izrada programa koji su navedeni u knjizi zahtevala je mnogo napora i vremena. U tome smo imali pomoć velikog broja saradnika kojima se ovom prilikom zahvaljujemo. Veliki broj programa u LISP*u napisao je mr Svetozar Rančić, asistent Prirodno-matematičkog fakulteta u Nišu, koji je magistrirao iz ove oblasti* [40]*. Takođe, mr Milan Tasić, asistent Tehnološkog fakulteta u Leskovcu, napisao je i testirao veći broj programa. Neki od ovih programa mogu se naći i u njegovoj magistarskoj tezi* [63]*. Dr Miroslav Ristić, asistent Prirodno-matematičkog fakulteta u Nišu, izradio je neke programe i razradio njihovu grafičku ilustraciju. U ovoj knjizi ugrađeni su i neki plodovi višesatnih konsultacija sa dr Nebojšom Stojkovićem, asistentom Ekonomskog fakulteta u Nišu. U pisanju programa u implementaciji simpleks metoda linearnog programiranja veliki doprinos dali su Ivan Stanković i Marko Milošević, asistenti pripravnici Prirodno-matematičkog fakulteta u Nišu. U izradi četvrte glave posebnu zahvalnost dugujemo dr Draganu Đorđeviću, docentu Prirodno-matematičkog fakulteta u Nišu, koji je pomogao da se navedeni rezultati, prvobitno formulisani za kompleksne matrice,*

prošire na Hilbertove prostore. Takođe smo zahvalni i dr Zoranu Budimcu, profesoru Prirodno-matematičkog fakulteta u Novom Sadu, na korisnim sugestijama u vezi implementacije metoda na programskom jeziku LISP.

Ova knjiga može biti korisna svima onima koji se bave matematičkim programiranjem, a posebno studentima redovnih i poslediplomskih studija na prirodno-matematičkim, tehničkim i drugim fakultetima na kojima se izučava ova problematika. Knjiga sadrži i niz novih rezultata dobijenih u poslednje vreme u radovima [48–59]. *Nadamo se da će ova knjiga pomoći mnogima koji žele da se naučno bave ovom problematikom, kao i onima koji rešavaju probleme optimizacije realnih sistema.*

Profesori Vera Vujčić-Kovačević i Ljubiša Kocić, u svojstvu recenzenata, pomogli su svojim sugestijama u poboljšanju kvaliteta knjige. Koristimo ovu priliku da im se najsrdačnije zahvalimo za trud koji su uložili.

Niš, avgust 2002. godine *Autori*

Sadržaj

III G L A V A

USLOVNA OPTIMIZACIJA

IV G L A V A

OPTIMIZACIJA I LINEARNI SISTEMI

I GLAVA

Uvod

Ova glava sadrži uvodna razmatranja u dva različita smisla. U prvom poglavlju je ukratko opisan neophodan matematički aparat za razumevanje suštine opisanih i implementiranih metoda. Takođe, prvo poglavlje sadrži globalne aspekte o suštini optimizacionog zadatka, podelama i osobinama metoda optimizacije. Drugo poglavlje predstavlja uvod za jedan sasvim novi pristup u implementaciji metoda optimizacije - pristup baziran na funkcionalnom stilu programiranja.

1. MATEMATIČKA PRIPREMA

1.1. Matematički model

U procesima upravljanja postoje tri osnovna pojma [34]:

1. *funkcija cilja* ili kriterijum upravljanja;

2. *skup ograničenja*;

3. *matematički model*.

Razmotrićemo ukratko svaki od ovih pojmova.

1. Osnovni preduslov svakog upravljačkog zadatka jeste postojanje definisanog cilja. U opštem slučaju jedan problem se može sastojati iz više različitih celina. Rešavanje takvog programa u celini naziva se ostvarivanje globalnog cilja. Vreme, kvalitet, troškovi ili efikasnost jesu najčešći ciljevi. U većini slučajeva postavljeni ciljevi na prvi pogled izgledaju paradoksalno. Preciznije, obično između postavljenih ciljeva postoje suprotnosti, protivurečni zahtevi i ograničenja, pa se udovoljavanje globalnom cilju svodi na rešavanje tih protivurečnosti. Drugim rečima, ako se može naći praktično rešenje tih protivurečnih uslova, tada je moguće i ostvarenje postavljenog cilja. Potrebno je da globalni cilj ostavi određenu slobodu ciljevima nižih hijerarhijskih nivoa. Glavni cilj bi trebalo da odredi opšte pravce, a ne i da reguliše najsitnije detalje.

Kod definisanja funkcije cilja najčešće se polazi od verbalnog opisa cilja a zatim se postepeno prelazi na analitičko formulisanje, sve dok se ne dobije njen potpun matematički oblik. Nije uvek lako naći matematički oblik funkcije, naročito kada su u pitanju složeni upravljački zadaci.

U matematičkom obliku funkcija cilja se izražava nekom funkcijom $Q(\mathbf{x})$, gde je $\mathbf{x} = (x_1, \ldots, x_n)$ n-dimenzionalni vektor. U ovom slučaju, $Q(\mathbf{x})$ je funkcija više promenljivih, za koju potrebno odrediti ekstremnu vrednost. U nekim slučajevima, umesto funkcije $Q(\mathbf{x})$ koristi se funkcional, što je u praksi ređi slučaj.

Funkcija cilja se minimizira ako su njome izraženi troškovi, utrošak materijala, vreme izvršenja zadatka, gubitak u proizvodnji, vreme transporta, utrošak goriva, itd. Ako funkcija cilja odražava dobit, pouzdanost, dohodak, itd, ona se maksimizira.

2. Skup ograničenja, označen sa L, utvrđuje se za svaki upravljački zadatak posebno. Skup ograničenja je sistem od m jednačina i (ili) nejednačina od promenljivih $x_1, \ldots, x_n$ (koje se koriste i u funkciji cilja). Generalno, skup ograničenja predstavlja skup hiper površina i (ili) hiper ravni n-dimenzionalnog prostora, kojima je ograničen domen (označen sa D). Iz domena D se bira onaj vektor $\mathbf{x}$ koji inicira ekstremnu vrednost ciljne funkcije $Q(\mathbf{x})$. Sve moguće vrednosti za $\mathbf{x}$ iz domena D nazivaju se *dopustivim planom*, a onaj vektor $\mathbf{x}$ koji obezbeđuje da funkcija $Q(\mathbf{x})$ ima ekstremnu vrednost naziva se *optimalnim planom*.

Pored skupa ograničenja L postoji i prirodni skup ograničenja koji se sastoji u tome da komponente vektora $\mathbf{x}$ moraju biti nenegativne veličine tj. $x_i \geq 0$, $i = 1, \ldots, n$. U sistemu ograničenja mogu nastupiti sledeći slučajevi:

(a) Sistem L može biti protivurečan, što znači da ne postoji dopustiv plan $\mathbf{x}$ koji zadovoljava sva ograničenja.

(b) Sistem L nije protivurečan, ali je oblast D neograničena. Tada postoji mogućnost određivanja optimalnog plana $\mathbf{x}$ samo ako je funkcija $Q(\mathbf{x})$ ograničena u neograničenoj oblasti D.

(c) Sistem L nije protivurečan i oblast D je ograničena. Tada se optimalni plan može odrediti u svim slučajevima, sem ako je funkcija $Q(\mathbf{x})$ neograničena u ograničenoj oblasti D.

3. Pravilna postavka zadatka je složen zadatak, i samo u retkim slučajevima se zadatak može postaviti u prvom pokušaju. Često se vrše izmene i dopune pa čak i forma i priroda modela. Olakšavajuća okolnost je što za određen broj zadataka ili za neke njihove delove postoje gotovi modeli tako da se kod novih zadataka može koristiti modifikacija takvih gotovih modela.

Postoje dva postupka za nalažanje rešenja matematičkih modela:

(a) Analitičko rešenje.

(b) Numeričko rešenje (primenom približnih numeričkih metoda).

Numeričko rešavanje se mnogo više primenjuje u praksi.

Konstrukcija matematičkog modela M za neki proces upravljanja sastoji se u definiciji funkcije cilja $Q(\mathbf{x})$, skupa ograničenja L kao i prikupljanje i sređivanje potrebnih polaznih podataka. Kaže se da trojka (Q, L, M) karakteriše određeni upravljački zadatak.

Primena matematičkih metoda u procesu primene odluka i njegovih izvršenja može se podeliti u tri faze: U prvoj fazi se konstruiše matematički model problema. U drugoj fazi se vrši izbor algoritma za implementaciju matematičkog modela, izrada programa za računar i njegovo testiranje. U okviru druge faze matematički model se usavršava, a ako nedostaju potrebni podaci vraća se prvoj fazi na doradu i proveru da li su predložene promene saglasne sa suštinom zadatka. U trećoj fazi se ispituje opravdanost ili neopravdanost predloženog rešenja. Neopravdanost može da nastupi npr. ako neki na izgled sporedni, a u suštini bitni faktori nisu uzeti u obzir. Ako u trećoj fazi postoje primedbe one se vraćaju na otklanjanje prvoj i drugoj fazi. U svim ovim fazama rad se odvija po grupama koje sačinjavaju stručnjaci različitih profila.

Odluke koje se donose na različitim nivoima upravljanja mogu se klasifikovati na različite načine. Jedna od klasifikacija odluka je sledeća:

(a) Ako pri upotrebi određenog algoritma dobijamo potpuno određeni rezultat, sa verovatnoćom koja je jednaka jedinici, kažemo da je odluka *deterministička*.

(b) Ako su pri postavci zadatka neki parametri slučajne veličine sa poznatom raspodelom, tada i rezultat koji se dobija ima određenu verovatnoću svoje verodostojnosti. Takva upravljačka odluka naziva se *probabilističkom*.

(c) Ako na donošenje odluke utiču veličine koje zavise od okruženja ili od protivnika koji ugrožavaju donosioce odluka, takve odluke se nazivaju *strategijskim*.

(d) Ako se odluke zasnivaju na rešavanju zadataka čiji su parametri približno procenjene veličine ili su to slučajne veličine sa nepoznatom raspodelom, tada se takve odluke nazivaju *statističkim*.

Nekada su u mnogim naučnim disciplinama, kakve su na primer ekonomija ili biologija, eksperimenti bili nedopustivi. Međutim, primenom odgovarajućih matematičkih modela koji definišu problem i razvojem metoda opti-

mizacije, stvara se mogućnost primene eksperimenata i u ovim oblastima. Tako se analizom matematičkog modela mogu pratiti uticaji promene bilo kog parametra na krajnji rezultat i tražiti optimalno rešenje. Na taj način se eksperiment u praksi svodi na eksperiment na modelu. Složenost i priroda matematičkog modela ima veliki uticaj na smanjenje (povećanje) dimenzije zadatka. Pod *dimenzijom zadatka* se podrazumeva ukupan broj jednačina i (ili) nejednačina u skupu L i broj nepoznatih.

1.2. Formulacija optimizacionog zadatka

Optimizacija je postupak nalaženja najboljeg rešenja nekog problema u određenom smislu i pri određenim uslovima [60].

Neophodne pretpostavke za ostvarenje zadatka optimizacije su:

1) *Objekat optimizacije*. Može biti proizvoljni proces, aparat, ljudska delatnost itd.
2) *Kriterijum optimalnosti*, koji se drugačije naziva efikasnost ili *funkcija cilja*. Funkcija cilja može biti, na primer, tehnički rashod, dobit ili čistoća materijala. Najbolja vrednost kriterijuma optimalnosti naziva se *ekstremum* ili *optimum*.
3) *Upravljivost objektom optimizacije*. Za izvršenje procesa optimizacije potrebno je da objekat optimizacije bude upravljiv, odnosno da ima izvestan stepen slobode. Da bi se osiguralo upravljanje objektom optimizacije neophodno je da on ima upravljačke parametre, koji mogu da se menjaju nezavisno jedan od drugih. Time se može definisati skup različitih stanja objekta optimizacije, iz koga se odabira optimalno stanje.
4) *Metod optimizacije*. Za zadati upravljački objekat i formulisani cilj, izražen kroz kriterijum optimalnosti, neophodno je da se izabere metod za izračunavanje optimuma. Nije moguće da se preporuči jedan univerzalni metod za rešavanje svih optimizacionih zadataka. Izbor konkretnog metoda se vrši na osnovu postavljenih ciljeva kao i prirode objekta optimizacije.

Jedna od najvažnijih pretpostavki za rešavanje optimizacionog zadatka je formulacija cilja, što se bazira na subjektivnoj pretpostavci. Pravilno formulisanje cilja neophodno je za pravilno rešavanje optimizacionog zadatka. U praksi se proces optimizacije izvršava na osnovu uprošćenog matematičkog modela procesa. Na osnovu tog modela formira se ciljna funkcija.

Svaki upravljiv objekat se karakteriše parametrima, i to: ulaznim $\mathbf{x} = (x_1, \dots, x_n)$ i izlaznim $\mathbf{y} = (y_1, \dots, y_m)$. Matematički model ovog objekta povezuje njegove parametre sistemom funkcija sledećeg oblika:

$$y_j = f_j(\mathbf{x}), \quad j = 1, \dots, m. \tag{1.2.1}$$

Kriterijum optimalnosti upravljivog objekta ili sistema je funkcija ulaznih i izlaznih parametara: $Q = Q(\mathbf{x}, \mathbf{y})$. Međutim iz (1.2.1) sledi da ciljna funkcija Q zavisi samo od upravljačkih parametara

$$Q = Q(\mathbf{x}) = Q(x_1, \dots, x_n).$$

U primenama je skup upravljačkih parametara $x_i, \quad i = 1, \dots, n$ ograničen, tj. upravljački parametri se menjaju unutar dozvoljenog prostora D_x upravljačkih parametara. Na taj način, upravljački parametri ispunjavaju uslove:

$$xmin_i \leq x_i \leq xmax_i, \quad i = 1, \dots, n \tag{1.2.2}$$

$$\mathbf{x} \in D_x\,. \tag{1.2.3}$$

Zadatak optimizacije definiše se na sledeći način: traži se maksimum ciljne funkcije $Q(\mathbf{x}) = Q(x_1, \dots, x_n)$, pod uslovom $\mathbf{x} \in D_x$.

Ponekad se upravljačkim parametrima mogu nametnuti uslovi definisani drugim funkcijama:

$$\varphi_l(x_1, \dots, x_n) = \varphi_{0_l}, \quad l = 1, \dots, m_1 < n, \tag{1.2.4}$$

$$\psi_j(x_1, \dots, x_n) \leq \psi_{0_j}, \quad j = 1, \dots, m_2. \tag{1.2.5}$$

Uslovi (1.2.4) se nazivaju funkcionalna ograničenja tipa jednakosti, dok se uslovi (1.2.5) nazivaju funkcionalna ograničenja tipa nejednakosti, ili ograničenja zadata oblastima.

Ciljna funkcija koja u dozvoljenoj oblasti upravljačkih parametara ima samo jedan ekstremum naziva se *jednoekstremalna* ili *unimodalna*, a u suprotnom se naziva *mnogoekstremalna* odnosno *multimodalna*.

U slučaju mnogoekstremalnih ciljnih funkcija, globalni ekstremum je najbolja vrednost ciljne funkcije između svih lokalnih ekstremuma. Ciljna funkcija može da bude linearna ili nelinearna, zavisno od toga da li je zavisnost $Q(\mathbf{x}) = Q(x_1, \dots, x_n)$ linearna ili nelinearna.

Zadatak matematičkog programiranja se sastoji u određivanju vektora $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ koji predstavlja rešenje sledećeg zadatka:

$$\begin{aligned} &\text{Minimizirati } Q(\mathbf{x}) \\ &\text{P.O.} \quad g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, m, \\ &\qquad\quad h_j(\mathbf{x}) = 0, \quad j = 1, \dots, p. \end{aligned} \tag{1.2.6}$$

Ako je ciljna funkcija Q linearna i kao su linearne i funkcije $\{g_i, \quad i = 1, \ldots, m\}$ i $\{h_j, \quad j = 1, \ldots, p\}$, sadržane u ograničenjima, tada zadatak (1.2.6) predstavlja *zadatak linearnog programiranja*. Ako je bar jedna od tih funkcija nelinearna, tada se dobijeni problem naziva *zadatak nelinearnog programiranja.* Ako uslovi u (1.2.6) odsustvuju radi se o *bezuslovnoj optimizaciji*, a inače se rešava problem *uslovne optimizacije*.

Algoritam za nalaženje maksimuma može da se iskoristi za nalaženje minimuma:

$$\min Q(\mathbf{x}) = -\max(-Q(\mathbf{x})).$$

Važi i obrnuto:

$$\max Q(\mathbf{x}) = -\min(-Q(\mathbf{x})).$$

1.2.1 LINEARNO PROGRAMIRANJE

Linearno programiranje je jedna od najjednostavnijih metoda za određivanje optimalnog rešenja u raznim zadacima optimizacije. Takvi zadaci se javljaju u različitim granama privrede, u ekonomiji, proizvodnji, obrazovanju, istraživanju itd.

Kao što je već napomenuto, za zadatke linearnog programiranja karakteristična je linearna funkcija cilja $Q(\mathbf{x})$ i skup linearnih ograničenja L. Funkcija $Q(\mathbf{x})$ predstavlja linearnu kombinaciju nepoznatih a L je sistem linearnih jednačina i (ili) nejednačina. Problem se svodi na nalaženje minimuma ili maksimuma linearne funkcije $Q(\mathbf{x})$ pri određenim linearnim ograničenjima. Broj nepoznatih i ograničenja može da bude veoma različit.

U zadacima linearnog programiranja postoje tri kategorije faktora koji učestvuju pri određivanju optimalnog rešenja [34]:

(a) ulazni faktori;

(b) izlazni faktori;

(c) strukturalni faktori.

Ulazni faktori su zadati uslovima privređivanja, proizvodnje, potrebama i troškovima.

Izlazni faktori karakterišu rezultat delatnosti.

Strukturalni faktori karakterišu proces rada, tehnologiju, resurse, itd.

Linearno programiranje je metod određivanja takve kombinacije uzajamno povezanih faktora, koja od niza mogućih kombinacija predstavlja najpovoljniju tj. traži se takva kombinacija koja će pored uslova L zadovoljiti i kriterijum optimalnosti ciljne funkcije.

Formulacija zadatka linearnog programiranja. [8], [23], [34], [63], [65]. Neka je $A = (a_{ij})_{m\times n}$ matrica sa vrstama $V_1, \dots, V_m$ i neka su $b \in \mathbb{R}^m$ i $c \in \mathbb{R}^n$ dati vektori. Matematička formulacija zadatka linearnog programiranja u *opštem obliku* sastoji se u sledećem: Odrediti komponente n-dimenzionalnog vektora $\mathbf{x}$ iz oblasti D za koje funkcija cilja $Q(\mathbf{x})$ dostiže maksimalnu (minimalnu) vrednost. Pri tome je vektor $\mathbf{x} = (x_1, \dots, x_n)$ zadat svojim koordinatama, oblast D je zadata sistemom linearnih jednačina i (ili) nejednačina

$$\begin{aligned} &V_i^T\mathbf{x} = b_i, \quad i \in I_1,\\ &V_i^T\mathbf{x} \geq b_i, \quad i \in I_2,\\ &V_i^T\mathbf{x} \leq b_i, \quad i \in I_3,\\ &x_k \geq 0, \quad k \in J, \end{aligned} \tag{1.2.7}$$

gde je

$$I_1+I_2+I_3=\{1,\dots,m\},\ \ I_1\cap I_2=\emptyset,\ \ I_1\cap I_3=\emptyset,\ \ I_2\cap I_3=\emptyset,\ \ j\subseteq\{1,\dots,n\}.$$

Funkcija cilja $Q(\mathbf{x})$ predstavlja linearnu kombinaciju nepoznatih x_k:

$$Q(\mathbf{x}) = Q(x_1, \dots, x_n) = c_1x_1 + \cdots + c_nx_n = \mathbf{c}^T\mathbf{x} \tag{1.2.8}$$

U slučaju $I_1 = \{1, \dots, m\}$, $J = \{1, \dots, n\}$, ograničenja (1.2.7) i (1.2.8) svode se na tzv. *standardni oblik* linearnog programiranja:

$$\begin{aligned} &\text{Maksimizirati } \mathbf{c}^T\mathbf{x},\\ &\text{P.O. } A\mathbf{x} = b,\\ &\mathbf{x} \geq \mathbf{0} \end{aligned} \tag{1.2.9}$$

Analogno, u slučaju $I_2 = \{1, \dots, m\}$, $J = \{1, \dots, n\}$, ograničenja (1.2.7) i (1.2.8) svode se *simetrični oblik* linearnog programiranja:

$$\begin{aligned} &\text{Maksimizirati } \mathbf{c}^T\mathbf{x},\\ &\text{P.O. } A\mathbf{x} \geq b,\\ &\mathbf{x} \geq \mathbf{0} \end{aligned} \tag{1.2.10}$$

Problem zadat u opštem obliku uvek se može transformisati u eqvivalentan problem u standardnom i simetričnom obliku. Iz opšteg oblika (1.2.7)-(1.2.8)

dobija se standardni oblik (1.2.9) dodavanjem, odnosno oduzimanjem tzv. izravnavajućih promenljivih $s_i > 0$, $i \in I_2 \cup I_3$. Nejednačine oblika $V_i^T \mathbf{x} \geq b_i$, $i \in I_2$ svode se na ekvivalentne jednačine $V_i^T \mathbf{x} - s_i = 0$, $i \in I_2$. Analogno, nejednačine oblika $V_i^T \mathbf{x} \leq b_i$, $i \in I_3$ svode se na jednačine $V_i^T \mathbf{x} + s_i = 0$, $i \in I_3$.

Takođe, problem zadat u opštem obliku (1.2.7)-(1.2.8) može se prevesti u simetričan oblik. Ograničenja tipa jednakost $V_i \mathbf{x} = b_i$ zamenjuju se nejednačinama $V_i^T \mathbf{x} \geq b_i$, odnosno $-V_i^T \mathbf{x} \leq -b_i$, $i \in I_1$. Takođe, množeći proizvoljno ograničenje oblika $V_i^T \mathbf{x} \leq b_i$ sa -1 dobija se ekvivalentno ograničenje u obliku $-V_i^T \mathbf{x} \geq -b_i$.

Na osnovu rečenog zaključuje se da se linearni program može zadati ravnopravno u opštem, standardnom ili simetričnom obliku.
Skup jednačina (1.2.9) se najčešće piše u razvijenom obliku na sledeći način:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \\ x_k \geq 0, \quad k = 1, \ldots, n. & \end{aligned}$$

Tada je konačna oblast D konveksna i ograničena skupom hiper ravni oblika:

$$P_j = \sum_{k=1}^{n} a_{jk}x_k - b_j = 0, \quad j = 1, \ldots, m, m+1, \ldots, m+n. \tag{1.2.11}$$

Ako je skup ograničenja L zadat nelinearnim vezama kaže se da je oblast D ograničena hiper površima. U nekim slučajevima oblast D može biti i neograničena.

Posmatrajmo jednačine (1.2.11) i neka je $j = 1, \ldots, r$, $(m \leq r \leq m+n)$. Bilo kojih n jednačina iz skupa (1.2.11) određuje u n-dimenzionalnom prostoru koordinate jednog vrha poliedra.

Ako pretpostavimo da hiper ravan $Q(\mathbf{x}) = Q(x_1, \ldots, x_n) = c$ nije paralelna ni sa jednom od hiper ravni P_j i ako je oblast D ograničena, onda funkcija cilja $Q(x_1, \ldots, x_n)$ dostiže maksimum (minimum) u jednom od vrhova poliedra. Ako je hiper ravan $Q(x_1, \ldots, x_n) = c$ paralelna bar sa jednom hiper ravni P_j, tada problem može imati beskonačno mnogo rešenja. U tom slučaju, funkcija cilja $Q(\mathbf{x})$ dostiže maksimum (minimum) u određenoj hiper ravni P_j.

Često se funkcija $Q(\mathbf{x})$ takvog oblika naziva *linearnom formom* promenljivih $\mathbf{x} = (x_1, \ldots, x_n)$ koje zadovoljavaju uslove (1.2.7) i (1.2.8). Sistem nejednačina i (ili) jednačina (1.2.7) naziva se *skupom ograničenja* L.

Matrica

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & & & \\ a_{m1} & a_{m2} & & a_{mn} \end{bmatrix}$$

koja je sastavljena od koeficijenata skupa ograničenja,naziva se *matrica ograničenja*.

Vektor

$$A_k = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix}$$

čije su komponente koeficijenti iz skupa ograničenja L uz promenljivu x_k, naziva se k-tim *vektorom skupa* L.

Vektor

$$B_k = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

sa koeficijentima koji predstavljaju slobodne članove iz skupa ograničenja, naziva se *vektor ograničenja*.

Određivanje vektora $\mathbf{x} = (x_1, \ldots, x_n)$ koji zadovoljavaju uslove (1.2.7) naziva se određivanje plana zadatka. Vrednosti $x_1, \ldots, x_n$ iz plana nazivaju se komponentama tog plana. Plan $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ koji obezbeđuje ekstremnu vrednost funkcije cilja $Q(\mathbf{x})$, naziva se *optimalni plan* ili *rešenje zadatka linearnog programiranja*.

Za optimalni plan $\mathbf{x}^*$ je ispunjeno $Q(\mathbf{x}^*) \geq Q(\mathbf{x})$ (u odnosu na bilo koji drugi plan $\mathbf{x} \in D$), u slučaju da je tražen maksimum. U slučaju da se radi o traženju minimuma onda mora biti zadovoljen uslov $Q(\mathbf{x}^*) \leq Q(\mathbf{x})$. Zadaci koji imaju bar jedan optimalni plan $\mathbf{x}^*$ pripadaju klasi rešivih problema.

Označimo kolone matrice A sa $K_1, \ldots, K_n$. Očigledno da se ne umanjuje opštost razmatranja ako se pretpostavi da među ograničenjima nema suvišnih. To znači da je u slučaju $m \leq n$ ispunjen uslov $\text{rang}(A) = m$. Prema tome, postoji bar jedan skup od m linearno nezavisnih kolona matrice A. Svaki maksimalan skup linearno nezavisnih kolona matrice A naziva se baza matrice A. Dve baze su *susedne* ako se razlikuju u jednoj koloni. Neka su $B = \{j_1, \ldots, j_m\}$ indeksi bazičnih kolona matrice A i neka je $A_B =$

$\{K_{j_1}, \ldots, K_{j_m}\}$ bazični minor matrice A. Promenljive x_j, $j \in B$ nazivaju se *bazične*, a preostale *nebazične* u odnosu na izabrane bazične kolone. U odnosu na izabrane bazične kolone, *bazično rešenje* se dobija kada se sve nebazične promenljive izjednače sa nulom i sistem $A\mathbf{x} = \mathbf{b}$ reši po bazičnim promenljivim. Svako bazično rešenje čije su komponente nenegativne naziva se *bazično dopustivo rešenje*, dok se baza A_B nazica *dopustivom bazom*.

Ako je broj ograničenja (1.2.7) veliki, tada postoji veliki broj varijanti koje dolaze u obzir kao rešenje. Zbog toga je potrebno formirati šemu za nalaženje optimalnog plana koji bi se dobio bez ispitivanja svih mogućih varijanti i izvođenja odgovarajućih računskih operacija kojih može biti više od

$$n! \approx \left(\frac{n}{e}\right)^2 \sqrt{2n\pi},$$

gde je n broj nepoznatih iz funkcije cilja $Q(\mathbf{x})$ [34].

Postoje različiti metodi za rešavanje zadataka linearnog programiranja. Jedna od njih je Dantzig-ov metod, koji je poznat pod nazivom *simpleks metod* [8], [23], [34], [63], [65].

Simpleks metod linearnog programiranja je do sada implementiran prvenstveno u proceduralnim programskim jezicima. U [45] je opisana implementacija simpleks metoda u programskom jeziku C. Najčešće su implementacije metoda linearnog programiranja u programskom jeziku FORTRAN [22], [30]. U radu [54] opisana je simbolička implementacija simpleks metoda u programskom paketu MATHEMATICA. U ovoj knjizi je od primarnog interesa implementacija nelinearnog programiranja. Simbolička implementacija simpleks metoda linearnog programiranja opisana je u poglavlju 3.5 treće glave.

U poslednjih dvadesetak godina razvijene su neke modifikacije simpleks metoda. Takođe, sve su popularniji tzv. unutrašnji metodi u rešavanju problema linearnog programiranja [8], [20].

1.2.2. NELINEARNO PROGRAMIRANJE

Svaki upravljački zadatak u kome je funkcija cilja $Q(\mathbf{x})$ i (ili) skup ograničenja L definisan nelinearnim jednačinama i (ili) nejednačinama, predstavlja zadatak nelinearnog programiranja. Optimalno rešenje nelinearnog optimizacionog problema izračunava se nekom od raspoloživih metoda, koja je najadekvatnija za nalaženje konkretnog rešenja.

Za razliku od zadataka linearnog programiranja, zadaci nelinearnog programiranja se ne mogu rešavati primenom nekog univerzalnog metoda (kao što je to simpleks metod za zadatke linearnog programiranja). Za zadatke

nelinearnog programiranja je za svaki konkretan slučaj, u zavisnosti od njegovog matematičkog modela, dimenzija i karaktera nelinearnosti, potreban nov metod ili prilagođavanje nekog od postojećih metoda. U velikom broju slučajeva čak i ne postoji prikladni metod na osnovu kojeg se može naći optimalno rešenje formulisanog zadatka nelinearnog programiranja, što znači da postoji još uvek veliki broj nerešivih ili teško rešivih zadataka nelinearnog programiranja.

Postoji više metoda optimizacije pomoću kojih se mogu rešavati neki zadaci nelinearnog programiranja. Svi ti metodi su specijalizovani za različite tipove zadataka nelinearnog programiranja, koji se formalno razlikuju po obliku matematičkog modela, tj. po obliku i dimenzijama funkcije cilja i skupa ograničenja. Tako, na primer, postoje specijalni metodi za linearna ograničenja i nelinearnu funkciju cilja, za funkcije cilja zadate kvadratnom formom, za celobrojne vrednosti promenljivih, itd. Otuda potiču i neki posebni nazivi za takve specifične zadatke nelinearnog programiranja, kao što su: kvadratno programiranje, celobrojno programiranje, itd.

Zadaci nelinearnog programiranja prekrivaju znatno šire područje upravljačkih zadataka i raznovrsniji su od zadataka koji se svode na primenu linearnog programiranja. Mnogi od njih još uvek nisu rešivi jer ne postoje razvijeni algoritmi čija bi primena dala određene efekte. Primenljivost određenih algoritama procenjuje se na osnovu broja računskih operacija koje treba obaviti u procesu nalaženja rešenja. Neki algoritmi u određenim zadacima nelinearnog programiranja, čak i uz primenu savremenih računara, nisu uvek primenljivi.

Opšta formulacija zadatka nelinearnog programiranja može se iskazati na sledeći način: Naći n-dimenzionalni vektor $\mathbf{x} = (x_1, \dots, x_n)$ kojim je omogućeno da funkcija cilja $Q(\mathbf{x})$ dobija maksimalnu (minimalnu) vrednost, a da pri tome budu zadovoljena ograničenja

$$\begin{aligned} F(\mathbf{x}) \geq 0, \quad G(\mathbf{x}) \leq 0, \quad H(\mathbf{x}) = 0 \\ \mathbf{x} \geq 0, \end{aligned} \tag{1.2.12}$$

gde su $F(\mathbf{x})$, $G(\mathbf{x})$ i $H(\mathbf{x})$ vektori čije su komponente definisane sledećim funkcijama, redom:

$$f_1(\mathbf{x}), \dots, f_{k_1}(\mathbf{x}), \quad g_1(\mathbf{x}), \dots, g_{k_2}(\mathbf{x}), \quad h_1(\mathbf{x}), \dots, h_{k_3}(\mathbf{x}).$$

Ograničenje (1.2.12) se u razvijenom obliku mogu predstaviti na sledeći

način:

$$
\begin{aligned}
f_i(x_1,\dots,x_n) &\geq 0, \qquad i=1,\dots,k_1\\
g_i(x_1,\dots,x_n) &\leq 0, \qquad i=1,\dots,k_2\\
h_i(x_1,\dots,x_n) &= 0, \qquad i=1,\dots,k_3,\ \ k_1+k_2+k_3=m,\\
x_j &\geq 0, \qquad j=1,\dots,n.
\end{aligned}
\tag{1.2.13}
$$

Uobičajeno je da se jednačine u izrazima (1.2.13) nazivaju uslovima, a nejednačine ograničenjima. I jednačine i nejednačine se jednim imenom nazivaju skup ograničenja, i označavaju sa L. Indeksi m i n međusobno su nezavisni, tj. m može biti manje, jednako ili veće od n.

Funkcije $Q(\mathbf{x})$ i $f_i(x_1,\dots,x_n),\ i=1,\dots,k_1,\ g_i(x_1,\dots,x_n),\ i=1,\dots,k_2,$ $h_i(x_1,\dots,x_n),\ i=1,\dots,k_3$ u opštem slučaju su nelinearne funkcije, pa otuda naziv nelinearno programiranje. Posebni slučajevi zadataka nelinearnog programiranja javljaju se kada funkcije sadržane u Q, F, G i H nisu istovremeno nelinearne funkcije, tj. kada je samo Q nelinearna funkcija, ili je neka od funkcija iz F, G, H nelinearna. Na toj osnovi, vezujući se za oblik funkcija $Q(\mathbf{x})$, $F(\mathbf{x})$, $G(\mathbf{x})$, $H(\mathbf{x})$, vrši se formalna klasifikacija zadataka nelinearnog programiranja.

Navedene su se neke od tih klasifikacija, koje u prvi plan ističu rešive zadatke nelinearnog programiranja i njihovu opštu formulaciju [34], [73].

a) Nelinearno programiranje sa linearnim skupom ograničenja.

U ovoj klasi zadataka nelinearnog programiranja skup ograničenja zadaje se funkcijama g_i, $i=1,\dots,m$ koje su linearne, tj.

$$g_i(\mathbf{x}) = \sum_{i=1}^{n} a_{ij}x_j \leq b_i, \quad i=1,\dots,m,$$

$$x_j \geq 0, \quad j=1,\dots,n.$$

b) nelinearno programiranje sa separabilnom funkcijom cilja.

U ovoj klasi zadataka nelinearnog programiranja funkcija cilja je definisana zbirom n funkcija od kojih svaka zavisi samo od jedne promenljive, tj. $Q(\mathbf{x})$ je oblika

$$Q(\mathbf{x}) = \sum_{j=1}^{n} f_j(x_j).$$

c) Kvadratno programiranje.

Ovu klasu zadataka nelinearnog programiranja karakteriše funkcija cilja zadata u obliku kvadratne forme

$$Q(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_i x_j.$$

Specijalnu podklasu zadataka kvadratnog programiranja čine zadaci kod kojih je skup ograničenja linearan. Zadaci ove vrste, kod kojih je funkcija cilja zadata kvadratnom formom, a skup ograničenja linearnim jednačinama i (ili) nejednačinama, spadaju u grupu relativno lako rešivih zadataka.

d) Celobrojno programiranje.

Zadaci nelinearnog programiranja koji pored uslova (1.2.13) moraju zadovoljavati i posebne uslove, koji se sastoje u tome da sve promenljive mogu uzimati samo celobrojne vrednosti, čine klasu zadataka koja se naziva *celobrojno programiranje*.

Kao specijalan slučaj ove klase zadataka u praksi se često javljaju zadaci kod kojih promenljive mogu uzimati samo dve vrednosti: nula i jedan. To su zadaci koji se izučavaju u okviru posebnog naziva: 0-1 programiranje.

Do sada su, pored ostalog što je napomenuto, razvijeni efikasni algoritmi za maksimizaciju (minimizaciju) konkavne (konveksne) funkcije $Q(\mathbf{x})$ u konkavnoj (konveksnoj) oblasti (D), koja je određena skupom ograničenja $g_i(\mathbf{x}) \geq 0$, $i = 1, \ldots, m$, gde su $g_i(\mathbf{x})$ takođe konkavne (konveksne) funkcije.

Maksimizacija (minimizacija) funkcije cilja $Q(\mathbf{x})$, pri skupu ograničenja $g_i(\mathbf{x}) \leq 0$, $i = 1, \ldots, m$, pretpostavlja egzistenciju početnog plana

$$\mathbf{x}^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)}),$$

za koji je zadovoljen skup ograničenja $g_i(\mathbf{x}) \geq 0$, $i = 1, \ldots, m$.

Metodi koji su razvijeni za rešavanje ovih zadataka omogućuju nalaženje samo lokalnih ekstremuma.

Imajući u vidu ukazane napomene, potrebno je istaći da u praksi postoji nekoliko tipičnih zadataka nelinearnog programiranja, kao što su: zadaci raspodele ograničenih resursa, neki transportni zadaci, zadaci vezani za upravljanje zalihama, itd.

1.3. Metodi optimizacije i njihove osobine

Zadaci optimizacije se mogu klasifikovati na osnovu većeg broja različitih kriterijuma [60].

1. U zadacima *statičke optimizacije* objekat se posmatra u nepromenljivom jedinstvenom stanju. U zadacima *dinamičke optimizacije* ciljna funkcija zavisi od parametara koji su dati u funkciji vremena, odnosno, objekat optimizacije se smatra promenljivim u vremenu i prostoru.

Kao što je ranije napomenuto, statička optimizacija može biti *linearna* i *nelinearna* optimizacija. U ovoj knjizi se prvenstveno izučavaju metodi nelinearne optimizacije.

2. Zavisno od upravljačkih parametara, zadaci optimizacije dele se na:

– Zadaci za jednodimenzionalnu optimizaciju ($n = 1$).

– Zadaci za višedimenzionalnu optimizaciju. U slučaju $n \in \{4, 5\}$ parametara, govori se o zadacima malih dimenzija, dok se u slučaju $5 < n < 20$ radi o zadacima srednjih dimenzija. Optimizacioni problemi sa $n > 20$ upravljačkih parametara određuju zadatke velikih dimenzija.

– Zadaci sa zadatom početnom tačkom ili sa nepoznatom početnom tačkom.

3. U zavisnosti od ciljne funkcije, metodi optimizacije se dele na:

– Metodi jednokriterijumske optimizacije, u kojima se optimizira jedna ciljna funkcija.

– Metodi višekriterijumske optimizacije, u kojima se optimizira više ciljnih funkcija.

– Metodi za diferencijabilne i nediferencijabilne funkcije. Za nediferencijabilne funkcije razvijeni su metodi koji ne koriste izvode ciljne funkcije (tzv. *negradijentni metodi*). Za diferencijabilne ciljne funkcije mogu se koristiti tzv. *gradijentni metodi*, u kojima se bitno koriste parcijalni izvodi ciljne funkcije.

– Zadata ili nepoznata tačnost lokalizacije ekstremuma.

– Metodi za jednoekstremalnu i višeekstremalnu ciljnu funkciju.

– Metodi za eksperimentalno određenu ili analitički zadatu ciljnu funkciju.

4. U zavisnosti od zadatih ograničenja, optimizacioni metodi se mogu podeliti na sledeći način:

– Optimizacija bez ograničenja.

– Optimizacija sa ograničenjima koja su zadata linearnim jednakostima i (ili) nejednakostima.

– Optimizacija sa funkcionalnim ograničenjima.

Metodi optimizacije se mogu podeliti na sledeći način.

a) *Analitički metodi* se zasnivaju na analizi izvoda ciljne funkcije pomoću matematičke analize. U ovim metodima se ekstremna vrednost funkcije $Q(\mathbf{x})$ određuje pomoću izračunavanja takvih vektora $\mathbf{x}$ za koje je $Q'(\mathbf{x}) = 0$. Za velike nelinearne probleme, analitički metodi postaju nezadovoljavajući, te nisu od posebnog značaja.

b) *Numerički (iterativni) metodi* se zasnivaju na definisanju numeričkih iteracija za približnu aproksimaciju rešenja. Ovakvi metodi su najpogodniji za programiranje. Dele se u dve grupe:

– *gradijentni metodi*, koji koriste izvod ciljne funkcije;

– *negradijentni metodi*, koji ne koriste izvod ciljne funkcije.

Numerički metodi su od najvećeg značaja, i oni će biti opisani u ovoj knjizi.

c) *Grafički metodi* koriste grafičko predstavljanje ciljne funkcije i ograničenja. Ekstremum ciljne funkcije se dobija iz grafa funkcije pretraživanjem. Ovi metodi se mogu primeniti samo na ciljne funkcije od jednog ili dva upravljačka parametra. Međutim, grafički metodi se odlikuju velikom preglednošću.

d) *Eksperimentalni metodi* prognoziraju ekstremum na osnovu izvršene serije eksperimenata. Ovi metodi ne koriste matematički model procesa. Eksperimentalni metodi se koriste samo u slučaju kada se matematički model objekta pokaže neadekvatan.

Osnovne osobine koje bi algoritam optimizacije trebalo da zadovolji jesu:

– Konvergencija (dobijanje numeričkog rešenja za konačan broj koraka). Pod konvergencijom metoda se podrazumeva njegova mogućnost da generiše numeričko rešenje koje se razlikuje od početnog za ne više od zadate tačnosti ciljne funkcije i (ili) upravljačkih parametara.

– Brza konvergencija (dobijanje rešenja za što kraće vreme i sa što manjim brojem izračunavanja vrednosti ciljne funkcije).

– Što manja alokacija memorije računara.

– Ispunjenje svih ograničenja nametnutih zadatkom.

– Univerzalnost. Poželjno je da algoritam bude primenljiv na što veću klasu zadataka. Univerzalni metod za rešavanje svih tipova optimizacionih zadataka ne postoji.

Ne može se preporučiti ni jedinstveni kriterijum za prekid numeričkih metoda optimizacije. Jedan od kriterijuma koristi razliku vrednosti upravljačkog parametra u dve uzastopne iteracije:

$$\frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k)}\|} \leq \varepsilon_1.$$

Ovaj kriterijum može da prouzrokuje prevremeni prekid algoritma ako je ciljna funkcija veoma osetljiva na ekstremum. U tom slučaju je norma $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$ mali broj, dok se $Q(x^{(k+1)})$ znatno razlikuje od $Q(x^{(k)})$.

Često se koristi kriterijum koji koristi vrednosti ciljne funkcije u dve uzastopne itercije:

$$\left|\frac{Q(\mathbf{x}^{(k+1)}) - Q(\mathbf{x}^{(k)})}{Q(\mathbf{x}^{(k)})}\right| \leq \varepsilon_2.$$

Međutim, ovaj kriterijum može da prekine algoritam "daleko" od optimalne tačke u slučaju kada je ciljna funkcija slabo osetljiva na ekstremum. U tom slučaju je veličina $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$ mnogo veća od razlike $Q(\mathbf{x}^{(k+1)}) - Q(\mathbf{x}^{(k)})$

Veoma je rasprostranjen kriterijum za ocenu tačnosti lokalizacije ekstremuma koji se zasniva na zadatim minimalnim koracima $hmin_i$, $i = 1, \ldots, n$ kojima se menjaju upravljački parametri. U ovom slučaju, algoritam se prekida kada je ispunjen uslov

$$h_i \leq hmin_i, \quad i = 1, \ldots, n.$$

Ako ciljna funkcija ima veliku osetljivost na ekstremum, dobijeno rešenje može da bude nezadovoljavajuće.

Da bi se izbegao nedostatak prethodnih kriterijuma, uvodi se dvostruki kriterijum:

$$(h_i \leq hmin_i, \quad i = 1, \ldots, m) \quad \text{i} \quad \left(1 - \frac{Q^{(1)} + Q^{(2)}}{2Q^*}\right) \leq \epsilon_3, \tag{1.3.1}$$

gde su $Q^{(1)}$ i $Q^{(2)}$ dve najbliže vrednosti ciljne funkcije njenoj optimalnoj vrednosti Q^*. Kada se prvi od ova dva kriterijuma ispuni, proverava se drugi kriterijum. Ako je i drugi kriterijum ispunjen, optimizacija se prekida. Ako je prvi kriterijum ispunjen a drugi nije, proces optimizacije se nastavlja sve do ispunjenja drugog kriterijuma. Međutim, kriterijum (1.3.1) nije pouzdan u slučaju $Q^* \approx 0$. U tom slučaju se preporučuje sledeća kombinacija kriterijuma:

$$(h_i \leq hmin_i, \quad i = 1, \ldots, n) \quad \text{i} \quad \left|\frac{\Delta_{(1)} + \Delta_{(2)}}{2}\right| \leq \epsilon_4, \tag{1.3.2}$$

gde je

$$\Delta_{(1)} = Q^* - Q^{(1)}, \quad \Delta_{(2)} = Q^* - Q^{(2)}.$$

Kao jedan od kriterijuma za prekid pretraživanja može se uzeti i

$$\overline{h_i} = \frac{h_i}{xr_i^*} \leq hmin_i, \quad i = 1, \ldots, n,$$

gde je $\mathbf{xr}^* = \{xr_1^*, \ldots, xr_n^*\}$ tekuća aproksimacija optimalne tačke $\mathbf{x}^*$.

1.4. Uslovi za postojanje ekstremuma

Na početku je navedeno nekoliko osnovnih pojmova.

Označimo sa $Q : \mathbb{R}^n \mapsto \mathbb{R}$ ciljnu funkciju definisanu na prostoru Euklidovih n-torki. Vektor-gradijent $\nabla Q(\mathbf{x})$ u n-dimenzionalnom prostoru ima n komponenti, koje su jednake parcijalnim izvodima po svakom upravljačkom parametru u tački $\mathbf{x}^{(k)}$, tj.

$$\nabla Q(\mathbf{x}^{(k)}) = \text{grad} Q(\mathbf{x}^{(k)}) = \left\{ \frac{\partial Q(\mathbf{x}^{(k)})}{\partial x_i} \right\}_{i=1,\ldots,n}. \tag{1.4.1}$$

Zbog jednostavnijeg označavanja usvaja se oznaka $Q^{(k)} = Q(\mathbf{x}^{(k)})$, tako da se gradijent u tački $\mathbf{x}^{(k)}$ označava na sledeći način:

$$\nabla Q(\mathbf{x}^{(k)}) = \left\{ \frac{\partial Q^{(k)}}{\partial x_1}, \ \ldots, \frac{\partial Q^{(k)}}{\partial x_n} \right\}. \tag{1.4.1a}$$

Hesseova matrica je kvadratna matrica parcijalnih izvoda drugog reda funkcije $Q(\mathbf{x})$ u tački $\mathbf{x}^{(k)}$:

$$\nabla^2 Q(x^{(k)}) = \mathbb{H}(\mathbf{x}^{(k)}) = \begin{bmatrix} \frac{\partial^2 Q^{(k)}}{\partial {x_1}^2} & \cdots & \frac{\partial^2 Q^{(k)}}{\partial x_1 \partial x_n} \\ \vdots & & \\ \frac{\partial^2 Q^{(k)}}{\partial x_n \partial x_1} & & \frac{\partial^2 Q^{(k)}}{\partial {x_n}^2} \end{bmatrix}. \tag{1.4.2}$$

Gradijentni metodi prvog reda koriste osnovna svojstva gradijenta. Najvažnije svojstvo gradijentnog vektora $\nabla Q(\mathbf{x})$ ciljne funkcije jeste da je on u svakoj tački $\mathbf{x}^{(k)} = (x_1^{(k)}, \ldots, x_n^{(k)})$ prostora normalan na površ sa konstantnom vrednošću $Q(\mathbf{x})$ (to je linija u slučaju dva upravljačka parametra), i prolazi kroz zadatu tačku. Drugim rečima, gradijentni vektor u svakoj tački $\mathbf{x}^{(k)}$ je vektor koji ima smer najbržeg rasta funkcije $Q(\mathbf{x})$ počev od tačke $\mathbf{x}^{(k)}$. Znači, ako se korak načini u smeru gradijenta funcije $Q(\mathbf{x})$, sigurno će biti u pravcu najvećeg povećanja vrednosti funkcije $Q(\mathbf{x})$, tj. prema maksimumu, dok će obrnuti korak sigurno biti prema minimumu.

Definicija 1.4.1. Funkcija $Q(\mathbf{x})$ je konveksna ako je

$$Q\left(\lambda\mathbf{x}+(1-\lambda)\mathbf{y}\right)\leq\lambda Q(\mathbf{x})+(1-\lambda)Q(\mathbf{y})$$

za sve vektore $\mathbf{x},\ \mathbf{y}\in\mathbb{R}^n$ i za sve realne brojeve $0\leq\lambda\leq 1$. Funkcija $Q(\mathbf{x})$ je konkavna ako je funkcija $-Q(\mathbf{x})$ konveksna.

Definicija 1.4.2. Skup M je konveksan ako zajedno sa svakim parom tačaka koje pripadaju M sadrži i pravolinijski segment koji spaja te dve tačke, tj. ako važi

$$\mathbf{x}\in M,\quad \mathbf{y}\in M,\quad 0\leq\lambda\leq 1\Rightarrow\lambda\mathbf{x}+(1-\lambda)\mathbf{y}\in M$$

za sve vektore $\mathbf{x},\ \mathbf{y}\in\mathbb{R}^n$ i za sve realne brojeve $0\leq\lambda\leq 1$.

S obzirom da je za svake dve tačke $\mathbf{x},\,\mathbf{y}\in M$

$$[\mathbf{x},\mathbf{y}]=\{\lambda\mathbf{x}+(1-\lambda)\mathbf{y}:\ \lambda\in[0,1]\}$$

Ova definicija konveksnog skupa može se iskazati na drugačiji način: skup M je konveksan ako je ili prazan ili ako za svaka dva svoja elementa $\mathbf{x},\,\mathbf{y}\in M$, skup M sadrži ceo interval $[\mathbf{x},\mathbf{y}]$.

U slučaju $\mathbf{x}=\mathbf{y}$ interval $[\mathbf{x},\mathbf{y}]$ se svodi na jednoelementni skup $\{\mathbf{x}\}$. Dakle, jednoelementni skupovi su konveksni skupovi. Takođe, čitav prostor $\mathbb{R}^n$ je konveksan skup.

Propozicija 1.4.1. *Presek bilo koje kolekcije konveksnih skupova je konveksan skup. Unija konveksnih skupova ne mora da bude konveksan skup.*

Definicija 1.4.3. Ciljna funkcija je neprekidna ako je u svakoj tački njene oblasti definisanosti ispunjen uslov

$$\lim_{\Delta\mathbf{x}\to\mathbf{0}}Q(\mathbf{x}+\Delta\mathbf{x})=Q(\mathbf{x}).$$

Definicija 1.4.4. Tačka $\mathbf{x}^{(0)}$ je tačka lokalnog maksimuma (minimuma) ako postoji okolina te tačke $U(\mathbf{x}^{(0)})$, tako da za svaki vektor $\mathbf{x}\in U(\mathbf{x}^{(0)})$ važi nejednakost $Q(\mathbf{x})\leq Q(\mathbf{x}^{(0)})$ $(Q(\mathbf{x})\geq Q(\mathbf{x}^{(0)}))$.

Sledeća teorema daje potrebne uslove za ekstremnu tačku nelinearnog bezuslovnog problema

$$\text{Minimizirati}\quad Q(\mathbf{x}),\quad \mathbf{x}\in\mathbb{R}^n \tag{1.4.3}$$

Teorema 1.4.1. *Neka je* $\mathbf{x}^{(0)}$ *tačka ekstremuma funkcije* $Q(\mathbf{x})$ *i neka u tački* $\mathbf{x}^{(0)}$ *postoje parcijalni izvodi funkcije* Q*. Tada je gradijent ciljne funkcije u tački* $\mathbf{x}^{(0)}$ *jednak nuli:*

$$\nabla Q(\mathbf{x}^{(0)}) = \text{grad}Q\left(\mathbf{x}^{(0)}\right) = \left\{ \frac{\partial Q\left(\mathbf{x}^{(0)}\right)}{\partial x_i} \right\}_{i=1,\dots,n} = \mathbf{0},$$

tj.

$$\sum_{i=1}^{n} \left(\frac{\partial Q(\mathbf{x}^{(0)})}{\partial x_i} \right)^2 = 0.$$

Komentar 1.4.1. Prema Teoremi 1.4.1, sledeći uslovi su neophodni da tačka $\mathbf{x}^*$ bude tačka lokalnog maksimuma nelinearnog programa (1.4.3):

(1.4.4) $Q(\mathbf{x})$ je diferencijabilna u tački $\mathbf{x}*$.

i

$$\nabla Q(\mathbf{x}^*) = \mathbf{0}. \tag{1.4.5}$$

Dovoljni uslovi da tačka $\mathbf{x}^*$ bude optimalna za program (1.4.3) jesu uslovi (1.4.4), (1.4.5) i sledeći uslov:

$$\nabla^2 Q(\mathbf{x}^*) > \mathbf{0}, \tag{1.4.6}$$

koji označava da je Hesseova matrica u tački $\mathbf{x}^*$ pozitivno definitna.

Dovoljni uslovi da tačka $\mathbf{x}^*$ bude tačka lokalnog minimuma za problem (1.4.3) jesu uslovi (1.4.4), (1.4.5) i uslov

$$\nabla^2 Q(\mathbf{x}^*) < \mathbf{0}, \tag{1.4.7}$$

tj. Hesseova matrica u tački $\mathbf{x}^*$ je negativno definitna.

Definicija 1.4.5. Tačka $\mathbf{x}^*$ je izolovana tačka optimuma alo je, u nekoj okolini te tačke, ona jedina tačka optimuma.

Ako je $\mathbf{x}^*$ tačka lokalnog minimuma ili lokalnog maksimuma funcije $Q(\mathbf{x})$, tada je $\nabla Q(\mathbf{x}^*) = 0$. Drugim rečima, ako je $\mathbf{x}^*$ lokalni optimum funkcije Q, tada je $\mathbf{x}^*$ njena stacionarna tačka. Obrnuto tvrđenje ne važi: ako je $\mathbf{x}^*$ stacionarna tčka funkcije Q, tada $\mathbf{x}^*$ ne mora da bude lokalni minimum te funkcije [34].

Teorema 1.4.2. [34] *Neka je $Q(\mathbf{x})$ dvaput neprekidno diferencijabilna funkcija. Ako je u nekoj tački $\mathbf{x}^*$ ispunjeno $\nabla Q(\mathbf{x}^*) = 0$ i ako je Hesseova matrica $\nabla^2 Q(\mathbf{x}^*)$ pozitivno definitna, tada funkcija Q ima lokalni minimum u tački $\mathbf{x}^*$.*

Ako je $\nabla Q(\mathbf{x}^*) = 0$ i ako je $\nabla^2 Q(\mathbf{x}^*)$ negativno definitna, tada je tačka $\mathbf{x}^*$ lokalni maksimum funkcije Q.

Teorema 1.4.3. [34] *Neka je $Q(\mathbf{x})$ dvaput neprekidno diferencijabilna funkcija. Ako je $\mathbf{x}^*$ lokalni minimum funkcije $Q(\mathbf{x})$, tada je $\nabla Q(\mathbf{x}^*) = 0$ i Hesseova matrica $\nabla^2 Q(\mathbf{x}^*)$ je pozitivno semidefinitna. Ako je $\mathbf{x}^*$ lokalni maksimum funkcije $Q(\mathbf{x})$, tada je $\nabla Q(\mathbf{x}^*) = 0$ i Hesseova matrica $\nabla^2 Q(\mathbf{x}^*)$ negativno semidefinitna.*

1.5. Konveksni konus i konveksni poliedar

Sa X_n, Y_n označavaju se linearni prostori dimenzije n nad poljem realnih brojeva. Elemente linearnog prostora X_n označavamo sa $\mathbf{a}, \mathbf{b}, \mathbf{v}$.

Definicija 1.5.1. Neka su $\mathbf{v}, \mathbf{d} \in X_n$ zadati elementi. Skup $\gamma = \{\mathbf{v} + t\mathbf{d} : t \geq 0\} \subset X_n$ naziva se *poluprava* sa početkom u $\mathbf{v}$ i smerom koji je određen sa $\mathbf{d}$. Poluprava je očigledno konveksan skup. Unija neke familije polupravih sa početkom u istom elementu $\mathbf{v} \in X_n$ naziva se konus sa vrhom u $\mathbf{v}$, i označava sa $C(\mathbf{v})$. Konus u opštem slučaju nije konveksan skup.

Presek proizvoljne kolekcije konveksnih konusa sa vrhom u $\mathbf{v}$ jeste konveksni konus sa vrhom u $\mathbf{v}$.

Propozicija 1.5.1. Neka je $Y \subset X_n$ podprostor prostora X_n, i neka je Y dimenzije m. Tada je skup $\mathbf{a} + Y$, $\mathbf{a} \in X_n$ linearna mnogostrukost dimenzije m.

U optimizaciji se najčešće koriste konveksni skupovi koji nastaju kao presek poluprostora. Poluprostori i hiperravni su često korišćeni pojmovi. Kako je pojam funkcionala usko povezan sa pojmom hiperravni, to povlači da se zajedno sa prostorom X_n posmatra njegov dualni prostor linearnih funkcionala X_n^*. Elemente dualnog prostora označavamo sa $\mathbf{a}^*$, $\mathbf{b}^*$. Vrednost funkcionala $\mathbf{y}^*$ na elementu $\mathbf{x} \in X_n$ označava se sa $\mathbf{y}^*(\mathbf{x})$.

Neka je $\mathbf{H}^*$ linearni funkcional i c realni broj. Od interesa su skupovi

$$H = \{\mathbf{x} \in X_n : \ \mathbf{H}^*(\mathbf{x}) = c\}.$$

U cilju jednostavnijeg izražavanja, kaže se da je skup H definisan jednačinom $\mathbf{H}^*(\mathbf{x}) = c$.

Propozicija 1.5.2. *Neka je* $\mathbf{H}^* \neq \mathbf{0}^*$ *linearni funkcional na prostoru* X_n *i neka je* c *realan broj. Skup* H *koji je definisan jednačinom* $\mathbf{H}^*(\mathbf{x}) = c$ *jeste linearna mnogostrukost dimenzije* $n-1$.

Propozicija 1.5.3. *Neka je* H *hiperravan prostora* X_n. *Tada postoji linearni funkcional* $\mathbf{H}^*$ *na* X_n *i realan broj* c *takvi da je hiperravan* H *definisan jednačinom* $\mathbf{H}^*(\mathbf{x}) = c$.

Uz pojam hiperravni tesno je povezan pojam poluprostora. Neka je H hiperravan u X_n koja je određena funkcionalom $\mathbf{H}^*$ i realnim brojem c. Tada se mogu posmatrati skupovi

$$H^+ = \{\mathbf{x} \in X_n : \ \mathbf{H}^*(\mathbf{x}) \geq c\}, \quad H^- = \{\mathbf{x} \in X_n : \ \mathbf{H}^*(\mathbf{x}) \leq c\}.$$

Jednostavnije se kaže da su poluprostori H^+ i H^- određeni nejednačinama $\mathbf{H}^*(\mathbf{x}) \geq c$ i $\mathbf{H}^*(\mathbf{x}) \leq c$, respektivno.

Poznato je da su skupovi H^+ i H^- konveksni. Dalje, očigledno je $H = H^+ \cap H^-$ i $X_n = H^+ \cup H^-$. Kaže se da su H^+ i H^- poluprostori određeni pomoću hiperravni H.

Definicija 1.5.2. Neka su $H_1, \ldots, H_m$ hiperravni prostora X_n, i neka su $H_1^-, \ldots, H_m^-$ odgovarajući poluprostori. Neprazan skup oblika

$$K = \bigcup_{i=1}^{m} H_i^- \tag{1.5.1}$$

naziva se konveksni poliedar.

Napomenimo da konveksnost skupa K sledi iz konveksnosti poluprostora $H_1^-, \ldots, H_m^-$.

Svakoj hiperravni H_i može se pridružiti funkcional $\mathbf{H}_i^*$ i realan broj b_i, $i = 1, \ldots, m$. Izaberimo bazu $\{\mathbf{e}_j\}$, $j = 1, \ldots, n \subset X_n$ i definišimo realne brojeve $a_{ij} = \mathbf{H}_i^*(\mathbf{e}_j)$. Tada je

$$H_i^- = \left\{ \mathbf{x} = (x_1, \ldots, x_n) \in X_n : \ \sum_{i=1}^{n} a_{ij} x_j \leq b_i \right\}. \tag{1.5.2}$$

Skup nejadnačina (1.5.2) može se skraćeno napisati u matričnom obliku $A\mathbf{x} \leq \mathbf{b}$, gde je A matrica sa koeficijentima a_{ij}, a $\mathbf{b}$ vektor sa koordinatama b_i, $i = 1, \ldots, m$, $j = 1, \ldots, n$.

Definicija 1.5.3. Pretpostavimo da je konveksni poliedar K definisan jednačinom (1.5.1). Element $\mathbf{v} \in K$ naziva se vrh konveksnog poliedra K ako postoji bar n hiperravni $H_{i_1}, \ldots, H_{i_n}$ među hiperravnima $H_1, \ldots, H_m$, tako da važi

$$\mathbf{v} = \bigcap_{k=1}^{n} H_{i_k}.$$

U definiciji vrha poliedra kaže se da je on definisan presekom bar n hiperravni (a ne presekom tačno n hiperravni). U opštem slučaju, vrh se može nalaziti u preseku više od n hiperravni.

Propozicija 1.5.4. *Ako konveksni poliedar K poseduje vrh $\mathbf{v}$, tada je rang sistema (1.5.2) jednak n.*

Svaki vrh $\mathbf{v} \in K$ je određen kao jedinstveno rešenje podsistema

$$\sum_{j=1}^{n} a_{i_l} j x_j = b_{i_l}, \quad l = 1, \ldots, n$$

ranga n, i tada se vrh $\mathbf{v} \in K$ može napisati u obliku $\mathbf{v} = \sum_{j=1}^{n} x_j \mathbf{e}_j$, tj. vrh $\mathbf{v}$ ima koordinate $x_1, \ldots, x_n$.

Iz propozicije 1.5.3 se može zaključiti da konveksni poliedar ima konačno mnogo vrhova.

2. SIMBOLIČKA OPTIMIZACIJA

Ovo poglavlje je zamišljeno kao motivacija i osnova za primenu funkcionalnog stila programiranja u implementaciji optimizacionih metoda.

2.1. Globalno o simboličkoj implementaciji

U literaturi su poznati programi za implementaciju numeričkih metoda optimizacije, koji su napisani u proceduralnim programskim jezicima, najviše u FORTRANu [13], [16], [22], [30], [37], [60], [70] kao i u jeziku C [24], [36], [71]. U radovima [41], [48–53] izučavana je primena funkcionalnog programskog jezika LISP u implementaciji optimizacionih metoda. U tim radovima je pokazano da proceduralni programski jezici nisu sasvim pogodni za implementaciju optimizacionih metoda, i da korišćenje funkcionalnog stila programiranja povlači određene prednosti. U radu [52] izučavana je primena programskog paketa MATHEMATICA u implementaciji različitih varijanti

metoda kaznenih funkcija za uslovnu optimizaciju. U radu [55] uvedena je modifikacija Hooke-Jeevesovog metoda optimizacije i njegova simbolička implementacija u paketu MATHEMATICA. U radu [54] izučava se implementacija simpleks metoda linearnog programiranja u MATHEMATICA. U ovoj knjizi ova problematika će biti izučavana detaljnije, na mnogo većem broju metoda i na većem nivou apstrakcije.

S druge strane, u programskom paketu MATHEMATICA [68], [69] dostupno je nekoliko funkcija za numeričku optimizaciju. Standardna funkcija *FindMinimum* izračunava lokalni minimum zadate funkcije počev od zadate tačke, i sledeći put *najstrmijeg opadanja* (*steepest descent*) te funkcije. Funkcija *FindMinimum* se može koristiti na jedan od sledećih načina:

`FindMinimum[f, {x, x0}]` izračunati lokalni minimum funkcije f, polazeći od tačke $x = x_0$;

`FindMinimum[f, {x, x0}, {y, y0}, ...]` izračunati lokalni minimum funkcije f nekoliko promenljivih;

`FindMinimum[f, {x, {x0,x1}}]` izračunati lokalni minimum koristeći x_0 i x_1 za prve dve vrednosti promenljive x (ovaj oblik se mora koristiti ako se ne mogu izračunati simbolički izvodi funkcije f);

`FindMinimum[f,{x, xstart,xmin,xmax}]` izračunati lokalni minimum polazeći od startne tačke *xstart*, i zaustavljajući pretraživanje kada x bilo kad izađe iz oblasti $[xmin,\ xmax]$.

Zaključujemo da u funkciji *FindMinimum* ciljna funkcija može da se selektuje kao prvi parametar. Preostali parametri jesu promenljive koje su upotrebljene u analitičkom izrazu ciljne funkcije kao i njihove početne vrednosti. Prilikom implementacije optimizacionih metoda, u ovoj knjizi se koristi nešto izmenjena reprezentacija ciljne funkcije. Preciznije, prvi argument takve unutrašnje forme je analitički izraz ciljne funkcije, a drugi argument predstavlja listu promenljivih koje su iskorišćene u analitičkom izrazu ciljne funkcije. Eventualne početne vrednosti promenljivih sadržane su u trećem argumentu. Pojedini metodi optimizacije ne zahtevaju poznavanje početne tačke. U tom slučaju, treći argument nije neophodan. Zatim sledi promenljiv broj argumenata, saglasno metodu optimizacije. Saglasno prethodno napomenutom, u numeričkoj bezuslovnoj optimizaciji u MATHEMATICA, može se koristiti jedino metod najstrmijeg pada! Već je ranije napomenuto da ne postoji univerzalni metod za bezuslovnu optimizaciju, što znači da funkcija *FindMinimum* ne moža da se adekvatno upotrebi za sve slučajeve. Osim toga, poseban je problem optimizacija nediferencijabilnih funkcija. Na kraju, samo se u izuzetnim slučajevima funkcijom *Find-*

Minimum može izračunati globalni minimum ciljne funkcije, što opravdava implementaciju metoda za globalnu optimizaciju.

Za uslovnu (linearnu i globalnu) optimizaciju u programskom paketu MATHEMATICA na raspolaganju su funkcije *LinearProgramming*, *ConstrainedMin* i *ConstrainedMax* [68], [69].

Standardna funkcija *LinearProgramming* koristi se za rešavanje zadataka linearnog programiranja. U ovoj funkciji je potrebno da se jednostavno navede vektor koji predstavlja koeficijente ciljne funkcije kao i matrica koja sadrži koeficijente zadatih linearnih ograničenja.

U izrazu `LinearProgramming[c, m, b]` argumenti funkcije su vektori b i c kao i matrica m. Ovom funkcijom se izračunava vektor x koji minimizira ciljnu funkciju $c.x$ prema ograničenjima $m.x \geq b$ i $x \geq 0$. Napomenimo da izraz $a.b$ u programskom jeziku MATHEMATICA predstavlja skalarni proizvod vektora a i b.

Funkcije *ConstrainedMin* i *ConstrainedMax* dozvoljavaju da se specificira linearna ciljna funkcija koja se minimizira ili maksimizira, kao i skup linearnih ograničenja zadatih nejednakostima i (ili) jednakostima. Uvek se pretpostavlja da promenljive mogu da imaju jedino nenegativne vrednosti. Preciznije, može se reći da je u MATHEMATICA implementirano jedino linearno programiranje.

`ConstrainedMin[f, {inequalities}, {x, y, ...}]` nalaženje globalnog minimuma za f, u regionu koji je specificiran sa *inequalities*;

`ConstrainedMax[f, {inequalities}, {x, y, ...}]` nalaženje globalnog maksimuma za f, u regionu koji je specificiran sa *inequalities*.

Maksimalna preciznost ovih funkcija je 6 cifara.

Sve ove funkcije koje su ugrađene u paketu MATHEMATICA predstavljaju jedan nekompletan sistem u odnosu na veliki broj metoda optimizacije koji su poznati.

Glavna namera ove knjige je da se opiše implementacija metoda nelinearnog programiranja, koristeći mogućnosti simboličkog procesiranja u funkcionalnim programskim jezicima, pre svega MATHEMATICA, a zatim i u jeziku LISP. Detalji u vezi programskog jezika MATHEMATICA mogu se naći u [21], [68], [69] a u vezi programskog jezika LISP mogu se naći u [12], [14], [44], [61], [67].

MATHEMATICA je jedan od najuniverzalnijih programskih paketa, koji je našao primenu skoro u svim matematičkim disciplinama. Međutim, s obzirom na razvijenu numeriku i velike mogućnosti u simboličkoj obradi po-

dataka, čini se da MATHEMATICA nije u dovoljnoj meri iskorišćena u implementaciji metoda matematičkog programiranja.

Takođe, iako je LISP razvijen primarno kao alat za podršku u razvoju aplikacija u oblasti veštačke inteligencije, on je popularan i za programere koji nisu direktno vezani za oblast veštačke inteligencije. Kao dijalekat LISPa, SCHEME [47] je jedan od najsvestranijih programskih jezika dostupnih danas, koristan za različite programske projekte, kako za simboličko tako i za numeričko procesiranje.

Algoritmi u kojima se primenjuju tehnike simboličke manipulacije podacima u implementaciji optimizacionih metoda, koji su opisani u ovoj knjizi, jesu osnova za primenu proizvoljnog funkcionalnog programskog jezika ili proizvoljne tehnike funkcionalnog programiranja, u simboličkoj implementaciji optimizacionih metoda.

Ova knjiga je, koliko je autorima poznato, prvi pokušaj da se ujedine mogućnosti simboličkog i numeričkog procesiranja u implementaciji optimizacionih metoda.

Primarna namera u ovoj knjizi je da se poboljša implementacija optimizacionih metoda, koja je do sada bazirana na proceduralnim programskim jezicima. Takođe, knjiga je motivisana i odsustvom standardnih funkcija u MATHEMATICA kojima se podržavaju metodi nelinearnog programiranja. Poboljšanja su pre svega izvedena primenom mogućnosti simboličkog procesiranja u funkcionalnim programskim jezicima MATHEMATICA i PC SCHEME. Naravno, slični principi bi važili i za primenu drugih funkcionalnih programskih jezika. Međutim, ovde su preferirani programski jezici MATHEMATICA i PC SCHEME, zbog njihovih mogućnosti u simboličkom procesiranju kao i u numeričkoj obradi podataka. Glavna je namera da se pokaže kako adekvatan programski jezik za implementaciju metoda nelinearne optimizacije nije FORTRAN niti C, već jezik primenljiv podjednako i u simboličkom procesiranju kao i u numeričkim izračunavanjima.

Dosadašnja je praksa da se programi kojima se implementiraju metodi za rešavanje optimizacionih problema pišu u proceduralnim programskim jezicima, najčešće u FORTRANu. Ciljne funkcije se kod ovakvog pristupa realizuju pomoću potprograma. Potprogrami kojima se zadaju ciljne funkcije moraju biti praćeni odgovarajućim potprogramima kojima se izračunavaju njihovi parcijalni izvodi. Ovo predstavlja ograničenje i čini na taj način razvijene programe zatvorenim za nove ciljne funkcije. Primena ovako implementairanih metoda na novu ciljnu funkciju zahteva intervenciju u kodu. Izgradnja software-a, u proceduralnim programskim jezicima, kojim bi se

omogućila upotreba proizvoljne funkcije i proizvoljnih ograničenja, zahteva implementaciju relativno složenije leksičke i sintaksne analiza.

U narednim poglavljima se izučava čitav niz prednosti koje se dobijaju primenom simboličkog procesiranja u implementaciji optimizacionih metoda. To je jedan od razloga zbog kojih se nameće potreba za postojanjem fleksibilnog software-a koji bi bio namenjen ovoj uskospecijalizovanoj problematici, a ciljna grupa korisnici za koje nije neophodno dobro poznavanje programiranja. Rešenje je u postojanju programa koji ne bi zavisio od konkretnog problema koji treba rešiti, već od tipa, tj. klase kojoj problem pripada s obzirom na metodologiju njegovog rešavanja. Dakle, potrebno je da se napiše program u kome implementirani metod može da rešava sve probleme iz klase kojoj je namenjen, i to na što jednostavniji način. U ovoj knjizi učinjen je pokušaj rešavanja ove problematike kroz uvođenje funkcionalnih jezika u ovu oblast, tj. konkretno MATHEMATICAe i LISPa, kao predstavnika ove grupe programskih jezika. U svakom poglavlju, koje je posvećeno odgovarajućoj klasi problema, date su prednosti ovakvog pristupa u odnosu na proceduralne programske jezike. Takođe su dati primeri u kojima se rešavaju konkretni problemia korišćenjem razvijenog software-a.

Kao nedostatak ovakvog pristupa se može navesti interpretatorska struktura programskih jezika MATHEMATICA i LISP, što utiče na brzinu izvršenja definisanih funkcija. Takođe, jedan od nedostataka simboličke implementacije jeste veći utrošak memorije u odnosu na klasičan numerički pristup.

Treba istaći da je oblast nelinearnog programiranja veoma široka i samo delimično pokrivena od strane matematičkih programskih alata. Algoritmima i programima koji su razvijeni u ovoj knjizi, opisano stanje je u dobroj meri korigovano. Sve prednosti napisanih programa u funkcionalnim programskim jezicima u odnosu na programe dostupne u proceduralnim programskim jezicima zasnovane su na simboličkom procesiranju i prirodnoj i jednostavnoj manipulaciji funkcijama kao tipu podataka prve vrste.

2.2. PRIMENA FUNKCIONALA

U literaturi je poznat problem repetitivne primene funkcije kao argumenta u funkcionalnim programskim jezicima. U vezi ove problematike, čitalac se upućuje na rad [56]. Interesantna je analiza ovakve repetitivne primene funkcija, u slučaju kada je funkcija u ulozi argumenta upravo neka od funkcija kojom se implementiraju metodi optimizacije.

Mapping funkcije u SCHEME, kakve su *map*, *apply* i $for–each$ mogu biti primenjene na sve definisane funkcije za implementaciju metoda numeričke

optimizacije. Neka $< fc >$ označava bilo koju od funkcija kojom se implementiraju neki od metoda optimizacije, i neka $< args >$ označava listu argumenata koja je potrebna za tu funkciju. Tada se može pisati

$$(apply \ <fc> \ '(<args>)).$$

Ovakvim izrazom se funkcija fc primenjuje na svaki element iz liste argumenata $args$, a dobijeni rezultati se objedinjuju u listu.

Na primer, neka $<fc>$ označava funkciju *presek*, kojom se implementira *metod zlatnog preseka*. Tada se može pisati

```
(apply presek
'((- (* 2 (* x x x x)) (* 3 x)) (x))
'(-1.0 1.0 0.01)
)
```

Funkcija *map* može biti primenjena na funkciju $<fc>$ na sledeći način:

Korak 1. Definisati novu funkciju $<fcn>$, pomoću izraza

$$(define \ (<fcn> \ \ q) \ (<fc> \ <largs>)),$$

gde je q prvi element u $< largs >$, dok ostatak liste $< largs >$ sadrži neke fiksirane vrednosti za druge parametre optimizacione procedure fc.

Korak 2. Primeniti funkcional *map* na $<fcn>$ kao i na nekoliko unutrašnjih formi ciljne funkcije, koje su date u listi:

$$(map \ \ <fcn> \ \ '(<function> \ \ \cdots)).$$

U ovom izrazu $(<function> \cdots)$ označava listu čiji su elementi označeni sa $<function>$ i predstavljaju unutrašnje reprezentacije selektovanih funkcija.

Primer 2.2.1. Sukscesivne primene funkcije *presek* mogu se postići na sledeći način [48]:

```
(define (presekmin q) (presek q -1.0 1.0 0.01))
(map presekmin '(((expt x 2)(x))  ((log (abs y))(y)))) .
```

Rezultat ovih izraza je lista koja sadrži vrednosti dobijene posle uzastopne primene metoda zlatnog preseka na funkcije x^2 i $\ln(|x|)$, koristeći u oba slučaja $a = -1.0$, $b = 1.0$, $dmin = 0.01$.

Na isti načina se mogu koristiti element maperi iz MATHEMATICA, na primer *Map* (za jednoargumentne funkcije) i *MapThread* (za multiargumentne funkcije).

`Map[f, {a, b, ... }]` primenjuje funkciju f na svaki element u listi, dajući za rezultat $\{f[a],\ f[b], ...\}$. Funkcija *Map* se može primeniti na proizvoljne izraze, ne samo na liste, jer se liste u MATHEMATICA tretiraju kao posebni izrazi čija je glava funkcija *List*. Koristeći $head[x,\ y,\ ...]$ kao prototip izraza u MATHEMATICA, može se pisati

$$Map[f,\ \ head[x,\ y,\ ...]] = head[f[x],\ f[y],\ ..].$$

Na primer, neka je sa fc označena bilo koja optimizaciona procedura, koja za rezultat daje ekstremnu vrednost. Takođe, neka su sa x, y,... označene unutrašnje forme funkcija koje se optimiziraju. Tada izraz oblika

```
Map[fc, Plus[x,y,...]]
```

proizvodi rezultat $fc(x) + fc(y) \cdots$. Preciznije, rezultat je zbir izračunatih ekstremnih vrednosti.

Ako je rezultat funkcije fc lista koja sadrži ekstremnu tačku i ekstremnu vrednost ciljne funkcije, tada se vrednost $fc(x) + fc(y) \cdots$ može izračunati izrazom

```
Map[Plus, Map[fc, First[x,y,...]]].
```

Sledi opis funkcije *MapThread*.

`MapThread[f, {{`a_1`, `a_2`,...}, {`b_1`, `b_2`,... }, ...}]` proizvodi rezultujuću listu $\{f[a_1,\ b_1, ...],\ f[a_2,\ b_2, ...],\ ...\}$.

Izraz `Thread[f[args]]` je ekvivalentan izrazu $MapThread[f\ , args]$.æ

II GLAVA

Bezuslovna optimizacija

Problematika razmatrana u ovoj glavi se odnosi na nalaženje tačke lokalnog optimuma date ciljne funkcije pri čemu nisu zadata ograničenja. Po poglavljima su navedene implementacije najvažnijih klasa metoda za optimizaciju funkcija. U poglavlju 1.2 izučava se implementacija negradijentnih metoda optimizacije za funkcije jedne promenljive. U poglavlju 1.3 opisana je implementacija metoda višedimenzionalne negradijentne optimizacije. Predmet poglavlje 2.4 jeste implementacija metoda optimizacije koje koriste samo prve parcijalne izvode ciljne funkcije, a u poglavlju 2.5 je opisana implementacija metoda koje koriste drugi izvod diferencijabilne ciljne funkcije ili neku njegovu aproksimaciju. U poglavlju 2.6 se izučavaju metodi tzv. promenljive metrike.

Optimizacija funkcija jedne promenljive je važna koliko zbog njih samih, toliko i zbog toga što veliki broj problema optimizacije funkcija n promenljivih sadrži korake u kojima se rešava problem optimizacije sa samo jednom promenljivom. U nekim metodima višedimenzionalne optimizacije potrebna je optimizacija po jednoj od promenljivih, dok se kod nekih metoda u ulozi upravljačkog parametra javlja neka druga veličina, kao na primer dužina koraka optimizacije kod gradijentnih metoda. U problemima jednodimenzione optimizacije može biti zadata jedna polazna tačka i dozvoljena oba smera pretraživanja, ili jedna polazna tačka i smer pretraživanja, ili granice intervala u okviru koga treba naći optimalnu vrednost funkcije. Za svaku od pomenutih varijanti problema implementirano je nekoliko efikasnih metoda koji ili koriste izvode ciljne funkcije ili su formulisani tako da ne koriste izvode ciljne funkcije.

Metodi bezuslovne optimizacije funkcija koje zavise od n promenljivih koje ne koriste izvode ciljne funkcije u principu nisu brze metode, ali su našle primenu u optimizaciji nediferencijabilnih funkcija, pa čak i funkcija koje nisu neprekidne u celoj svojoj oblasti definisanosti.

Metodi bezuslovne optimizacije koji su bazirani na prvim parcijalnim izvodima ciljne funkcije nazivaju se *gradijentni metodi prvog reda*. Kod gradijentnih metoda prvog reda, polazeći od startne tačke, ciljna funkcija

se minimizira u smeru suprotnom od gradijenta ili se na osnovu gradijenata iz poslednje dve aproksimacije optimalne tačke formira smer pretraživanja.

Korišćenje drugog izvoda ciljne funkcije dovodi do brzih metoda. Newtonov metod je klasični predstavnik ovakvih metoda. U ovom metodu je za svaku aproksimaciju optimalne tačke potrebno izračunati inverznu Hesseovu matricu. Poznata je "hirovitost" ovog metoda, a jedan od načina da se ona smanji je u modifikaciji gde se uvodi jednodimenziono pretraživanje. Osim toga, implementirani su i metodi kod kojih se inverzna Hesseova matrica aproksimira odgovarajućim matricama.

1. NEGRADIJENTNI METODI

1.1. Prednosti simboličke implementacije negradijentnih metoda

Ovim metodima se izračunava ekstremna vrednost ciljne funckije bez korišćenja njenih izvoda. Drugačije se oni nazivaju *metodi direktnog pretraživanja*. Zasnivaju se na upoređivanju izračunatih vrednosti ciljne funkcije, pri čemu je za svako poboljšanje vrednosti funkcije cilja neophodno poznavanje vrednosti funkcije u prethodnom koraku (ili prethodnim koracima).

Glavne prednosti koje proizilaze iz primene simboličke obrade podataka u metodima jednodimenzionalne optimizacije su:

(1U) Ciljna funkcija, koja nije definisana potprogramom, može da bude plasirana u listu parametara bilo koje funkcije kojom se implementira neki od metoda optimizacije. U proceduralnim programskim jezicima, ciljna funkcija koja nije definisana potprogramom može se koristiti kao parametar samo uz prethodno obezbeđen, relativno složeni, proces leksičke i sintaksne analize. Ova osobina proizilazi iz mogućnosti transformacije date ciljne funkcije u pogodnu unutrašnju reprezentaciju. Ovakva unutrašnja forma se može koristiti u listi formalnih parametara.

(2U) Takođe, objektivna funkcija može da se konstruiše u procesu optimizacije iz zadatih podataka, a potom da bude primenjena. Na primer, ciljna funkcija jednog argumenta može biti konstruisana pri simboličkoj implementaciji nekog od metoda višedimenzionalne optimizacije (kako negradijentnog, tako i gradijentnog).

(3U) Za proizvoljnu jednoargumentnu funkciju $f(x)$ može se jednostavno formirati nova funkcija $F(\alpha)$, koja je definisana sa

$$F(\alpha) = Q(x) = f(x, \alpha), \qquad x, \alpha \in \mathbb{R},$$

gde je α zadati parametar. Preciznije, u ovom slučaju se zamenjuju uloge argumenta i parametra. Pri tome je bitna pretpostavka da funkcija f nije definisana potporogramom, već da je zadata kao formalni parametar, ili da je konstruisana iz zadatih podataka.

Glavne prednosti simboličke implementacije višedimenzionalnih metoda pretraživanja jesu:

(1M) Mogućnost da se ciljna funkcija zada u listi parametara, a ne potprogramom (uopštenje od **(1U)**).

(2M) Može se efikasno manipulisati sa listom argumenata ciljne funkcije.

(3M) Takođe, sledeći problem je nepogodan za implementaciju u proceduralnim programskim jezicima: Transformisati proizvoljnu funkciju od n argumenata $f(x_1, \dots, x_n)$ u novu funkciju, koja zavisi od jednog argumenta

$$F(x_k) = f\big(x_1^{(0)}, \dots x_{k-1}^{(0)}, x_k, x_{k+1}^{(0)}, \dots, x_n^{(0)}\big),$$

gde su $x_i^{(0)}$ $(i = 1, \dots, n,\ i \neq k)$ zadati realni brojevi.

Na početku su opisane prednosti **(1U)** i **(1M)**. U programima koji su napisani u proceduralnim programskim jezicima, bez leksičke i sintaksne analize, mogu da se koriste jedino ciljne funkcije koje su definisane u potprogramima (tzv. test funkcije) [2], [13], [36], [37], [60]. Takođe, u gradijentnim metodima, moraju se definisati potprogrami kojima se izračunavaju parcijalni izvodi ciljne funkcije.

Primer 1.1.1. Funkcija $f(x, y, z) = (x-1)^2 + (y-1)^2 + (z-1)^2$ zadata je pomoću potprograma [36]

```
float func(x)
float x[];
{ int i; float f=0.0;
   for(i=1; i<=3; i++) f+=(x[i]-1.0)*(x[i]-1.0);
   return f;
}
```

Parcijalni izvodi $df[1], df[2]$ i $df[3]$ ove funkcije određeni su sledećim potprogramom [36]

```
void dfunc(x,df)
float x[], df[];
{ int i;
   for(i=1; i<=3; i++) df[i]+=2.0*(x[i]-1.0);
```

}

Ako se ciljna funkcija promeni, korisnik mora da promeni sve ove potprograme. Isti principi su korišćeni i u programskom jeziku FORTRAN [13], [37].

U programskim jezicima FORTRAN ili C, moguće je koristiti ciljnu funkciju u listi formalnih parametara, jedino uz prethodnu leksičku i sintaksnu analizu unetog izraza.

S druge strane ovaj problem se jednostavno može rešiti u funkcionalnim programskim jezicima. Unutrašnju formu ciljne funkcije čine analitički izraz ciljne funkcije i lista njenih argumenata. Neka je $q\_$ formalni parametar koji predstavlja ciljnu funkciju u paketu MATHEMATICA, dok formalni parametar $var\_$ predstavlja listu argumenata funkcije Q. Ako je $x0$ lista koja predstavlja datu tačku, tada se vrednost $q0 = q[x0]$ može izračunati na sledeći način:

```
q0=q;
Do[q0=q0/.var[[i]]− >x0[[i]], {i,Length[var]}];
```

U jednodimenzionalnom slučaju, za zadati realan broj $x0$ može se pisati:

```
q0=q;     q0=q0/.var[[1]]− >x0;
```

U funkcijama kojima se implementiraju optimizacioni metodi u LISPu, realna ciljna funkcija $Q(x_1, \ldots x_n) = Q(\mathbf{x})$ je predstavljena dvoelementnom listom oblika

```
'( Q(x)  (x) )
```

Prvi element te liste jeste selektovana LISPovska aritmetička funkcija, dok je drugi argument lista njenih argumenata.

Ovakva unutrašnja reprezentacija, označena sa q, date ciljne funkcije, može da se transformiše u odgovarajući *lambda-izraz*:

```
(set!  fun (eval (list 'lambda (cadr q) (car q)))).
```

Ovakva lambda funkcija se može primeniti na zadatu listu argumenata v:

```
(apply fun v)
```

Primer 1.1.2. Unutrašnja forma funkcije $f(x) = 2x^3 - \dfrac{\log(x)}{4a}$ u LISPu je sledeća lista:

```
((- (* 2 (expt x 3)) (/ (log x) (* 4 a)))  (x)) .
```

Odgovarajuća unutrašnja forma u paketu MATHEMATICA je:

```
2*x^3-Log[x]/(4*a),{x} .
```

U ovom primeru se podrazumeva da je a zadati parametar. Međutim, funkcija f se može definisati i kao funkcija parametra a, pri čemu promenljiva x preuzima ulogu parametra. To se može jednostavno učiniti promenom liste argumenata funkcije:

```
((- (* 2 (expt x 3)) (/ (log x) (* 4 a)))  (a))
```

ili

```
2*x^3-Log[x]/(4*a),{a} .
```

Ovim je opisana prednost **(3U)**. Prednost **(2U)** će biti korišćena kasnije kod implementacije negradijentnih metoda višedimenzionalne optimizacije (u vezi **(3M)**) ili kod implementacije gradijentnih metoda optimizacije (u vezi sa **(3G)**). Prednost **(3G)** simboličke implementacije biće opisana kasnije.

1.2. Jednodimenzionalna negradijentna optimizacija

Zadata je ciljna funkcija $Q = Q(x)$ koja zavisi od jednog parametra x. Problem je naći lokalni maksimum (minimum) te funkcije, uz uslov nametnut upravljačkom parametru: $a \leq x \leq b$. Potrebno je naći takvu vrednost upravljačkog parametra x^* za koju ciljna funkcija ima optimalnu vrednost unutar dozvoljene oblasti:

$$Q(x^*) = Q_{\max} > Q(x), \quad a \leq x \leq b, \quad a \leq x^* \leq b.$$

U matematičkoj analizi, poznato je da se optimalna vrednost x^* može naći kao rešenje jednačine $Q'(x) = 0$. Međutim, za složene matematičke modele i složene transcendentne zavisnosti, analitičko rešenje ove jednačine je često puta teško naći, ili je pak to nemoguće, pa je stoga neophodno koristiti neke numeričke metode kojima se ono približno određuje. Osim toga, može se dogoditi da rešenje jednačine $Q'(x) = 0$ ne bude tačka ekstremuma. Sve to opravdava korišćenje različitih metoda optimizacije, u kojima se ekstremna vrednost ne određuje kao rešenje jednačine $Q'(x) = 0$.

U ovoj glavi je opisana implementacija nekoliko metoda jednodimenzionalne optimizacije u programskim jezicima MATHEMATICA i LISP. O metodima jednodimenzionalne optimizacije može se, na primer, naći u [2], [13], [15], [34], [35], [60], [73]. Ovi metodi su korisni zato što se u mnogim metodima za

rešavanje problema optimizacije sa n promenljivih, javljaju koraci u kojima treba rešiti problem optimizacije sa samo jednom promenljivom. Ovde su prvo obrađeni metodi kod kojih je poznat interval u kojem se nalazi optimalno rešenje, a koji upoređuju vrednosti funkcije i na taj način ocenjuju interval u kojem se nalazi optimalno rešenje. Što se više vrednosti upoređuje, to će interval biti manji i tačnost lokalizacije ekstremuma bolja. Zanimljivo je da kod ovih metoda dužina intervala zavisi samo od broja upoređivanja, a ne i od same funkcije. Metodi upoređivanja vrednosti funkcije ne zahtevaju da funkcija bude diferencijabilna; štaviše, funkcija može biti i prekidna. Važno je da funkcija bude unimodalna. Kažemo da je funkcija (jedne promenljive) f *unimodalna* na intervalu I ako f ima minimum u nekoj tački x^* intervala I i ako za svake dve tačke x_1 i x_2 intervala I, koje ispunjavaju uslov $x_1 < x_2$ važi:

$$x_1 < x_2 \leq x^* \quad \text{povlači} \quad f(x_1) > f(x_2),$$

$$x^* \leq x_1 < x_2 \quad \text{povlači} \quad f(x_2) > f(x_1).$$

U daljem tekstu, obrađeni su takvi metodi jednodimenzionalne optimizacije kod kojih je poznata polazna tačka, na osnovu koje se određuje interval I koji sadrži tačku optimuma.

Metodi jednodimenzionalne negradijentne optimizacije mogu biti podeljeni u tri grupe.

A. *Različiti metodi skaniranja.* Metodi skaniranja se zasnivaju na ispitivanju ciljne funkcije u različitim tačkama oblasti $[a, b]$, sve dok se ne dostigne dovoljno mali interval $\Delta_{\min}$ u kome je lokalizovan ekstremum. U ovu grupu metoda spadaju:

1. skeniranje sa konstantnim korakom;
2. skeniranje sa promenljivim korakom;
3. Jednodimezioni simpleks metod;
4. Metod dihotomije;
5. Metod zlatnog preseka

B. *Interpolacioni metodi.* U slučaju kada početni interval $[a, b]$ nije zadat, ekstremum može da se pretražuje interpolacionim i ekstrapolacionim metodima. Osnovna ideja ovih metoda jeste definisanje polinomne aproksimacije drugog ili većeg stepena tačnosti kojom se aproksimira ciljna funkcija. Ova aproksimacija se formira na osnovu nekoliko izračunatih vrednosti ciljne funkcije, kao i na osnovu tekućeg maksimuma $x_{\max}^{(k)}$ aproksimativnog polinoma.

Ako su za ciljnu funkciju $Q(x)$ izračunate vrednosti Q_a, Q_m, Q_b, koje odgovaraju parametrima $x^{(a)}$, $x^{(m)}$, $x^{(b)}$, respektivno, funkcija $Q(x)$ može se aproksimirati algebarskim polinomom drugog stepena

$$\hat{Q}(x) = b_0 + b_1(x - x^{(a)}) + b_{11}(x - x^{(a)})(x - x^{(m)}).$$

Koeficijenti b_0, b_1 i b_{11} se izračunavaju po formulama

$$b_0 = Q_a, \qquad b_1 = \frac{Q_m - Q_a}{x^{(m)} - x^{(a)}}, \qquad b_{11} = \frac{D}{x^{(b)} - x^{(m)}},$$

gde je

$$D = \frac{Q_b - Q_a}{x^{(b)} - x^{(a)}} - \frac{Q_m - Q_a}{x^{(m)} - x^{(a)}}.$$

Ekstremne vrednosti ciljne funkcije $Q(x)$ se procenjuju pomoću $\hat{Q}'(x) = 0$, odakle sleduje

$$x^* \approx x_{\max}^{(k)} = \frac{1}{2}(x^{(a)} + x^{(m)}) - \frac{b_1}{2b_{11}}.$$

Interpolacioni metodi, kako za jednodimenzionalo, tako i za višedimenzionalno pretraživanje ekstremuma razlikuju se prema načinu generisanja tačaka koje su neophodne za polinomnu aproksimaciju. U ovu grupu metoda spadaju:

1. Metod Davies-Swann-Campey (DSC);
2. Powelov jednodimenzionalni metod.

C. *Metodi aproksimacije polinomom.* Ovi metodi se sastoje u sledećem: Da bi se odredila tačka optimuma x^* funkcije Q, ona se najpre aproksimira polinomom $p(x)$ na intervalu $[a, b]$ koji sadrži tačku x^*. Zatim se određuje tačka optimuma xm polinoma $p(x)$. Za približnu vrednost tačke x^* može se uzeti tačka xm. Sada se interval $[a, b]$ smanjuje, funkcija Q se aproksimira novim polinomom i postupak se nastavlja sve dok se ne dostigne željena tačnost. Za aproksimativne polinome obično se uzimaju polinomi drugog i trećeg reda. U tom smislu se može govoriti o *metodu parabole* ili tzv. *kubnom metodu*.

Iz ove grupe metoda izučavaćemo samo metod parabole.

1.2.1. SKENIRANJE SA KONSTANTNIM KORAKOM

U ovom metodu uzastopno se ispituje ciljna funkcija, počev od neke vrednosti a do vrednosti b upravljačkog parametra, sa fiksiranim korakom, koji

se naziva korak skaniranja i označava sa Δ. Od dobijenih rezultata uzima se najbolji. Na taj način je ektremna vrednost lokalizovana sa tačnošću $\Delta_{\min} = \pm\Delta$. Tačnost lokalizacije ektrema je veća ako se smanjuje korak Δ.

Algoritam metoda skaniranja sa konstantnim korakom može se opisati na sledeći način:

Korak 1. Uneti vrednosti za granice optimizacije a, b i korak skaniranja Δ.

Korak 2. Staviti $Q_{\max} = Q(a)$, u slučaju maksimuma, odnosno $Q_{\min} = Q(a)$, u slučaju minimuma.

Korak 3. Staviti $X_m = a$, $X = a$.

Korak 4. $X = X + \Delta$.

Korak 5. Ako je $X > b$ za izlaz iz algoritma uzeti $Q_{\max}$ (odnosno $Q_{\min}$) i vrednost parametra X_m za koji je optimalna vrednost dostignuta; inače, preći na sledeći korak.

Korak 6. Izračunati $Q_1 = Q(X)$. Ako je $Q_1 > Q_{\max}$, u slučaju maksimuma staviti $Q_{\max} = Q_1$, $X_m = X$. U slučaju minimuma, kada je ispunjen uslov $Q_1 < Q_{\min}$, staviti $Q_{\min} = Q_1$, $X_m = X$. Zatim preći na *Korak 4*.

Pogodnosti ovog metoda su: (a) laka algoritmizacija; (b) sa malim korakom Δ može se pronaći globalni ekstrem.

Nedostatak metoda je veliki broj izračunavanja vrednosti ciljne funkcije.

Metod se može implementirati koristeći jedino prednost **(1U)** simboličkog pristupa.

```
skk[q_,pr_,a_,b_,del_]:=
   Block[{xm=x=a, q0, qm, izb, Lista={} },
     izb = Input["1 za minimum, 2 za maksimum:"];
     q0=q; q0=q/.pr[[1]]->x; qm=N[q0];
     Lista=Append[Lista,a,q0];
     x=x+del;
     While[N[x]<=b,
        q0=q/.pr[[1]]->x;
        If[(izb==1 && N[q0]<N[qm])||
           (izb==2 && N[q0]>N[qm]),
              xm=x; qm=N[q0]; Lista=Append[Lista,{xm,qm}];
        ];
        x=x+del
     ];
     {xm,qm, Lista}
```

```
]
```

Program se poziva sa $skk[q, pr, a, b, del]$, pri čemu je:

q: ciljna funkcija;

pr: lista koja sadrži parametar ciljne funkcije;

a, b: granice oblasti skaniranja;

del: korak skaniranja.

Argumenti q i pr predstavljaju tzv. *unutrašnju formu* ciljne funkcije.

Rezultati testiranja programa.

```
In[1]:=skk[N[Sin[x+3]],{x},-2,2,0.1]    (*minimum*)
Out[1]={1.7, -0.999923, {{-0.8, 0.808496}, {-0.7, 0.745705}, {-0.6, 0.675463},
   {-0.5, 0.598472}, {-0.4, 0.515501}, {-0.3, 0.42738}, {-0.2, 0.334988},
   {-0.1, 0.239249}, {6.38378 10^-16 , 0.14112}, {0.1, 0.0415807}, {0.2, -0.0583741},
   {0.3, -0.157746}, {0.4, -0.255541}, {0.5, -0.350783}, {0.6, -0.44252},
   {0.7, -0.529836}, {0.8, -0.611858}, {0.9, -0.687766}, {1., -0.756802},
   {1.1, -0.818277}, {1.2, -0.871576}, {1.3, -0.916166}, {1.4, -0.951602},
   {1.5, -0.97753}, {1.6, -0.993691}, {1.7, -0.999923}}}
In[2]:= skk[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x},-1,1,0.2]    (*maksimum*)
Out[2]={-1., 0.504916, {{-1., 0.504916}}}
In[3]:= skk[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x},-1,1,0.2]    (*minimum*)
Out[3]={-0.2, -0.888221, {{-1., 0.504916},{-0.8, -0.115198}, {-0.6, -0.544206},
   {-0.4, -0.796135},{-0.2, -0.888221}}}
```

Implementacija u LISPu može se opisati na sledeći način. Ulazni parametri imaju ranije definisani smisao. Rezultat je lista koja sadrži ekstremnu tačku mx u intervalu $[a,\ b]$ i odgovarajuću ekstremnu vrednost mf.

```
(define (fixst q a b del izb)
  (let ((mf 1) (f 1) (vr 1) (x 1) (mx 1))
     ; Transformisati formu q u lambda-izraz f
     (set!  f (eval (list 'lambda (cadr q) (car q))))
     ; Naći tačku ekstrema i ekstremnu vrednost
     (set!  mf (apply f a)) (set!  mx a)
     (do ((x (+ a del) (+ x del)))
          ((> x b) (newline) (list mx mf))
          (set!  vr (apply f x))
          (cond ( (or (and (< vr mf) (equal izb 'm))
                      (and (> vr mf) (equal izb 'x)))
                  (set!  mf vr) (set!  mx x)
) )  )     )     )
```

1.2.2. SKENIRANJE SA PROMENLJIVIM KORAKOM

Ovaj algoritam u izvesnoj meri otklanja nedostatak skaniranja sa konstantnim korakom. Za prvo skeniranje se uzima relativno veliki konstantni korak $\Delta^{(1)}$ i grubo se lokalizuje tačka ekstremuma $x_m^{(1)}$. Zatim se izdvaja podoblast $x_m^{(1)} \pm \Delta^{(1)}$ i u njoj se izvrši skeniranje sa manjim korakom $\Delta^{(2)}$. Korak se smanjuje, sve dok se ne postigne zadata tačnost $\Delta_{\min}$, što se postiže ispunjenjem uslova $\Delta^{(k)} < \Delta_{\min}$ za neko $k > 1$.

Prednost ovog algoritma, u odnosu na konstantno skeniranje, jeste manji broj izračunavanja vrednosti cijlne funkcije, dok je njegov nedostatak veća verovatnoća da se u slučaju ciljne funkcije sa velikim brojem ekstremuma propusti globalni ekstremum zbog relativno velikog početnog koraka skaniranja.

```
spk[q_,pr_,a_,b_,del_,delmin_]:=
   Block[{delta=del, x,xm, q0,qm, dg=a,gg=b,izb, Lista={} },
       izb = Input["1 za mimimum, 2 za maksimum:"];
       While[N[delta]>=N[delmin],
           xm=x=N[dg]; qm=q/.pr[[1]]->dg;
           Lista=Append[Lista,{xm,qm}];
           x = N[x+delta];
           While[x<=N[gg],
                q0=q/.pr[[1]]->x;
                If[(izb==1&&N[q0]<N[qm]) ||
                   (izb==2&&N[q0]>N[qm]),
                    qm=N[q0]; xm=N[x];
                    Lista=Append[Lista,{xm,qm}]
                ];
                x=N[x+delta];
        ];
        dg=If[xm-delta < a, a, N[xm - delta]];
        gg=If[xm+delta > b, b, N[xm + delta]];
        delta=N[delta/4];
       ];
       {xm,qm, Lista}
   ]
```

Formalni parametri programa $spk[q, pr, a, b, del, delmin]$ imaju sledeći smisao:

q: ciljna funkcija;

pr: lista koja sadrži argument funkcije cilja;

a, b: granice oblasti skaniranja;

del: početni korak skaniranja;

delmin: minimalna vrednost koraka skaniranja, tj. zahtevana tačnost.

Lokalna promenljiva *izb* određuje izračunavanje minimuma ili maksimuma.

Rezultati testiranja programa.

```
In[1]:=spk[x^2-5x+8,{x},-4,4,0.1,0.02]    (*minimum*)
Out[1]={2.5, 1.75, {{-4., 44.}, {-3.9, 42.71}, {-3.8, 41.44}, {-3.7, 40.19}, {-3.6, 38.96},
   {-3.5, 37.75}, {-3.4, 36.56}, {-3.3, 35.39}, {-3.2, 34.24}, {-3.1, 33.11},
   {-3., 32.}, {-2.9, 30.91}, {-2.8, 29.84}, {-2.7, 28.79}, {-2.6, 27.76},
   {-2.5, 26.75}, {-2.4, 25.76}, {-2.3, 24.79}, {-2.2, 23.84}, {-2.1, 22.91},
   {-2., 22.}, {-1.9, 21.11}, {-1.8, 20.24}, {-1.7, 19.39}, {-1.6, 18.56},
   {-1.5, 17.75}, {-1.4, 16.96}, {-1.3, 16.19}, {-1.2, 15.44}, {-1.1, 14.71},
   {-1., 14.}, {-0.9, 13.31}, {-0.8, 12.64}, {-0.7, 11.99}, {-0.6, 11.36},
   {-0.5, 10.75}, {-0.4, 10.16}, {-0.3, 9.59}, {-0.2, 9.04}, {-0.1, 8.51},
   {2.41474 10^-15, 8.}, {0.1, 7.51}, {0.2, 7.04}, {0.3, 6.59}, {0.4, 6.16},
   {0.5, 5.75}, {0.6, 5.36}, {0.7, 4.99}, {0.8, 4.64}, {0.9, 4.31}, {1., 4.},
   {1.1, 3.71}, {1.2, 3.44}, {1.3, 3.19}, {1.4, 2.96}, {1.5, 2.75}, {1.6, 2.56},
   {1.7, 2.39}, {1.8, 2.24}, {1.9, 2.11}, {2., 2.}, {2.1, 1.91}, {2.2, 1.84},
   {2.3, 1.79}, {2.4, 1.76}, {2.5, 1.75}, {2.4, 1.76}, {2.425, 1.75562},
   {2.45, 1.7525}, {2.475, 1.75062}, {2.5, 1.75}}}
In[2]:= spk[N[sin [x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x},-4,4,0.1,0.02]    (*minimum*)
Out[2]= {-0.175, -0.889493,
   {{-4., 36.736}, {-3.9, 34.6512}, {-3.8, 32.6326}, {-3.7, 30.6791},
   {-3.6, 28.7898}, {-3.5, 26.9637}, {-3.4, 25.2001}, {-3.3, 23.4982},
   {-3.2, 21.8573}, {-3.1, 20.277}, {-3., 18.7568}, {-2.9, 17.2962}, {-2.8, 15.8948},
   {-2.7, 14.5524}, {-2.6, 13.2687}, {-2.5, 12.0435}, {-2.4, 10.8765},
   {-2.3, 9.76751}, {-2.2, 8.71643}, {-2.1, 7.72303}, {-2., 6.78708}, {-1.9, 5.90836},
   {-1.8, 5.08657}, {-1.7, 4.32137}, {-1.6, 3.61237}, {-1.5, 2.95909},
   {-1.4, 2.36096}, {-1.3, 1.81731}, {-1.2, 1.32737}, {-1.1, 0.890247},
   {-1., 0.504916}, {-0.9, 0.170208}, {-0.8, -0.115198}, {-0.7, -0.352783},
   {-0.6, -0.544206}, {-0.5, -0.691309}, {-0.4, -0.796135}, {-0.3, -0.860942},
   {-0.2, -0.888221}, {-0.3, -0.860942}, {-0.275, -0.871178}, {-0.25, -0.879107},
   {-0.225, -0.884774}, {-0.2, -0.888221}, {-0.175, -0.889493}}}
```

Odgovarajuća funkcija u LISPu je:

```
(define (varst q a b delta dmin izb)
  (let ((mx 1) (l 1) (lg 1))
     ;Korak 1. Prekinuti kada je korak manji od dmin
     (do ()
        ((< delta dmin) (newline) l)
        ;Korak 2. Koristeći uniformno pretraživanje
        ; naći listu l, koja sadrži
```

```
            ; ekstremnu tačku i ekstremnu vrednost
            (set!  l (fixst q a b delta izb))
            ;Korak 3.  Definisati novo uniformno pretraživanje
            (set!  lg (/ (abs (- b a)) delta))
            (set!  mx (car l))
            (set!  a (- mx delta)) (set!  b (+ mx delta))
            (set!  delta (/ (abs (- b a)) lg)
    ) ) )
```

1.2.3. JEDNODIMENZIONALNI SIMPLEKS METOD

U nekim zadacima optimizacije može se dogoditi da se izostavi donja i/ili gornja granica optimizacije. U takvim slučajevima preporučuje se jednodimenzionalni simpleks metod, koji se drugačije naziva skeniranje sa promenljivim povratnim korakom. Takođe, ovaj metod može se koristi i u slučaju zadatih granica upravljačkog parametra $a \leq x \leq b$, kao i za nalaženje tačke ekstremuma ciljne funkcije koja zavisi od više upravljačkih parametara $Q(\mathbf{x})$, $\mathbf{x} = (x_1, \ldots, x_n)$, prema zadatom pravcu (slučajno, gradijentno i dr.). Postoji nekoliko varijanti simpleks metoda.

Varijanta I. Pretraživanje ekstremuma počinje sa zadatom veličinom početnog koraka, koja se neprekidno smanjuje, ali može i da menja smer. Ova varijanta se koristi kada je zadata jedna od granica a ili b, ili obe granice upravljačkog parametra. Ovde je opisana varijanta u kojoj se koriste i donja i gornja granica upravljačkog parametra. skeniranje započinje sa relativno velikim korakom $\Delta^{(0)}$, a nastavlja se sve dok se ne dobije neuspešan rezultat za ciljnu funkciju. U tom slučaju se menja smer skaniranja i uzima se korak manje dužine $\Delta^{(1)}$. Kada se ponovo dobije neuspešan rezultat menja se smer i nastavlja skeniranje sa manjim korakom $\Delta^{(2)}$. Iterativni proces se nastavlja sve dok se ne dostigne željena tačnost $\Delta^{(k)} \leq \Delta_{\min}$, za neko $k \geq 1$. Preporučuje se da svaki sledeći korak bude četiri puta manji od prethodnog, tj. $\Delta^{(k+1)} = -\Delta^{(k)}/4$. Ovakav izbor koraka obezdeđuje bržu konvergenciju.

Opisani postupak prikazan je na slici 1.2.1.

Ulazne veličine:

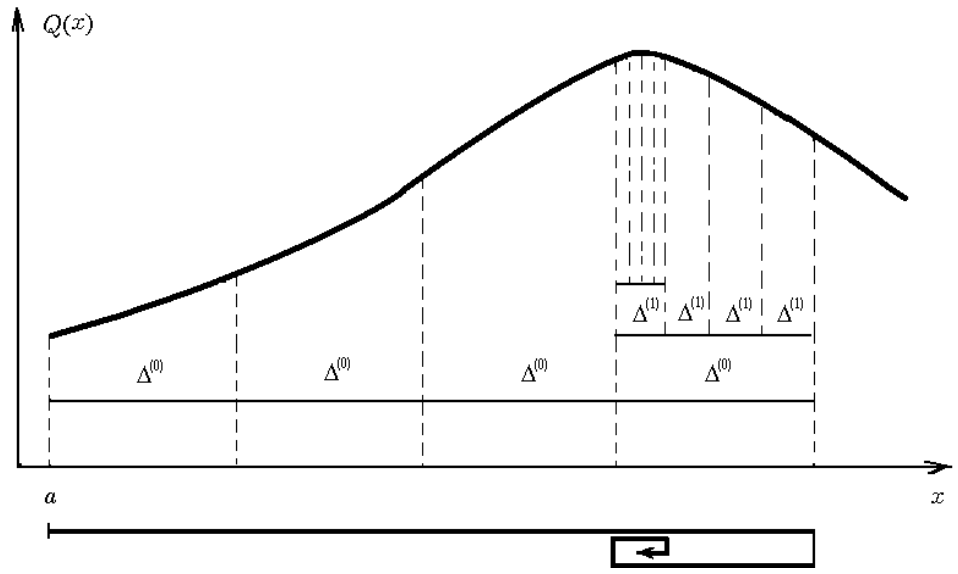
$q\_$, $pr\_$: ciljna funkcija i lista sa njenim parametrom;

$a\_$, $b\_$: donja i gornja granica skaniranja.

$korak\_$: početni korak skaniranja;

$minkor\_$: minimalni korak skaniranja.

Lokalne promenljive:



Sl. 1.2.1

xt, $q1$: tekuća tačka i vrednost ciljne funkcije;

xm, qm: tekuća optimalna tačka i optimalna vrednost;

izb: parametar koji određuje lokalizaciju minimuma ($izb = 1$), odnosno maksimuma ($izb = 2$).

Algoritam prve varijante simpleks metoda:

Korak 1. Početna vrednost tačke optimuma i početna ekstremna vrednost: $x = xm = a$, $qm = q(a)$;

Korak 2. $x = x + del$;

Korak 3. `While` ciklus, koji se prekida kada je ispunjen uslov $|del| < dmin$. Unutar ciklusa treba izvršiti sledeće korake:

Korak 3.1. Ako je $x > b$ uzeti $x = xm = b$, $qm = q(b)$, $del = -del/4$.
Ako je $x < a$ uzeti $x = xm = a$, $qm = q(a)$, $del = -del/4$.

Korak 3.2. Izračunati $q1 = q(x)$.
Ako je $izb = 2$, $q1 > qm$ ili $izb = 1$, $q1 < qm$ postaviti $qm = q1$, $xm = x$.

Korak 3.3. Povećati x za tekući korak: $x = x + del$.

Korak 4. Izlazne veličine su xm i qm.

```
simplexI[q_,pr_List,a_,b_,korak_,minkor_]:=
   Block[{dg=a,gg=b, del=korak,dmin=minkor, xm=xt=a,
          q1,qa,qb,qm,it=0, izb, Lista={} },
       qa=N[q/.pr[[1]]->dg]; qb=N[q/pr[[1]]->gg];
       qm=qa; Lista=Append[Lista,{a,qa}];
       xt=N[xt+del];
       izb=Input["1 za minimum, 2 za maksimum "];
       While[N[Abs[del]]>=dmin && it<100,
           If[xt>b, xm=b; xt=b; qm=qb; del=N[-del/4] ];
           If[xt<a, xm=a; xt=a; qm=qa; del=N[-del/4] ];
           If[xt>=a && xt<=b,
             q1=N[q/.pr[[1]]->xt];
             If[(izb==1&&q1<qm)||(izb==2&&q1>qm),
                  xm=N[xt];qm=N[q1];
                  Lista=Append[Lista,{xm,qm}],
                  del=N[-del/4]
             ];
           ];
           it+=1;
           xt=N[xt+del]
       ];
       {N[xm], N[qm], Lista}
    ]
```

Odgovarajuća funkcija u LISPu je:

```
(define (sim q a b delta dmin izb)
  (let ((x 1) (xm 1) (qe 1) (fc 1) (qa 1) (qb 1) (q1 1))
  ; Korak 1.   Izracunati pocetne vrednosti
  (set!  fc (eval (list 'lambda (cadr q) (car q))))
  (set!  x a) (set!  xm a)
  (set!  qa (apply fc a)) (set!  qe qa)
  (set!  qb (apply fc b))
  (do ()
     ((< (abs delta) dmin) (print dmin) (list xm qe))
      (set!  x (+ x delta))
      (cond ( (> x b)
          (set!  xm b) (set!  x b) (set!  qe qb)
          (set!  delta (- (/ delta 4.0)))
   )
```

```
      ( (< x a)
         (set!  xm a) (set!  x a) (set!  qe qa)
         (set!  delta (- (/ delta 4.0)))
      )
      ( t
         (set!  q1 (apply fc x))
         (cond ( (or (and (< q1 qe) (equal?  izb 'm))
                     (and (> q1 qe) (equal?  izb 'x))
                 )
                 (set!  xm x) (set!  qe q1)
               )
               (t (set!  qe q1) (set!  delta (- (/ delta 4.0)))
   ) ) ) ) ) ) ) )
```

Varijanta II. Pretraživanje kod ove varijante započinje od izabrane početne tačke x_0, sa zadatom veličinom početnog koraka (razmer simpleksa), koji se uvećava sve dok se ne dođe u oblast ekstremne tačke. Tada korak počinje da se smanjuje.

Ova varijanta se preporučuje kada oblast pretraživanja nije definisana, tj. u slučaju $-\infty < x < +\infty$.

Ulazne veličine:

$q\_$, $pr\_$: ciljna funkcija i lista koja sadrži parametar ciljne funkcije;

$pocx\_$: početna tačka;

$del = korak\_$, $dmin = minkor\_$: početna i minimalna vrednost koraka.

Lokalne promenljive:

in: indikator kojim se signalizira da je potrebna promena smera još u početnoj tački;

id: indikator koji označava da je bar jednom dostignuta uspešna vrednost i da se korak ne može povećavati;

x, $q1$: tekuća tačka i vrednost ciljne funkcije;

xm, qm: optimalna tačka i optimalna vrednost;

izb: parametar koji određuje lokalizaciju minimuma ($izb = 1$), odnosno maksimuma ($izb = 2$).

Glavni koraci u algoritmu:

Korak 1. Početne vrednosti: $in = 0$, $id = 0$, $x = xm = x0$, $qm = q(x0)$;

Korak 2. $x = x + del$;

Korak 3. `While` ciklus, koji se prekida kada je ispunjen uslov $|del| < dmin$. Unutar ciklusa izvršiti sledeće korake:

Korak 3.1. Izračunati $q1 = q(x)$.
Ako je $izb = 1$, $q1 < qm$ ili $izb = 2$, $q1 > qm$, preći na *Korak 3.2*, inače preći na *Korak 3.3*.

Korak 3.2. Postaviti $xm = x$, $qm = q1$, $in = 1$.
Ako je $id = 0$ staviti $del = 2 * del$, a zatim preći na *Korak 3.4*.

Korak 3.3. Ako je $in = 0$ postaviti $x = x0$, $del = -del$, $in = 1$, inače postaviti $id = 1$, $del = -del/4$; $qm = q1$.
Preći na *Korak 3.4*.

Korak 3.4. Povećati x: $x = x + del$.

Korak 4. Izlazne veličine su xm i qm.

```
simplexII[q_,pr_,pocx_,korak_,minkor_]:=
  Block[{x0=N[pocx],del=korak,dmin=minkor,in=id=it=0,qm,
                x=xm=N[pocx],izb, Lista ={} },
      izb=Input["Zelite li minimum(1) ili maksimum(2)?"];
      qm=N[q/.pr[[1]]->pocx];
      Lista=Append[Lista,{x0,qm}];
      x=x+del;
      While[Abs[del]>=dmin && it<100,
         q1=N[q/.pr[[1]]->x];
         If[(izb==1 && q1<qm) || (izb==2 && q1>qm),
            xm=x;qm=q1;in=1; If[id==0,del=del*2];
            Lista=Append[Lista,{xm,qm}],
            If[in==0,
               x=x0; del=-del; in=1,
               id=1; del=-del/4
            ]
         ];
         x=x+del;  it=it+1;
      ];
      If[id==0, Print["Funkcija nema ekstremum "] ];
      Return[{xm,qm, Lista}]
  ]
```

Rezultati testiranja programa.

In[1]:= simplexI[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x},-4,4,0.1,0.02]

Out[1]={-0.2, -0.888221, {{-4, 36.736}, {-3.9, 34.6512}, {-3.8, 32.6326}, {-3.7, 30.6791},
{-3.6, 28.7898}, {-3.5, 26.9637}, {-3.4, 25.2001}, {-3.3, 23.4982},
{-3.2, 21.8573}, {-3.1, 20.277}, {-3., 18.7568}, {-2.9, 17.2962}, {-2.8, 15.8948},
{-2.7, 14.5524}, {-2.6, 13.2687}, {-2.5, 12.0435}, {-2.4, 10.8765},
{-2.3, 9.76751}, {-2.2, 8.71643}, {-2.1, 7.72303}, {-2., 6.78708}, {-1.9, 5.90836},
{-1.8, 5.08657}, {-1.7, 4.32137}, {-1.6, 3.61237}, {-1.5, 2.95909},
{-1.4, 2.36096}, {-1.3, 1.81731}, {-1.2, 1.32737}, {-1.1, 0.890247},
{-1., 0.504916}, {-0.9, 0.170208}, {-0.8, -0.115198}, {-0.7, -0.352783},
{-0.6, -0.544206}, {-0.5, -0.691309}, {-0.4, -0.796135}, {-0.3, -0.860942},
{-0.2, -0.888221}}}

In[2]:=simplexI[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x},-4,4,0.1,0.02] (*maksimum*)

Out[2]={-4., 36.736, {{-4, 36.736}}}

In[3]:= simplexII[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]], {x},-4,0.1,0.02] (*minimum*)

Out[3]={-0.9, 0.170208, {{-4., 36.736}, {-3.9, 34.6512}, {-3.7, 30.6791},
{-3.3, 23.4982}, {-2.5, 12.0435}, {-0.9, 0.170208}}}

In[4]:=simplexII[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]], {x},-4,0.1,0.02] (*maksimum*)

Funkcija nema ekstremum

Out[4]= {-6.33825 10^{28} , 1.0114 10^{72} , {{-4., 36.736}, {-4.1, 38.8881}, {-4.3, 43.3994},
{-4.7, 53.2898}, {-5.5, 76.9076}, {-7.1, 142.499}, {-10.3, 357.581},
{-16.7, 1174.24}, {-29.5, 4806.91}, {-55.1, 22740.1}, {-106.3, 117048.},
{-208.7, 630730.}, {-413.5, 3.48107 10^{6} }, {-823.1, 1.94489 10^{7} },
{-1642.3, 1.09336 10^{8} }, {-3280.7, 6.1657 10^{8} }, {-6557.5,3.4824 10^{9} },
{-13111.1, 1.9684 10^{10} }, {-26218.3, 1.11306 10^{11} }, {-52432.7, 6.2952 10^{11} },
{-104862., 3.56076 10^{12} }, {-209719., 2.01417 10^{13} }, {-419434., 1.13936 10^{14} },
{-838865., 6.44511 10^{14} }, {-1.67773 10^{6} , 3.64588 10^{15} },
{-3.35545 10^{6} , 2.06241 10^{16} }, {-6.71089 10^{6} , 1.16668 10^{17} }, ...
{-1.58456 10^{28} , 3.16063 10^{70} }, {-3.16913 10^{28} , 1.78792 10^{71} },
{-6.33825 10^{28} , 1.0114 10^{72} }}}

In[5]:= simplexII[x^2,{x},-1,0.1,0.01] (*minimum*)

Out[5]= {-0.3, 0.09, {{-1., 1.}, {-0.9, 0.81}, {-0.7,0.49}, {-0.3, 0.09}}}

In[6]:= simplexI[x^2,{x},-1,1,0.1,0.01] (*minimum*)

Out[6]= {-1.38778 10^{-16},1.92593 10^{-32} ,
{{-1, 1.}, {-0.9, 0.81}, {-0.8, 0.64}, {-0.7, 0.49}, {-0.6,0.36},
{-0.5, 0.25}, {-0.4, 0.16}, {-0.3, 0.09}, {-0.2, 0.04}, {-0.1, 0.01},
{-1.38778 10^{-16} , 1.92593 10^{-32} }}}

In[7]:= simplexII[Sin[x],{x}, -Pi ,0.1,0.001] (*maksimum*)

Out[7]= {-4.64159, 0.997495, {{-3.14159, 0}, {-3.24159,0.0998334}, {-3.44159, 0.29552},
{-3.84159, 0.644218}, {-4.64159, 0.997495}}}

Iz poslednjeg primera se može zaključiti da se kod druge varijante simplex metoda zbog uvećavanja koraka skaniranja, kod periodičnih i drugih funkcija koje imaju više ekstremnih vrednosti, često "preskače" najbliža ekstremna

vrednost u odnosu na polaznu tačku, a locira se neka udaljenija. Zato se može desiti slučaj da se "preskoči" globalni ekstremum i da kao rezultat dobijemo neki od lokalnih ekstremuma. Ovo je, evidentno, nedostatak druge varijante simplex metoda.

S druge strane, druga varijanta simpleks metoda je bolja za ispitivanje funkcija definisanih na čitavoj realnoj pravoj, jer se njome može konstatovati nepostojanje zahtevane ekstremne vrednosti, dok se prvim varijantom metoda za ekstremnu vrednost proglašava najveća, odnosno najmanja vrednost funkcije na konkretnom intervalu.

1.2.4. METOD DIHOTOMIJE

Metod dihotomije (deljenja na pola) predstavlja posebno skeniranje unutar intervala $[a, b]$, pri čemu se u svakoj iteraciji oblast smanjuje dva puta, a u nekim slučajevima četiri puta. Najpre se izračuna vrednost funkcije u pet tačaka, koristeći korak skaniranja $\Delta = (b - a)/4$. Od pet dobijenih rezultata odabira se najbolji, označen sa qm i odgovarajuća vrednost xm. Ako vrednost xm odgovara jednoj od granica a ili b, pretraživanje se nastavlja na 1/4 oblasti i izračunavaju se vrednosti ciljne funkcije u tri nove tačke. Ako je dobijena vrednost unutar intervala $[a, b]$, odbacuje se 1/2 oblasti i izračunavaju se dve nove vrednosti ciljne funkcije. Izlazni kriterijum je $\Delta < \Delta_{\min}$, pri čemu je $\Delta_{\min}$ minimalna vrednost koraka.

Ulazne veličine:

$q\_$, $pr\_$: ciljna funkcija i lista sa parametrom ciljne funkcije;

$a\_$, $b\_$: granice skaniranja;

$dmin\_$: minimalna vrednost koraka.

Lokalne promenljive:

del: parametar koraka;

izb: parametar koji određuje lokalizaciju minimuma, odnosno maksimuma.

x, qa, qb, $q1$, $q2$, $q3$: tekuća tačka i vrednosti ciljne funkcije;

xm, qm: tekuća optimalna tačka i optimalna vrednost;

$xmax$, $qmax$: optimalna tačka i optimalna vrednost.

Algoritam:

Korak 1. Postaviti $del = (b - a)/4$, a zatim generisati četiri ekvidistantne tačke u intervalu $[a, b]$:

$qa = q(a)$, $x1 = a + 2 * del$, $q1 = q(x1)$,
$x2 = a + del$, $q2 = q(x2)$, $x3 = b - del$, $q3 = q(x3)$, $qb = q(b)$.

Korak 2. U slučaju minimuma izračunati

$qm = qmax = \min\{qa, q2, q1, q3, qb\}$,

a za slučaj maksimuma izračunati

$qm = qmax = \max\{qa, q2, q1, q3, qb\}$.

Korak 3. `While` ciklus, koji se prekida kada je ispunjen uslov $|del| < dmin$. Unutar ciklusa izvršiti sledeće korake:

Korak 3.1. Ako je vrednost qm "bolja" od vrednosti $qmax$ uraditi:

Korak 3.1.1. Selektovati sledeće slučajeve:

A. Za $qm = qa$ postaviti
$b = x2$; $qb = q2$; $x1 = a + (b - a)/2$; $q1 = q(x1)$; $xm = a$;
B. Za $qm = qb$ postaviti
$a = x3$; $qa = q3$; $x1 = a + (b - a)/2$; $q1 = q(x1)$; $xm = b$;
C. Za $qm = q1$ postaviti
$a = x2$; $b = x3$; $qa = q2$; $qb = q3$; $xm = x1$;
D. Za $qm = q2$ postaviti
$b = x1$; $qb = q1$; $x1 = x2$; $q1 = q2$; $xm = x2$;
E. Za $qm = q3$ postaviti
$a = x1$; $qa = q1$; $x1 = x3$; $q1 = q3$; $xm = x3$;

Korak 3.1.2. Postaviti:

$del = (b - a)/4$, $x2 = a + del$, $q2 = q(x2)$, $x3 = b - del$,
$q3 = q(x3)$.

Korak 3.2. Ako vrednost qm nije "bolja" od vrednosti $qmax$ uraditi:

$a = a + del$; $b = b - del$; $del = (b - a)/4$;
$x2 = a + del$; $x3 = b - del$; $x1 = a + 2 * del$;
$qa = N[q/.pr[[1]] -> c]$; $qb = N[q/.pr[[1]] -> d]$;
$q1 = N[q/.pr[[1]] -> x1]$; $q2 = N[q/.pr[[1]] -> x2]$;
$q3 = N[q/.pr[[1]] -> x3]$;

Korak 3.3. U slučaju minimuma izračunati $qm = \min\{qa, q2, q1, q3, qb\}$, a za slučaj maksimuma $qm = \max\{qa, q2, q1, q3, qb\}$.

Korak 4. Izlazne veličine su $xmax$ i $qmax$.

```
dih[q_,pr_List,a_,b_,dmin_]:=
   Block[{c=a,d=b,qa,qb,q2,q3,x1=c+(d-c)/2,q1,
          del=N[(d-c)/4],x2=c+del,x3=d-del,qm,xm,l,
```

```
      dm=dmin,izb,qmax,xmax, Lista ={} },
qa=N[q/.pr[[1]]->c]; qb=N[q/.pr[[1]]->d];
q1=N[q/.pr[[1]]->x1]; q2=N[q/.pr[[1]]->x2];
q3=N[q/.pr[[1]]->x3];
izb=Input["1 za minimum, 2 za maksimum "];
If[izb==2, qm=qmax=Max[qa,qb,q1,q2,q3],
           qm=qmax=Min[qa,qb,q1,q2,q3]
];
Which[qm==qa,xmax=c,
      qm==qb,xmax=d,
      qm==q1,xmax=x1,
      qm==q2,xmax=x2,
      True, xmax=x3
];
Lista=Append[Lista,{xmax,qmax}];
While[Abs[del]>dm,
  If[(izb==1&&N[qm]<N[qmax])||(izb==2&&N[qm]>N[qmax]),
     qmax=qm;
     Which[qm==qa, d=x2;qb=q2;x1=c+(d-c)/2;
                   q1=N[q/.pr[[1]]->x1];xmax=c,
           qm==qb, c=x3;qa=q3;x1=c+(d-c)/2;
                   q1=N[q/.pr[[1]]->x1];xmax=d,
           qm==q1, c=x2;d=x3;qa=q2;qb=q3;xmax=x1,
           qm==q2, d=x1;qb=q1;x1=x2;q1=q2;xmax=x2,
           True,   c=x1;qa=q1;x1=x3;q1=q3;xmax=x3
     ];
     Lista=Append[Lista,{xmax,qmax}];
     del=(d-c)/4;
     x2=c+del; q2=N[q/.pr[[1]]->x2];
     x3=d-del; q3=N[q/.pr[[1]]->x3],
     c=c+del; d=d-del;
     del=(d-c)/4;
     x2=c+del; x3=d-del; x1=c+2*del;
     qa=N[q/.pr[[1]]->c]; qb=N[q/.pr[[1]]->d];
     q1=N[q/.pr[[1]]->x1]; q2=N[q/.pr[[1]]->x2];
     q3=N[q/.pr[[1]]->x3];
  ];
  If[izb==2,qm=Max[qa,qb,q1,q2,q3],
            qm=Min[qa,qb,q1,q2,q3]
```

```
    ]
  ];
  {N[xmax],N[qmax], Lista}
]
```

Test primeri. Korišćene su sledeće test funkcije:

$q1[x] := x^2, \quad q2[x] := N[Sin[x-1]] + x^2 * N[Sqrt[Abs[x-1]]].$

In[1]:= dih[Sin[x-1]+x^2*Sqrt[Abs[x-1]],{x},-1,1,0.001] (*minimum*)

Out[1]= {-0.172607, -0.889502, {{0, -0.841471}, {-0.21875,-0.885842}, {-0.148437, -0.888512}, {-0.183594, -0.889298}, 0.166016, -0.889428}, {-0.174805, -0.889495}, {-0.172607, -0.889502}}}

In[2]:=dih[Sin[x-1]+x^2*Sqrt[Abs[x-1]],{x},-1,1,0.001] (*maksimum*)

Out[2]= {-1., 0.504916, {{-1, 0.504916}}}

In[3]:=dih[x^2,{x},-1,1,0.001] (*minimum*)

Out[3]= {0, 0, {{0, 0}}}

In[4]:= dih[x^2,{x},-1,1,0.001] (*maksimum *)

Out[4]= {-1,1,{{-1., 1.}}}

1.2.5. METOD ZLATNOG PRESEKA

Brža konvergencija u prethodnom metodu se može dobiti ako se interval $[a, b]$ ne deli celim brojem, već iracionalnim brojem. Jedan od takvih metoda je metod *zlatnog preseka*. On je primenljiv i na nedifirencijabilne funkcije. Pretpostavlja se da je funkcija $Q(x)$ definisana i neprekidna na intervalu $[a, b]$ i da na tom intervalu ima samo jedan lokalni ekstremum. Algoritam se sastoji u sledećem.

Izračunava se vrednost funkcije na krajevima intervala kao i u dve unutrašnje tačke x_1 i x_2. Između četiri izračunate vrednosti izdvaja se ona koja je najbolja, u smislu da je najveća (kod maksimizacije) ili da je najmanja (kod minimizacije). Ako je najbolja vrednost u tački x_1, postupak se nastavlja na intervalu $[a, x_2]$, a ako je najbolja vrednost u tački x_2, postupak se nastavlja na intervalu $[x_1, b]$. Neka je najbolja vrednost u x_1. Na intervalu $[a, x_2]$ postavljamo: $b = x_2$, $x_2 = x_1$, a zatim izračunavamo vrednost funkcije u jednoj unutrašnjoj tački, koju označavamo sa x_1. Pogodno je da se novom tačkom x_1 sledeći interval podeli slično prethodnom intervalu:

$$b - x_2 = x_1 - a, \qquad \frac{b - x_1}{b - a} = \frac{x_1 - a}{b - x_1} = \xi.$$

Odavde se dobija sledeći sistem linearnih jednačina:

$$\begin{aligned} b - x_1 &= \xi(b-a), \\ x_1 - a &= \xi(b-x_1), \\ b - x_2 &= x_1 - a. \end{aligned}$$

Rešenje ovih jednačina je

$$x_1 = a + \xi(b-a), \quad x_2 = b - \xi(b-a), \qquad \xi = \frac{2}{3+\sqrt{5}} \approx 0,381966.$$

To znači da je u svakoj iteraciji

$$x_1 = a + \xi(b-a), \qquad x_2 = b - \xi(b-a).$$

U slučaju da je $Q(x_1) = Q(x_2)$, proces optimizacije se nastavlja na intervalu $[a,b] = [x_1, x_2]$, izračunavanjem vrednosti funkcije u dve unutrašnje tačke određene prema poslednjoj formuli.

Proces se prekida kada se ispuni uslov $\mid b - a \mid < \varepsilon$, gde je ε unapred zadata tačnost. U tom slučaju, ekstremna vrednost funkcije jednaka je $Q((a+b)/2)$.

Algoritam metoda zlatnog preseka:

Korak 1. Odrediti početni interval $[a,b]$ i specificirati tačnost e.

Korak 2. Izračunati
$x1 = a + \xi(b-a)$; $x2 = a + (1-\xi)(b-a) = a + b - x1$;
$q1 = q(x1)$; $q2 = q(x2)$; $d = b - a$;

Korak 3. `While` ciklus, koji se prekida u slučaju $d \leq e$. Unutar ciklusa izvršiti sledeće korake:

Korak 3.1. Ako je $q1 = q2$ postaviti $d = x2 - x1$.
Moguća su sledeća dva slučaja:

A1. Za $d > e$ postaviti
$a = x1$; $b = x2$; $x1 = a + \xi(b-a)$; $x2 == a + b - x1$;
$q1 = q(x1)$; $q2 = q(x2)$; $d = b - a$;

A2. Za $d \leq e$ postaviti $d = d/2$; $xm = x1 + d$; $qm = q(xm)$

Korak 3.2. Ako je $q1 \neq q2$ izračunati $d = (1-\xi)(b-a)$.
Moguća su dva slučaja:

B1. Ako je vrednost $q2$ "bolja" od vrednosti $q1$ ispitati sledeće slučajeve:

C1. Ako je $d > e$ izračunati

$a = x1$; $x1 = x2$; $q1 = q2$; $x2 = (1 - \xi)(b - a)$; $q2 = q(x2)$;

C2. Ako je $d \leq e$ postaviti $xm = x2$; $qm = q2$;

B2. Ako je vrednost $q2$ "lošija" od $q1$ ispitati slučajeve:

D1. Ako je $d > e$ izračunati

$b = x2$; $x2 = x1$; $q2 = q1$; $x1 = a + \xi(b - a)$; $q1 = q(x1)$;

D2. Ako je $d \leq e$ postaviti $xm = x1$; $qm = q1$;

Korak 4. Rezultat je lista sa vrednostima xm i qm.

Sledi implementacija u paketu MATHEMATICA.

```
zlatni[q_,pr_,dg_,gg_,e_]:=
    Block[{a=dg,b=gg,x1,x2,q1,q2,ksi,xm,qm,d,izb},
      izb=Input["1 za min 2 za max "];
      ksi=N[2/(3+Sqrt[5])];
      x1=a+ksi*(b-a); x2=a+b-x1;
      q1=N[q/.pr[[1]]->x1]; q2=N[q/.pr[[1]]->x2];
      Print[{a,x1,x2,b},{q1,q2}];
      d=Abs[b-a];
      While[d>e,
         If[q1==q2,
            d=x2-x1;
            If[d>e,
                a=x1; b=x2;
                x1=a+ksi*(b-a); x2=a+b-x1;
                q1=N[q/.pr[[1]]->x1]; q2=N[q/.pr[[1]]->x2],
                d=d/2; xm=x1+d; qm=N[q/.pr[[1]]->xm]
            ]
         ];
         If[q1=!=q2,
             d=(1-ksi)*(b-a);
             If[(q2>q1 && izb==2) || (q2<q1 && izb==1),
                If[d>e,
                   a=x1; x1=x2; q1=q2;
                   x2=a+b-x1; q2=N[q/.pr[[1]]->x2],
                   xm=x2; qm=q2
                ],
```

```
                If[d>e,
                   b=x2; x2=x1; q2=q1;
                   x1=a+ksi*(b-a); q1=N[q/.pr[[1]]->x1],
                   xm=x1; qm=q1
                ]
             ]
          ];
          Print[{a,x1,x2,b},{q1,q2}];
       ];
       Return[{xm,qm}]
    ]
```

Test primeri.

```
In[1]:= zlatni[x^2-1,{x},-5,5,0.01]
1 za min 2 za max 1
{-5, -1.18034, 1.18034, 5}{0.393202, 0.393202}
{-1.18034, -0.27864, 0.27864, 1.18034}{-0.922359, -0.922359}
{-0.27864, -0.0657781, 0.0657781, 0.27864}{-0.995673, -0.995673}
{-0.0657781, -0.0155281, 0.0155281, 0.0657781}{-0.999759, -0.999759}
{-0.0155281, -0.00366569, 0.00366569, 0.0155281}{-0.999987, -0.999987}
{-0.0155281, -0.00366569, 0.00366569, 0.0155281}{-0.999987, -0.999987}
Out[1]= {0., -1.}
```

Odgovarajuća funkcija u LISPu je napisana za slučaj minimizacije.

```
(define (goldsec q a b dmin)
 (let ((f1 1) (f2 1) (delta 1) (x1 1) (x2 1) (xm 1) (qm 1)
          (f 1) (q1 1) (q2 1) (qa 1) (qb 1))
     ; Korak 1.  Definisati Fibonaccijeve brojeve f1, f2
     ; i ciljnu funkciju f
     (set!  f1 (/ (- 3.0 (sqrt 5.0)) 2.0))
     (set!  f2 (- 1.0 f1))
     (set!  f (eval (list 'lambda (cadr q) (car q))))
     ; Korak 2.  Početno razbijanje intervala [a,b]
     (set!  x1 (+ a (* f1 (- b a))))
     (set!  x2 (+ a (* f2 (- b a))))
     (set!  q1 (apply f x1)) (set!  q2 (apply f x2))
     (set!  qa (apply f a)) (set!  qb (apply f b))
     ; Korak 3.  Ciklus
     (do ()
        ( (< delta dmin) (list xm qm delta))
        (cond ( (= q1 q2)   ; case f(x1) = f(x2)
```

```
      (set!  delta (- x2 x1))
      (cond ( (> delta dmin)
           (set!  a x1) (set!  b x2)
           (set!  x1 (+ a (* f1 (- b a))))
           (set!  x2 (+ a (* f2 (- b a))))
           (set!  q1 (apply f x1))
           (set!  q2 (apply f x2))
           (set!  qa (apply f a))
           (set!  qb (apply f b))
         )
         (t (set!  delta (/ delta 2.0))
           (set!  xm (+ delta x1)) (set!  qm (apply f xm))
    ) )  )
    (t
      (set!  delta (* f2 (- b a)))
      (cond ( (< q2 q1)   ; case f(x2) < f(x2)
      (cond ( (> delta dmin)
          (set!  a x1) (set!  x1 x2) (set!  q1 q2)
          (set!  x2 (+ a (* f2 (- b a))))
          (set!  q2 (apply f x2))
      )
      (t (set!  xm x1) (set!  qm q1)
    ) ) )
      (t (cond ( (> delta dmin)   ; case f(x2) > f(x1)
         (set!  b x2)
         (set!  x2 x1) (set!  q2 q1)
         (set!  x1 (+ a (* f1 (- b a))))
         (set!  q1 (apply f x1))
       )
       (t (set!  xm x1) (set!  qm q1)
) ) ) ) ) ) ) ) )
```

Rezultati testiranja programa. Primenjujući metod zlatnog preseka sa minimalnim korakom 10^{-5} na ciljnu funkciju $x \mapsto 2x^4 - 3x$, dobija se sledeća lista apscisa:

$$(a = 0.721114411901743 \quad x_1 = 0.72112037276273$$
$$x_2 = 0.721124056777422 \quad b = 0.721130017638408).$$

Rezultujući par koji sadrži optimalnu vrednost argumenta i pripadajuću vrednost ciljne funkcije, i dobijen je u 24. koraku:

$$(0.72112037276273 \quad -1.62253076647434).$$

Vrednost poslednjeg koraka d je $0.964487567840431 * 10^{-5}$. Odgovarajući rezultat u [73] je dobijen u 24. koraku:

$$(0.7211207 \quad 0.7210795 \quad 0.7211866 \quad 0.7211371)$$

Međutim, može se dobiti numerički rezultat sa većom tačnošću od odgovarajućeg u [71]. Na primer, koristeći metod zlatnog preseka sa vrednošću minimalnog koraka 10^{-8}, dobija se sledeći rezultat:

$$(0.721124769617705 \quad -1.62253076659583).$$

Najveća preciznost u SCHEME je 10^{-15}. Metod zlatnog preseka, primenjen sa vrednošću minimalnog koraka 10^{-15}, daje rezultat u 56 koraka. Poslednjih pet iteracija generiše sledeće rezultate:

$$\begin{aligned}
&(a = 0.72112478450948 \quad x_1 = 0.721124784509484\\
&x_2 = 0.721124784509486 \quad b = 0.72112478450949)\\
&(a = 0.72112478450948 \quad x_1 = 0.721124784509482\\
&x_2 = 0.721124784509484 \quad b = 0.721124784509486)\\
&(a = 0.721124784509482 \quad x_1 = 0.721124784509484\\
&x_2 = 0.721124784509485 \quad b = 0.721124784509486)\\
&(a = 0.721124784509483 \quad x_1 = 0.721124784509484\\
&x_2 = 0.721124784509484 \quad b = 0.721124784509485).
\end{aligned}$$

Poslednja vrednost koraka je jednaka $0.960617790035482 * 10^{-15}$, a poslednja aproksimacija lokalnog minimuma je

$$x^* \approx x_{56} = 0.721124784509484.$$

1.2.6. METOD DAVIES-SWANN-CAMPEY (DSC)

Pri izvršenju DSC algoritma povećava se veličina koraka, sve dok se ne prevaziđe ekstremum, a zatim se definiše kvadratna interpolacija na osnovu dobijenih rezultata.

Algoritam ovog metoda (u slučaju maksimuma) je sledeći:

Korak 1. Izabrati početnu tačku $x^{(0)} \in \mathbb{R}$, početni korak $\Delta^{(0)}$ i minimalni korak $\Delta_{\min}$.

Korak 2. Izračunati $Q^{(0)} = Q(x^{(0)})$.

Korak 3. Izračunati $Q^{(1)} = Q(x^{(1)}) = Q(x^{(0)} + \Delta^{(1)})$, sa $\Delta^{(1)} = \Delta^{(0)}$.

Korak 4. Ako je $Q^{(1)} \geq Q^{(0)}$, preći na *Korak 7*.

Korak 5. Ako je $Q^{(1)} < Q^{(0)}$ postaviti $\Delta^{(0)} = -\Delta^{(0)}$ i vratiti se na *Korak 3*.

Korak 6. Ako počev od $x^{(0)}$, koristeći $\pm\Delta^{(0)}$, nije dobijen zadovoljavajući rezultat iskoristiti tri izračunate vrednosti ciljne funkcije $Q(x)$ za kvadratnu interpolaciju i preći na *Korak 13*, inače preći na sledeći korak.

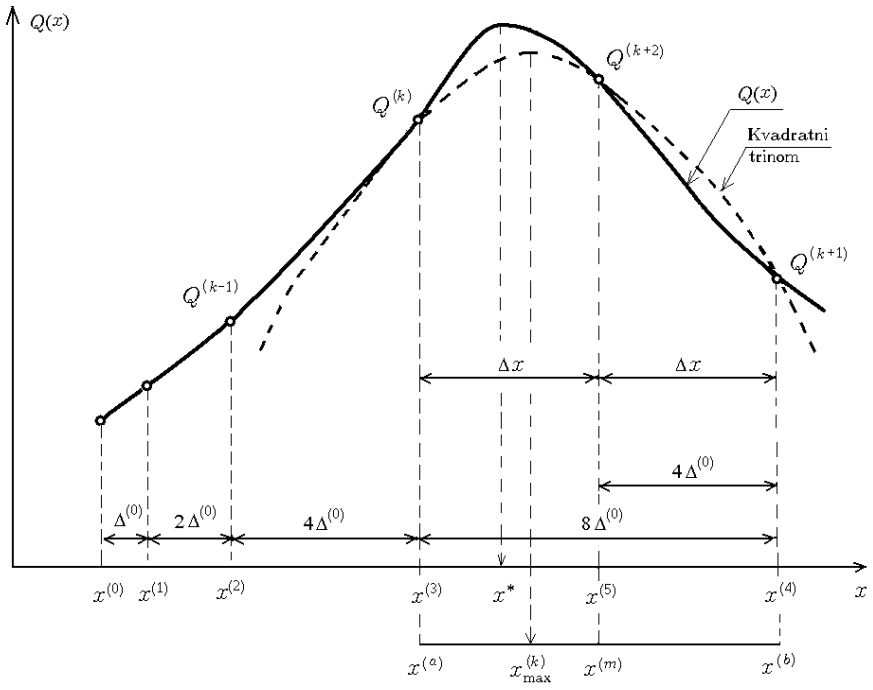
Korak 7. Udvostručiti korak

$$\Delta^{(k+1)} = 2\Delta^{(k)}.$$

Korak 8. Izračunati

$$x^{(k+1)} = x^{(k)} + \Delta^{(k+1)}, \quad Q^{(k+1)} = Q(x^{(k+1)}).$$

Korak 9. Ako je $Q^{(k+1)} \geq Q^{(k)}$, algoritam produžiti od *Koraka 7*, inače od *Koraka 10*.



Sl. 1.2.2

Korak 10. Izračunati vrednost funkcije cilja $Q(x)$ u suprotnom pravcu od $x^{(k+1)}$, na rastojanju jednakom polovini tekućeg koraka:

$$x^{(k+2)} = x^{(k+1)} - \frac{\Delta^{(k+1)}}{2}, \quad Q^{(k+2)} = Q(x^{(k+2)}).$$

Korak 11. Na osnovu četiri vrednosti ciljne funkcije $Q(x)$, koje su izračunate ekvidistantno sa jednakim korakom upravljačkog parametra x (označimo ih redom sa $x^{(k-1)}$, $x^{(k)}$, $x^{(k+2)}$ i $x^{(k+1)}$), odbacuje se $x^{(k+1)}$ ili $x^{(k-1)}$, zavisno od položaja maksimalne vrednosti funkcije $Q(x)$.

Korak 12. Izračunati vrednost sledećeg koraka

$$\Delta x = \frac{\Delta^{(k+1)}}{2}.$$

Korak 13. Za tri vrednosti ciljne funkcije i odgovarajuće vrednosti upravljačkog parametra x koristimo sledeće oznake:

$$Q_m = Q(x^{(m)}), \quad Q_a = Q(x^{(a)}), \quad Q_b = Q(x^{(b)}),$$

gde je $x^{(m)}$ centralna tačka, dok su tačke x_a i x_b određene sa $x^{(a)} = x^{(m)} - \Delta x$ i $x^{(b)} = x^{(m)} + \Delta x$.

Korak 14. Prema kvadratnoj interpolaciji tekuće približavanje maksimumu $x_{\max}^{(k)}$ definisano je izrazom

$$x_{\max}^{(k)} = x^{(m)} + \frac{\Delta x(Q_a - Q_b)}{2(Q_a - 2Q_m + Q_b)}.$$

Korak 15. Ako je $|\Delta x| \leq \Delta_{\min}$ traženje se završava, izračunava se $Q(x_{\max}^{(k)})$ i edituju vrednosti za $x^* = x_{\max}^{(k)}$ i $Q(x^*)$.

Korak 16. Ako je $|\Delta x| > \Delta_{\min}$ smanjiti korak L puta ($1 \leq L \leq 4$):

$$\Delta^{(0)} = \frac{\Delta^{(0)}}{L}.$$

Korak 17. Uzeti $x^{(0)} = x_{\max}^{(k)}$ i vratiti se na *Korak 2*.

Prednost DSK algoritma je u tome što ne zahteva granice za upravljački parametar u toku pretraživanja.

Nedostatak algoritma je u velikom broju izračunavanja vrednosti ciljne funkcije dok se ne uđe u oblast maksimuma, naročito kada je početna tačka $x^{(0)}$ daleko od maksimuma, a početni korak $\Delta^{(0)}$ mali.

```
Dsk[q_,pr_List,x01_,del0_,delmin_]:=
   Block[{x0=x01,q0,q1,q2,q3,x1,x2,x3,del1,dx=10,xkmax,
```

```
   qmax,xm,xa,xb,qm,qa,qb,d=del0,it=0,izb, Lista={} },
izb=Input["1 za minimum, 2 za maksimum "];
(* Korak 15 *)
While[Abs[dx]>=delmin && it<100,
   (* Korak 2 *)
   q0=N[q/.pr[[1]]->x0];
   Lista=Append[Lista,{x0,q0}];
   (* Korak 3 *)
   x1=x0+d; q1=N[q/.pr[[1]]->x1];
   (* Korak 5 *)
   If[(izb==2&&q0>q1) || (izb==1&&q0<q1),
      d=-d; x2=x0+d; q2=N[q/.pr[[1]]->x2];
      (* Korak 6 *)
      If[(izb==2&&q0>q2)||(izb==1&&q0<q2),
         xa=x2;  xm=x0;   xb=x1,
         q1=q2; x1=x2
      ];
   ];
   (* Korak 4 *)
   If[(izb==2&&q1>q0) || (izb==1&&q1<q0),
      del1=2*d;
      x2=x1+del1; q2=N[q/.pr[[1]]->x2];
      (* Koraci 7,8,9 *)
      While[((izb==2&&q2>q1)||(izb==1&&q2<q1))
                          &&(it<100),
            x0=x1;  q0=q1;
            x1=x2;  q1=q2;
            d=del1; del1=2*d;
            x2=x1+del1; q2=N[q/.pr[[1]]->x2];
            it+=1
      ];
      (* Korak 10 *)
      x3=x2-del1/2; q3=N[q/.pr[[1]]->x3];
      (* Korak 11 *)
      If[(izb==2&&q3>q1) || (izb==1&&q3<q1),
          x0=x1; q0=q1; x1=x3; q1=q3,
          x2=x3; q2=q3
      ];
      (* Korak 12 *)
```

```
            dx=d/2;
            xm=x1;  xa=x0;   xb=x2;
          ];
          (* Korak 13 *)
          qa=N[q/.pr[[1]]->xa]; qm=N[q/.pr[[1]]->xm];
          qb=N[q/.pr[[1]]->xb];
          (* Korak 14 *)
          xkmax=xm+dx*(qa-qb)/(2*(qa-2*qm+qb));
          qmax=N[q/.pr[[1]]->xkmax];
          Lista=Append[Lista,{xkmax,qmax}];
          (* Korak 16 *)
          d /=4;
          (* Korak 17 *)
          x0=xkmax;
          it+=1;
        ];
        qmax=N[q/.pr[[1]]->xkmax];
        Return[{xkmax,qmax, Lista}]
    ]
```

Odgovarajuća funkcija u LISPu, za slučaj maksimuma ciljne funkcije ima oblik:

```
(define (dsk q x dxmin)
 ; Priprema:  Deklarisanje promenljivih i ciljne funkcije f
 (let ((fun '()) (x0 x) (x1 x) (x2 x) (x3 x) (xmin x) (xa x)
       (xb x) (xc x) (fx0 1) (fx1 1) (fx2 1) (fx3 1) (fxmin 1)
       (fxa x) (fxb x) (fxc x) (delta 0.001) (dx 0.001))
       (set!  fun (eval (list 'lambda (cadr q) (car q) ) ) )
      ; Glavna petlja iteracije:
      (do ()
       ((< (abs dx) dxmin) (list xmin fxmin))
       (set!  dx delta) (set!  fx0 (apply fun x0))
       (set!  x1 (+ x0 dx)) (set!  fx1 (apply fun x1))
       (set!  dx (if (> fx1 fx0) (- 0.0 dx) dx))
       (set!  x1 (+ x0 dx)) (set!  fx1 (apply fun x1))
       (set!  dx (* 2.0 dx))
       (set!  x2 (+ x1 dx)) (set!  fx2 (apply fun x2))
       (do ()
           ( (> fx2 fx1) ())
```

```
            (set!  x0 x1) (set!  fx0 fx1)
            (set!  x1 x2) (set!  fx1 fx2)
            (set!  dx (* 2.0 dx))
            (set!  x2 (+ x1 dx)) (set!  fx2 (apply fun x2)))
        (set!  x3 (/ (+ x1 x2) 2.0))
        (set!  fx3 (apply fun x3))
        (set!  dx (/ dx 2.0))
        (if (> fx0 fx2)
            (begin
               (set!  xa x1) (set!  fxa fx1)
               (set!  xb x3) (set!  fxb fx3)
               (set!  xc x2) (set!  fxc fx2))
            (begin
               (set!  xa x0) (set!  fxa fx0)
               (set!  xb x1) (set!  fxb fx1)
               (set!  xc x3) (set!  fxc fx3)))
        (set!  xmin (+ xb
                  (/ (* dx (- fxa fxc))
                    (* 2.0 (- (+ fxa fxc) (* 2.0 fxb))))))
        (set!  fxmin (apply fun xmin))
        (set!  x0 (if (< fxmin fxb) xmin xb))
        (set!  delta (abs (/ delta 2.)))
) ) )
```

1.2.7. JEDNODIMENZIONALNI POWELOV METOD

U metodu Powela kvadratna aproksimacija se vrši na osnovu rezultata dobijenih u prva tri koraka. U slučaju maksimuma, algoritam ovog metoda se može opisati na sledeći način.

Korak 1. Izabrati početnu tačku $x^{(0)}$, početni korak $\Delta^{(0)}$ i minimalni korak $\Delta_{\min}$.

Korak 2. Izračunati $Q^{(0)} = Q(x^{(0)})$.

Korak 3. Izračunati $Q^{(1)} = Q(x^{(1)}) = Q(x^{(0)} + \Delta^{(0)})$.

Korak 4. Ako je $Q^{(1)} > Q^{(0)}$, izračunati $x^{(2)} = x^{(1)} + 2\Delta^{(0)}$ i preći na *Korak 6*.

Korak 5. Ako je $Q^{(1)} \leq Q^{(0)}$ staviti $\Delta^{(0)} = -\Delta^{(0)}$ i vratiti se na *Korak 3*.

Korak 6. Izračunava se $Q^{(2)} = Q(x^{(2)})$.

Korak 7. Izračunava se vrednost upravljačkog parametra, koja određuje približnu vrednost ekstremuma funkcije $Q(x)$, a koja se dobija na

osnovu izraza

$$x_{\max}^{(k)} = \frac{A}{2B}\,, \tag{1.2.1}$$

pri čemu su

$$A = \big[(x^{(1)})^2 - (x^{(2)})^2\big]Q^{(0)} + \big[(x^{(2)})^2 - (x^{(0)})^2\big]Q^{(1)} + \big[(x^{(0)})^2 - (x^{(1)})^2\big]Q^{(2)},$$

$$B = (x^{(1)} - x^{(2)})Q^{(0)} + (x^{(2)} - x^{(0)})Q^{(1)} + (x^{(0)} - x^{(1)})Q^{(2)}.$$

Korak 8. U skupu $\{x^{(0)}, x^{(1)}, x^{(2)}\}$ određuje se vrednost $x = x^{(t)}$ za koju ciljna funkcija ima maksimalnu vrednost, tj.

$$Q(x^{(t)}) = \max_x\{Q^{(0)}, Q^{(1)}, Q^{(2)}\}.$$

Korak 9. Pretraživanje prekinuti ako je ispunjen uslov

$$\Delta^{(t)} = |x_{\max}^{(k)} - x^{(t)}| \le \Delta_{\min}. \tag{1.2.2}$$

Tada se izračunava vrednost ciljne funkcije $Q_{\max} = Q(x_{\max}^{(k)})$ i edituju vrednosti: $Q_{\max}$, $x^* = x_{\max}^{(k)}$, $\Delta^{(t)}$.

Ako uslov nije ispunjen, produžiti od *Koraka 10*.

Korak 10. Izračunava se $Q(x_{\max}^{(k)})$ i iz skupa $\{x^{(0)}, x^{(1)}, x^{(2)}\}$ se isključuje tačka u kojoj ciljna funkcija ima minimalnu vrednost. Isključena tačka se zamenjuje sa $x_{\max}^{(k)}$, a algoritam se nastavlja od *Koraka 7*.

Nedostatak algoritma je neophodnost da imenilac izraza (1.2.1) bude različit od nule.

Naveden je listing funkcije koja odgovara Powelovom metodu.

```
Powel[q_,pr_List,x01_,del0_,delmin_]:=
   Block[{x0=x01,q0,q1,q2,x1,x2,in=0,xkmax,
          qmax,qxt,xt,xkmin,a,b,
          delt=10,p,it, d=del0, izb, Lista={} },
      izb=Input["1 za minimum, 2 za maksimum"];
      (* Korak 2.  *)
      q0=N[q/.pr[[1]]->x0];
      Lista=Append[Lista,{x0,q0}];
```

```
(* Korak 3.  *)
x1=x0+d; q1=N[q/.pr[[1]]->x1];
If[(izb==2&&q1<=q0&&in==0)||(izb==1&&q1>=q0&&in==0),
    d=-d; x1=x0+d; q1=N[q/.pr[[1]]->x1];
    in=1
];
If[(izb==2&&q1<=q0&&in==1)||(izb==1&&q1>=q0&&in==1),
   Print["Tacnost= ",d];
   Return[{x0,q0, Lista}]
];
x2=x1+2*d; q2=N[q/.pr[[1]]->x2];
it=2;
While[Abs[delt]>=delmin,
     (* Korak 7.  *)
     a=(x1^2-x2^2)*q0+(x2^2-x0^2)*q1+(x0^2-x1^2)*q2;
     b=(x1-x2)*q0+(x2-x0)*q1+(x0-x1)*q2;
     xkmax=a/(2*b); qmax=N[q/.pr[[1]]->xkmax];
     Lista=Append[Lista,{xkmax,qmax}];
     (* Korak 8.  *)
     If[izb==1,qxt=Min[q0,q1,q2],qxt=Max[q0,q1,q2]];
     Which[qxt==q0, xt=x0,
           qxt==q1, xt=x1,
           qxt==q21, xt=x2
     ];
     (* Korak 10.  *)
     delt=Abs[xkmax-xt];
     If[izb==2, qxt=Min[q0,q1,q2],
                qxt=Max[q0,q1,q2] ];
     Which[qxt==q0, x0=xkmax; q0=qmax,
           qxt==q1, x1=xkmax; q1=qmax,
           qxt==q2, x2=xkmax; q2=qmax
     ];
     (* Ocuvanje redosleda x0<x1<x2 *)
     If[x1<x0,
           p=x0; x0=x1; x1=p; p=q0; q0=q1; q1=p
     ];
     If[x2<x0,
           p=x0; x0=x2; x2=p; p=q0; q0=q2; q2=p
     ];
```

```
            If[x2<x1,
                  p=x1; x1=x2; x2=p; p=q1; q1=q2; q2=p
            ];
            it+=1;
            If[it>100, Print["Tacnost nije dostignuta "];
                Return[xkmax,N[q/.pr[[1]]->xkmax], Lista]
            ]
        ];
        Print["tacnost= ", delt];
        Return[{xkmax,N[q/.pr[[1]]->xkmax], Lista}];
    ]
```

1.2.8. DSC-POWELOV METOD

Metod *DSC*-Powela predstavlja kombinaciju dva prethodno razmatrana metoda: DSC i Powelovog metoda. Pokazao se efektivnijim i od jednog i od drugog metoda ponaosob. Metod se može opisati sledećim glavnim koracima:

Korak 1. Iskoristiti deo *DSK* algoritma, od *Koraka 1* do *Koraka 14*, u kojima se izračunava vrednost $x_{\max}^{(k)}$.

Korak 2. Algoritam produžiti od *Koraka 8* Powelovog metoda, pri čemu skupu $\{x^{(0)}, x^{(1)}, x^{(2)}\}$ odgovara skup $\{x^{(a)}, x^{(m)}, x^{(b)}\}$ iz metoda *DSC*.

Najveći problemi u vezi ovog metoda su izbor početne tačke i početnog koraka. Sledi implementacija ovog metoda u paketu MATHEMATICA.

```
DskPowel[q_,pr_List,x01_,del0_,delmin_]:=
   Block[{x0=x01,q0,q1,q2,q3,x1,x2,x3,del1,dx=10,xkmax,qmax,
      xm,xa,xb,qm,qa,qb,d=del0,it=0,delt=10,p,izb,Lista={}},
     izb=Input["1 za minimum, 2 za maksimum "];
     (* Zajednicki kriterijum za izlaz *)
     While[Abs[dx]>=delmin&&Abs[delt]>=delmin&&it<100,
        (* Deo DSC metoda *)
        q0=N[q/.pr[[1]]->x0];
        Lista=Append[Lista,{x0,q0}];
        x1=x0+d; q1=N[q/.pr[[1]]->x1];
        If[(izb==2 && q0>q1) || (izb==1 && q0<q1),
            d=-d; x2=x0+d; q2=N[q/.pr[[1]]->x2];
            If[(izb==2 && q0>q2) || (izb==1 && q0<q2),
                 xa=x2; xm=x0;  xb=x1,
                 q1=q2; x1=x2
```

```
    ];
];
If[(izb==2&&q1>q0) || (izb==1&&q1<q0),
     del1=2*d;
     x2=x1+del1; q2=N[q/.pr[[1]]->x2];
     While[((izb==2&&q2>q1)||(izb==1&&q2<q1))
            &&(it<100),
           x0=x1;  q0=q1;
           x1=x2;  q1=q2;
           d=del1; del1=2*d;
           x2=x1+del1; q2=N[q/.pr[[1]]->x2];
           it+=1
     ];
     x3=x2-del1/2; q3=N[q/.pr[[1]]->x3];
     If[(izb==2&&q3>q1) || (izb==1&&q3<q1),
         x0=x1; q0=q1; x1=x3; q1=q3,
         x2=x3; q2=q3
     ];
     dx=d/2;
     xm=x1; xa=x0; xb=x2
];
qa=N[q/.pr[[1]]->xa]; qm=N[q/.pr[[1]]->xm];
qb=N[q/.pr[[1]]->xb];
xkmax=xm+dx*(qa-qb)/(2*(qa-2*qm+qb));
qmax=N[q/.pr[[1]]->xkmax];
Lista=Append[Lista,{xkmax,qmax}];
(* Priprema za Powelov metod *)
x0=xa; q0=qa; x1=xm; q1=qm; x2=xb; q2=qb;
(* Deo Powelovog metoda *)
If[izb==1, qxt=Min[q0,q1,q2], qxt=Max[q0,q1,q2] ];
Which[qxt==q0, xt=x0,
      qxt==q1, xt=x1,
      qxt==q2, xt=x2
];
delt=Abs[xkmax-xt];
If[izb==2, qxt=Min[q0,q1,q2], qxt=Max[q0,q1,q2] ];
Which[qxt==q0, x0=xkmax; q0=qmax,
      qxt==q1, x1=xkmax; q1=qmax,
      qxt==q2, x2=xkmax; q2=qmax
```

```
        ];
        If[x1<x0,
              p=x0; x0=x1; x1=p; p=q0; q0=q1; q1=p
        ];
        If[x2<x0,
              p=x0; x0=x2; x2=p; p=q0; q0=q2; q2=p
        ];
        If[x2<x1,
              p=x1; x1=x2; x2=p; p=q1; q1=q2; q2=p
        ];
        it+=1;
        If[it>100, Print["Tacnost = ",delt];
            Return[{xkmax,N[q/.pr[[1]]->xkmax], Lista}]
        ]
    ];
    Print["tacnost= ",delt];
    Return[{xkmax,N[q/.pr[[1]]->xkmax], Lista}];
  ]
```

Test primeri.

In[1]:= Dsk[x^2-1,{x},0.5,0.1,0.0001] (*minimum*)

Out[1]= {0., -1., {{0.5, -0.75},{0., -1.}}}

In[2]:= Dsk[x^2-1,{x},0.5,0.1,0.0001] (*maksimum*)

Out[3]= {1.90148 10^{29} , 3.61561 10^{58} , {{0.5, -0.75}, {1.90148 10^{29} , 3.61561 10^{58} }}}

In[4]:= Dsk[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x}, 0.5,0.1,0.0001] (*minimum*)

Out[4]= {{-0.17267, -0.889502, {{0.5, -0.302649}, {-0.175053, -0.889493},
{-0.175053, -0.889493}, {-0.16737, -0.889455}, {-0.16737, -0.889455},
{-0.125699, -0.885805}, {-0.125699, -0.885805}, {-0.17101, -0.889498},
{-0.17101, -0.889498}, {-0.174296, -0.889498}, {-0.174296, -0.889498},
{-0.172695, -0.889502}, {-0.172695, -0.889502}, {-0.17262, -0.889502},
{-0.17262, -0.889502}, {-0.172317, -0.889502}, {-0.172317, -0.889502},
{-0.17267, -0.889502}}}

In[5]:= Dsk[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x}, 0.5,0.1,0.0001] (*maksimum*)

Out[5]= {2.51459 10^{29} , 3.17081 10^{73},{{0.5,-0.302649},{2.51459 10^{29}, 3.17081 10^{73}}}}

In[6]:= Powel[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x}, 0.5,0.1,0.0001] (*minimum*)

Out[6]= {-0.172658, -0.889502, {{0.5, -0.302649}, {-0.725152, -0.297439},
{-0.118346, -0.884567}, {-0.250773, -0.878896}, {-0.170376, -0.889494},
{-0.172041, -0.889502}, {-0.172681, -0.889502}, {-0.172658, -0.889502}}}

In[7]:= Powel[N[Sin[x-1]]+x^2*N[Sqrt[Abs[x-1]]],{x}, 0.5,0.1,0.0001] (*maksimum*)

Out[7]= {-0.0474527, -0.863848, {{0.5, -0.302649}, {1.84886, 3.89992},

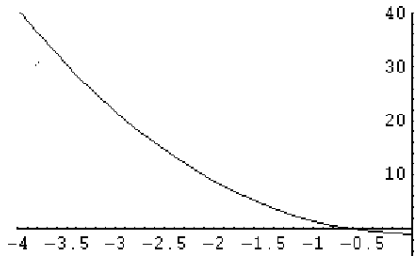
```
{0.374131, -0.475066}, {0.166035, -0.715426}, {0.0329998, -0.822115},
{-0.023233, -0.85325},{-0.0408846, -0.861146}, {-0.0457311, -0.863153},
{-0.0470057, -0.863669},{-0.0473369, -0.863802}, {-0.0474227, -0.863836},
{-0.0474449, -0.863845}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}, {-0.0474527, -0.863848},
{-0.0474527, -0.863848}, {-0.0474527, -0.863848}}}
```

```
In[8]:= Powel[x^2-1,{x},0.5,0.1,0.0001]   (*minimum*)
Out[8]= {-4.33681 10^-16,-1., {{0.5, -0.75}, {-1.15648 10^-15, -1.}, {-4.33681 10^-16, -1.}}}
In[9]:= DskPowel[x^2-1,{x},0.5,0.1,0.0001]   (*minimum*)
Out[9]= {0.,-1., {{-0.75, -0.64},{0., -1.}}}
```

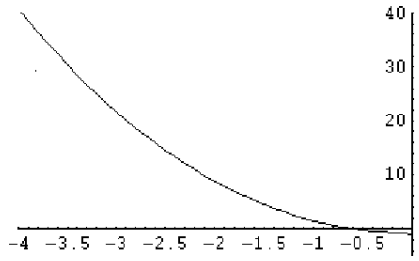
Grafička ilustracija rezultata. Rezultati minimizacija zadatih izrazima

```
skk[N[Sin[x-1]-x*Sqrt[x^2]*N[Sqrt[Abs[x-1]]]+Sqrt[x^2],{x},-4,4,0.1,0.02]
spk[N[Sin[x-1]-x*Sqrt[x^2]*N[Sqrt[Abs[x-1]]]+Sqrt[x^2],{x},-4,4,0.02]
simplexI[N[Sin[x-1]-x*Sqrt[x^2]*N[Sqrt[Abs[x-1]]]+Sqrt[x^2],{x},-4,4,0.1,0.002]
simplexII[N[Sin[x-1]-x*Sqrt[x^2]*N[Sqrt[Abs[x-1]]]+Sqrt[x^2],{x},-4,0.1,0.002]
Powel[N[Sin[x-1]-x*Sqrt[x^2]*N[Sqrt[Abs[x-1]]]+Sqrt[x^2],{x},-4,0.1,0.002]
```

prikazani su, redom, na slikama 1.2.3 – 1.2.7.

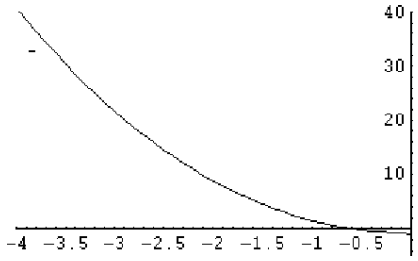


Sl. 1.2.3

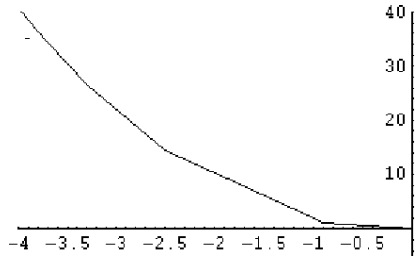


Sl. 1.2.4

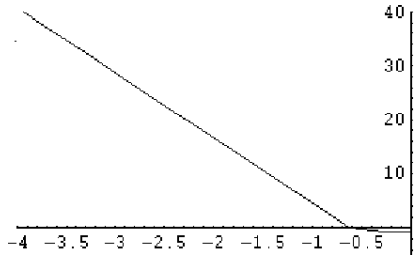
Što je metod "primitivniji," veći je broj izračunavanja vrednosti ciljne funkcije, te je trajektorija kretanja prema ekstremumu sličnija grafiku ciljne funkcije (u ovom primeru paraboli).



Sl. 1.2.5



Sl. 1.2.6



Sl. 1.2.7

1.2.9. METOD PARABOLE

Kod metoda parabole u prvom koraku se izračunavaju tri tačke koje ispunjavaju uslove

$$x_1 < x_2 < x_3, \qquad Q(x_1) > Q(x_2), \qquad Q(x_2) < Q(x_3). \tag{1.2.3}$$

Za nalaženje ovakvih tačaka razrađeno je više algoritama. Najprimitivnija strategija se sastoji u fiksiranju neke vrednosti $x = x_0$ i računanju vrednosti funkcije Q u tačkama

$$x_1 = x_0+h,\ x_2 = x_0+2h,\ x_3 = x_0+3h,\ x_1 = x_2,\ x_2 = x_3,\ x_3 = x_0+4h, \ldots$$

za neku izabranu vrednost koraka h. Posle lokacije ovakvih tačaka, lokalni minimum x^* funkcije Q se nalazi u intervalu $[x_1, x_3]$.

Tri tačke x_1, x_2 i $x3$ sa odgovarajućim vrednostima $Q(x_1)$, $Q(x_2)$, $Q(x_3)$ određuju kvadratnu aproksimaciju, tj. parabolu

$$p(x) = a + bx + cx^2.$$

Koeficijenti a, b, c dobijaju se rešavanjem sistema linearnih algebarskih jednačina

$$\begin{aligned} a + bx_1 + cx_1^2 &= Q(x_1) \\ a + bx_2 + cx_2^2 &= Q(x_2) \\ a + bx_3 + cx_3^2 &= Q(x_3). \end{aligned} \tag{1.2.4}$$

Posle izračunavanja koeficijenata, minimum polinoma $p(x)$ se nalazi kao rešenje jednačine

$$p'(x) = b + 2\,cx = 0,$$

odakle se dobija

$$\tilde{x} = -\frac{b}{2\,c}.$$

Tačka $\tilde{x}$ je aproksimacija optimalne tačke x^*. Sada postoje četiri tačke: "stare" tačke x_1, x_2, x_3 i "nova" tačka $\tilde{x}$. Međutim, za određivanje nove kvadratne aproksimacije potrebne su tri tačke. Od gore navedenih tačaka, u sledećoj iteraciji označimo sa x_2 tačku u kojoj funkcija ima najmanju vrednost (x_2 može biti "stara" tačka x_2). Susednu tačku koja se nalazi levo od "nove" tačke x_2 označimo sa x_1, a susednu tačku desno od "nove" tačke x_2 označimo sa x_3. Za ovako označene tri tačke ponovo važe uslovi (1.2.3), te se postupak može nastaviti.

Teorijski je moguće da je proizvoljno izabrana tačka x_2 na početku procedure upravo tačka minimuma parabole $p(x)$. U tom slučaju, metod parabole ulazi u beskonačnu petlju u kojoj stalno daje $\tilde{x} = x_2$. Da bi se izašlo iz petlje, treba promeniti vrednost x_2 na $x_2 + \varepsilon$, gde je ε neki mali pozitivan broj, na primer, $\varepsilon = 10^{-5}$.

Algoritam ovog metoda se može opisati na sledeći način:

Korak 1. Odrediti proizvoljne tačke $x_1 < x_2 < x_3$ za koje je ispunjeno
$Q(x_1) > Q(x_2), \qquad Q(x_2) < Q(x_3)$.

Korak 2. Odrediti rešenje a, b, c iz sistema (1.2.4).

Korak 3. Izračunati $\tilde{x} = -b/(2c)$, a zatim $p(\tilde{x}) = a + b\tilde{x} + c\tilde{x}^2$ i $Q(\tilde{x})$.

Korak 4. Ako je $|\,Q(\tilde{x}) - p(\tilde{x})\,| \leq \varepsilon$ proces se prekida, a $\tilde{x}$ je približna aproksimacija za x^*.

Ako je $|\,Q(\tilde{x}) - p(\tilde{x})\,| > \varepsilon$ označiti sa x_2 onu od četiri tačke u kojoj funkcija Q ima najmanju vrednost, sa x_1 susednu tačku levo, a sa x_3 susednu tačku desno od "nove" tačke x_2, a zatim preći na *Korak 3*.

1.3. Višedimenzionalna negradijentna optimizacija

U ovom poglavlju se izučava problem izračunavanja optimalne vrednosti ciljne funkcije $Q(\mathbf{x}) = Q(x_1, \dots x_n)$, koja zavisi od n argumenata $x_1, \dots, x_n$, bez korišćenja izvoda ciljne funkcije. Vektor $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ za koji funkcija postiže ekstremnu vrednost u nekoj oblasti $\Gamma_{\mathbf{x}}$ naziva se tačka optimuma i ona se može izračunati različitim metodima.

Metodi višedimenzionalne negradijentne optimizacije mogu se podeliti u sledeće grupe: *metodi skaniranja*, *sukscesivna aproksimacija po koordinatama*, *metodi slučajnog pretraživanja* i *Powelov višedimenzionalni metod*.

A. *Metodi skaniranja.* Zasnivaju se na upoređivanju nekih od izračunatih vrednosti funkcije cilja u određenoj oblasti. Iz ove grupe metoda izdvojićemo:

1. skeniranje sa konstantnim i promenljivim korakom;

2. skeniranje po spirali.

B. *Sukscesivna aproksimacija po koordinatama.* Poseban tip višedimenzionalnog pretraživanja je baziran na promeni jedne od promenljivih u jednom trenutku, pri čemu se sve ostale zadržavaju konstantnim, sve dok se ne pronađe ekstremum nekom od metoda jednodimenzionalne optimizacije. Za svaku promenu nezavisne promenljive, vrednost cijlne funkcije se upoređuje sa vrednošću u prethodnoj iteraciji. Ako je vrednost ciljne funkcije poboljšana u datom koraku, tada nova vrednost ciljne funkcije zamenjuje staru. Međutim, ako je iterativni korak neuspešan, tada ostaju stare vrednosti za funkciju cilja i ekstremnu tačku. Ciklus se završava kada posle pretraživanja po svim koordinatama ne mogu da se dobiju poboljšanja vrednosti ciljne funkcije.

U cilju implementacije metoda višedimenzionalnog pretraživanja, potrebno je da se definiše algoritam transformacije ciljne funkcije $Q(x_1, \dots, x_n)$ u odgovarajuću funkciju parametra x_k $(1 \le k \le n)$:

$$Q(x_1^{(0)}, \dots, x_{k-1}^{(0)}, x_k, x_{k+1}^{(0)}, \dots x_n^{(0)}) = f(x_k),$$

gde je $\mathbf{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})$ dati vektor. Ovaj algoritam je nepodesan za implementaciju u proceduralnim programskim jezicima, ali se može efikasno implementirati u funkcionalnim programskim jezicima. Ovo je suština osobine **(2M)** simboličke implementacije.

U daljem tekstu sledi opis implementacije osobine **(2M)** u LISPu. Unutrašnja forma nove funkcije $f(x_k)$ može se generisati na sledeći način:

Korak 1. Zameniti x_i sa $x_i^{(0)}$ $(1 \leq i \leq n,\ i \neq k)$ u prvom elementu liste koja reprezentuje unutrašnju formu ciljlne funkcije Q. Pri tome se u PC SCHEME mora definisati funkcija $SUBST$ za zamenu svih pojava jednog elementa liste drugim elementom. Ova funkcija može biti napisana koristeći principe iz [12], [14], [67]. LISP izraz konstruisan na taj način predstavlja prvi element rezultujuće unutrašnje forme funkcije

$$Q(x_1^{(0)}, \ldots, x_{k-1}^{(0)}, x_k, x_{k+1}^{(0)}, \ldots x_n^{(0)}) = f(x_k).$$

Korak 2. Lista argumenata funkcije $f(x_k)$ jednaka je listi (x_k). Preciznije, rezultujuća unutrašnja reprezentacija je

$$\big(\big(Q(x_1^{(0)}, \ldots, x_{k-1}^{(0)}, x_k, x_{k+1}^{(0)}, \ldots x_n^{(0)})\big)(x_k)\big).$$

Primer 1.3.1. Neka je funkcija definisana pomoću

$$Q(x_1, x_2) = \frac{x_1^2}{\log(x_2)} + \sqrt{x_2}\,.$$

Za zadatu tačku $(x_1^{(0)}, x_2^{(0)}) = (1, 3)$, ona se može transformisati u funkciju $f(x_1) = Q(x_1, x_2^{(0)})$ transformacijom unutrašnje forme funkcije $Q(x_1, x_2)$:

```
((+ (/ (expt x1 2) (log x2)) (sqrt x2))(x1 x2))
```

u novu listu

```
((+ (/ (expt x1 2) (log 3.0)) (sqrt 3.0))(x1))
```

Odgovarajuća LISP funkcija, definisana za zadatu unutrašnju formu q ciljne funkcije, datu početnu tačku $x0 = (x_1^{(0)}, \ldots x_n^{(0)})$ i zadati ceo broj k, ima oblik:

```
(define (newfun1 q x0 k)
  (let ((f 1) (f1 1) (p 1) (n 1))
     ; Korak 1.  Izdvojiti funkciju f i listu argumenata p
     (set!  f (car q)) (set!  f1 f)
     (set!  n (length (set!  p (cadr q))))
     ; Korak 2.  Za svako i ≠ k zameniti x_i sa x_i^(0)
     ;       u listi koja reprezentuje f.
     (do ((x 1) (i 1))
        ((= i n) (list f (list x)))
```

```
            (if (= i k) (set!  x (car p))
                (begin (set!  f1 (subst (car x0) (car p) f))
                       (set!  f f1)) )
            (set!  x0 (cdr x0)) (set!  p (cdr p))
  ))   )
```

U ovu grupu metoda spadaju:

1. Gauss-Seidelov metod;
2. Metod Hooke-Jeevesa.

U programskoj implementaciji ovih metoda potvrđuje se prednost **(2U)**.

C. *Metodi slučajnog pretraživanja.* Metodi slučajnog pretraživanja zasnivaju se na ispitivanju ciljne funkcije u oblasti upravljačkih parametara na slučajan način. Za primenu ovih metoda nije neophodna čak ni neprekidnost ciljne funkcije.

D. *Powelov višedimenzionalni metod.* Ovaj metod ima svojstvo kvadratnog završavanja i koristi konjugovane pravce.

1.3.1. SKENIRANJE SA KONSTANTNIM I PROMENLJIVIM KORAKOM

Kao i kod funkcija jedne promenljive, *pretraživanje skeniranjem* može se koristiti i za funkcije više promenljivih.

Daćemo opis algoritma za metod skaniranja sa konstantnim korakom za dva upravljačka parametra. Za zadatu vrednost upravljačkog parametra x_2 vrši se skeniranje po upravljačkom parametru x_1, sa definisanim korakom Δx_1 u zadatom untervalu $x_{1\min} \leq x_1 \leq x_{1\max}$, i upamte se koordinate ekstremuma XE_1 i XE_2 kao i vrednost ekstremuma QM. Zatim se parametar x_2 promeni za zadati korak Δx_2, dok se procedura za skeniranje po upravljačkom parametru x_1 ponavlja. skeniranje se završava kada upravljački parametar x_2 dostigne (ili prestigne) gornju granicu $x_{2\max}$.

Algoritam za metod skaniranja sa konstantnim korakom, za slučaj dva upravljačka parametra:

Korak 1. Definisati ulazne veličine: $x_{1\min}$, $x_{2\min}$, $x_{1\max}$, $x_{2\max}$, Δx_1, Δx_2

Korak 2. Postaviti $QM = Q(x_{1\min}, x_{2\min})$, $XE_1 = x_{1\min}$, $XE_2 = x_{2\min}$.

Korak 3. Definisati dvostruku `for` petlju,
za vrednosti promenljive x_1 od $x_{1\min}$ do $x_{1\max}$ sa korakom Δx_1
i za vrednosti promenljive x_2 od $x_{2\min}$ do $x_{2\max}$ sa korakom Δx_2.
Unutar ciklusa izvršiti sledeće algoritamske korake:

Korak 3.1. Izračunati $Q_1 = Q(x_1, x_2)$.

Korak 3.2. Ako je $Q_1 > QM$ postaviti: $QM = Q_1$, $XE_1 = x_1$, $XE_2 = x_2$.

Korak 4. Prikazati vrednosti promenljivih QM, XE_1 i XE_2.

Za proizvoljan broj $n \geq 2$ upravljačkih parametara, vrednost nekog od parametara povećava se za odgovarajući korak kada se završe ciklusi skaniranja za sve prethodne parametre.

Tačnost metoda je veća ukoliko su manji koraci skaniranja Δx_i, $i = 1, \ldots, n$. Međutim, kada se umanjuju koraci Δx_i i povećava broj upravljačkih parametara n, broj izračunavanja vrednosti ciljne funkcije, označen sa S, drastično raste. Uz pretpostavku da je za svaki parametar korak Δ isti, potreban je sledeći broj izračunavanja:

$$S = \left(\frac{1}{\Delta} + 1\right)^n.$$

Naprimer, za $n = 4$ i $\Delta = 0.01$, broj potrebnih izračunavanja ciljne funkcije je 101^4. Zato se ovaj metod koristi u zadacima sa malim brojem upravljačkih parametara ($n = 2, 3, 4$) i sa razumno odabranom tačnošću lokalizacije ekstremuma.

S druge strane, metod je lak za algoritmizaciju. Organizacija pretraživanja ne zavisi od oblika ciljne funkcije. Sa malim korakom Δ i ciljnom funkcijom koja se jednostavno izračunava, ovaj algoritam dozvoljava izračunavanje globalnog ektremuma sa zadatom tačnošću Δ.

Uz pretpostavku da su $x1d$ i $x1g$ granice za prvi argument ciljne funkcije, i da su $x2d$ i $x2g$ granice za njen drugi argument, kao i da su $delx1$ i $delx2$ koraci skaniranja po parametrima $x1$ i $x2$, respektivno, algoritam ove metoda je sledeći:

```
skk2[q_,var_List,x1d_,x1g_,x2d_,x2g_,delx1_,delx2_]:=
    Block[{d1=x1d,d2=x2d,g1=x1g,g2=x2g,
         qmax=q[d1,d2],xe1=d1,xe2=d2,x1,x2,qm,izb,Lista={}},
       izb=Input["1 za min 2 za max"];
       Lista =Append[Lista,{xe1,xe2}];
       For[x1=d1,x1<=g1,x1+=delx1,
          For[x2=d2,x2<=g2,x2+=delx2,
               qm=q;
               qm=N[qm/.var[[1]]->x1];
               qm=N[qm/.var[[2]]->x2];
               If[(izb==2&&qm>qmax)||(izb==1&&qm<qmax),
```

```
                    qmax=qm;xe1=x1;xe2=x2;
                    Lista =Append[Lista,{xe1,xe2}]
                ]
            ]
        ];
        Return[{{xe1,xe2},qmax, Lista}]
    ]
```

Smanjenje broja izračunavanja ciljne funkcije $Q(x)$ može se postići promenljivim korakom skaniranja. Na početku se uzimaju relativno veliki koraci $\Delta_i^{(0)}$, $i = 1, \ldots, n$. Posle lokalizacije ekstremne tačke $\{XE_1, \ldots, XE_n\}$ sa ovako velikim korakom uzima se nova oblast $XE_i \pm \Delta_i^{(0)}$, koja se skanira sa manjim korakom $\Delta_i^{(1)}$, $i = 1, \ldots, n$. Postupak se ponavlja sve dok se ne postigne željena tačnost $\Delta_{\min}$:

$$\Delta_i^{(k)} \leq \Delta_{\min}, \ i = 1, \ldots, n.$$

Kao ilustracija, uzete su test funkcije

$$q1[x] := x^2 + y^2, \quad q2[x] := x^2 + xy - 2x * Sin[x].$$

Za prvu funkciju, poziv

$$skkmax2[q1, \{x, y\}, -2, 2, -2, 2, 0.01, 0.02]$$

je dao maksimum 8 u tački $\{-2, 2\}$. Za drugu funkciju poziv

$$skkmax2[q2, \{x, y\}, -2, 2, -2, 2, 0.01, 0.02]$$

dao je maksimum $4 * Sin[-2]$ u tački $\{-2, 2\}$.

1.3.2. SKENIRANJE PO SPIRALI

Metod se koristi u slučaju kada nisu zadate granice upravljačkih parametara ili kada su zadate samo neke granice upravljačkih parametara. Tada se skeniranje izvršava od početne tačke x_0. Za $n = 2$ može se koristiti ili Arhimedova spirala sa polarnim koordinatama

$$R = a\varphi$$

ili logaritamska spirala

$$R = ae^{m\varphi},$$

pri čemu su a i m konstante.

Ako se pri skaniranju po spirali sa $(2k\pi;\ k = 1, 2, \dots)$ ne dobije bolji rezultat od najboljeg rezultata u prethodnom razvoju $2(k-1)\pi$, tačka sa maksimalnom vrednošću za $Q(x)$ izabira se za novu početnu tačku x_0. Zatim se sprovodi novo skeniranje sa manjim radijusom spirale, sve dok se ne postigne prethodno zadata tačnost $a_{\min} \leq \Delta_{\min}$.

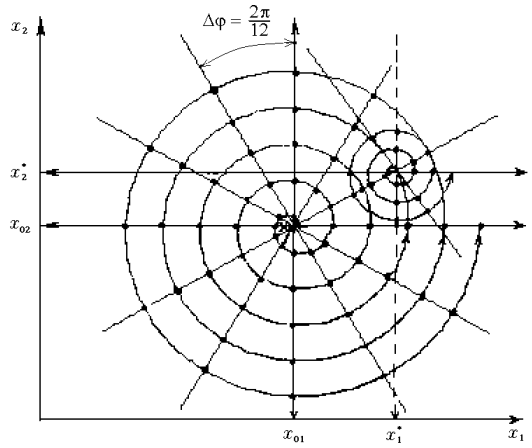
Kako je ciljna funkcija zadata u Dekartovim koordinatama, neophodno je da se u svakoj iteraciji izračunavaju x_1 i x_2 prema formulama

$$x_1 = R\cos\varphi + x_{01}, \quad x_2 = R\sin\varphi + x_{02}.$$

Preporučuje se $\Delta\varphi = \pi/4$ ili $\pi/6$.

Ako početna tačka nije zadata, odabira se na slučajan način.

Na slici 1.3.1 prikazano je skeniranje po Arhimedovoj spirali, za slučaj dva upravljačka parametra.



Sl. 1.3.1

Pri razvoju po logaritamskoj spirali obuhvata se veća površina pri svakom razvoju, za isti radijus razvoja a u odnosu na Arhimedovu spiralu. To znači da se pomoću logaritamske spirale može brže da lokalizuje oblast ekstrema.

```
pospirali[q_,pr_,a_,del_,l_,x01_,x02_]:=
  Block[{c=x01,d=x02,xe1=c,xe2=d,max1=q[xe1,xe2],k=1, fi=0,
       max=max1,a1=a,r,x1,x2,y,xm1,xm2,ind=0,izb, Lista={} },
      izb=Input["1 za min 2 za max= "];
```

```
      max1=q;
      max1=N[max1/.var[[1]]->xe1]; max1=N[max1/.var[[2]]->xe2];
      max=max1;
      Lista=Append[Lista,xe1,xe2];
      fi=fi+Pi/6;
      While[ind=0,
         If[fi>2*k*Pi,
            If[max>max1,
               max1=max;xe1=xm1;xe2=xm2;k=k+1;fi=fi+Pi/6;
               Lista=Append[Lista,xe1,xe2],
               If[a1>del,
                  a1=a1/l;c=xe1;d=xe2;k=1;fi=Pi/6;max=max1,
                  ind=1
               ]
            ],
            r=a1*fi;
            x1=c+r*Cos[fi]; x2=d+r*Sin[fi];
            y=q/.pr->{x1,x2};
            If[(izb==2&& y>max)||(izb==1&& y<max),
               max=y; xm1=x1; xm2=x2
            ];
            fi=fi+Pi/6
         ]
      ];
      {{xe1,xe2}, max, Lista}
]
```

Program se poziva pomoú

$$pospirali[q, pr, a, del, l, x01, x02]$$

pri čemu je q, pr unutrašnja forma ciljne funkcije, a je konstanta, del je zadata tačnost, l faktor smanjenja koraka, a $(x01, x02)$ su koordinate početne tačke.

1.3.3. GAUSS-SEIDELOV METOD

Gauss-Seidelov metod se sastoji u sukscesivnoj izmeni upravljačkih parametara, tj. on predstavlja metod za mnogostruko sukscesivno pretraživanje po jednom upravljačkom parametru.

Algoritam Gauss-Seidelovog metoda se sastoji u sledećem:

Korak 1. Izabrati određeni redosled upravljačkih parametara $x_1, \ldots, x_n$.

Korak 2. Lokalizovati ekstremum x_1^* po prvom upravljačkom parametru x_1, prema jednom od metoda za jednodimenzionalnu optimizaciju, uzimajući konstantne vrednosti $x_{02},\ x_{03}, \ldots, x_{0n}$ za ostale upravljačke parametre.

Korak 3. Nađena vrednost x_1 se uzima za konstantnu i ekstremum se lokalizuje po drugom upravljačkom parametru. Dobijenu vrednost za upravljački parametar označimo sa x_2^*.

Korak 4. Ova procedura se ponavlja sve dok se ne izvrši lokalizacija i poslednjeg upravljačkog parametra x_n.

Korak 5. Kriterijum zaustavljanja je dostizanje tačke $x^* = \{x_1^*, \ldots, x_n^*\}$, od koje pri izmeni od $\pm\Delta_{\min}$ za svaki upravljački parametar ne može se naći bolji rezultat.

U slučaju da izlazni kriterijum nije ispunjen algoritam se nastavlja od Koraka 2.

Kako oblast upravljačkih parametara nije lokalizovana, za lokalizaciju svake koordinate ekstremuma preporučuje se jednodimenzionalni simpleks metod.

Gauss-Seidelov etod se koristi za mali broj upravljačkih parametara ($n = 3, 4$) i ima različitu brzinu konvergencije u zavisnosti od ciljne funkcije $Q(x)$.

Sledi implementacija ovog metoda.

Ulazne veličine:

q_, var_List: ciljna funkcija i lista njenih parametara;
x_List: izabrana početna tačka;
*eps_*: zadata tačnost.

Lokalne promenljive:

del: kriterijum dostignute tačnosti.

```
Gaussei[q_,prom_List,x_List,eps_]:=
    Block[Lis={},rad=True,p,pr=prom,x0=x1=x,
      qm=q0=q,del=1,n=Length[prom],
      Lis=Append[Lis,x0];
      Print["Izabor metoda jednodim.  optim."];
      Print["<1> skeniranje konstantnim korakom"];
      Print["<2> skeniranje promenljivim korakom"];
      Print["<3> simplexI metod"];
      Print["<4> simplexII metod"];
```

```
        Print["<5> zlatni presek"];
        Print["<6> metod dihotomije"];
        Print["<7> DSC metod"];
        Print["<8> Powelov metod"];
        Print["<9> DSC-Powelov metod"];
        metod=Input[];
        While[del>=eps,
             For[i=1,i<=n,i++,
                qm=q;
                Do[qm=qm/.prom[[j]]->x0[[j]],j,i-1];
                qm=qm/.prom[[i]]->p;
                Do[qm=qm/.prom[[j]]->x0[[j]],j,i+1,n];
                (*jednodimenzionalna optimizacija po p*)
                Which[metod==1,p0=skk[qm,{p},0,1,0.01],
                  metod==2,p0=spk[qm,{p},0,1,0.5,eps/10],
                  metod==3,p0=simplexI[qm,{p},0,1,0.5,eps/10],
                  metod==4,p0=simplexII[qm,{p},-1,0.1,eps/10],
                  metod==5,p0=zlatni[qm,{p},-1,1,eps/10],
                  metod==6,p0=dih[qm,{p},-1,1,eps/10],
                  metod==7,p0=Dsk[qm,{p},-1,0.1,eps/10],
                  metod==8,p0=Powel[qm,{p},-1,0.1,eps/10],
                  metod==9,p0=dskpowel[qm,{p},-1,0.1,eps/10]
                ];
                  x0[[i]]=p0[[1]]
             ];
             del=Sqrt[Sum[[x1[[i]]-x0[[i]]]^2,i,n]]
             Lis=Append[Lis,x0];
             x1=x0;
        ];
        Do[q0=q0/.prom[[i]]->x0[[i]],i,n];
        {x0,q0,Lis}
    ]
```

1.3.4. HOOKE-JEEVESOV METOD

Ideja ovog metoda se sastoji u tome da se iz tačke $\mathbf{x}^k$ naprave "koraci" duž svake koordinatne ose:

$$\mathbf{x}^k \pm \mathbf{d}_i^k, \quad i = 1, \ldots, n,$$

gde su "koraci" $\mathbf{d}_i^k$ definisani sa

$$\mathbf{d}_1^k = \begin{bmatrix} \delta \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{d}_2^k = \begin{bmatrix} 0 \\ \delta \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots, \quad \mathbf{d}_n^k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \delta \end{bmatrix}.$$

U prethodnim vektorima δ je neki fiksiran pozitivan broj, k označava redni broj iteracije, dok i označava koordinatu po kojoj se pravi "korak."

Optimizacija počinje od proizvoljne tačke

$$\mathbf{x}_0 = \mathbf{x}_B^0$$

koja se naziva "bazična tačka." Kasnije se izračunavaju nove bazične tačke $\mathbf{x}_B^1, \mathbf{x}_B^2, \ldots$, kojima se aproksimira tačka optimuma $\mathbf{x}^*$. Naziv *bazična* tačka potiče iz činjenice da se oko te tačke prave koraci duž koordinatnih osa, pri čemu ova tačka služi kao "baza." Označimo sa t_i^k "tekuću" promenljivu po kojoj se vrši optimizacija. Na početku je

$$\mathbf{t}_0^1 = \mathbf{x}_B^0.$$

Prvo pretražujemo pozitivnu osu x_1, tj. pravimo korak

$$\mathbf{t}_0^1 + \mathbf{d}_1^0 = \mathbf{t}_0^1 + \begin{bmatrix} \delta \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

i računamo $Q(\mathbf{t}_0^1 + \mathbf{d}_1^0)$. Ako je $Q(\mathbf{t}_0^1 + \mathbf{d}_1^0) \geq Q(\mathbf{t}_0^1)$, ovaj korak nas je doveo do "brda," te u slučaju minimizacije ovaj korak proglašavamo neuspešnim. Sada se pretražuje negativna osa x_1:

$$\mathbf{t}_0^1 - \mathbf{d}_1^0 = \mathbf{t}_0^1 - \begin{bmatrix} \delta \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Ako je ponovo $Q(\mathbf{t}_0^1 - \mathbf{d}_1^0) \geq Q(\mathbf{t}_0^1)$, vraćamo se u $\mathbf{x}_B^0$, stavljamo

$$\mathbf{t}_1^1 = \mathbf{t}_0^1$$

i počinjemo sa pretraživanjem duž ose x_2, tj. pravimo korak

$$\mathbf{t}_1^1 + \mathbf{d}_2^0 = \mathbf{t}_1^1 + \begin{bmatrix} 0 \\ \delta \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Pretpostavimo da je sada $Q(\mathbf{t}_1^1 + \mathbf{d}_2^0) \leq Q(\mathbf{t}_1^1)$. U ovom slučaju, korak je uspešan. Stavimo

$$\mathbf{t}_2^1 = \mathbf{t}_1^1 + \mathbf{d}_2^0.$$

Sada se ne pretražuje negativni deo ose x_2, nego iz $\mathbf{t}_2^1$ odmah pretražujemo treću osu. To pretraživanje ponovo ide najpre u pozitivnom smeru:

$$\mathbf{t}_2^1 + \mathbf{d}_3^0 = \mathbf{t}_2^1 + \begin{bmatrix} 0 \\ 0 \\ \delta \\ \vdots \\ 0 \end{bmatrix}.$$

Ukoliko je ovaj korak bio uspešan, stavljamo

$$\mathbf{t}_3^1 = \mathbf{t}_2^1 + \mathbf{d}_3^0$$

i nastavimo sa koracima duž četvrte ose. Ukoliko je korak neuspešan formira se korak po negativnom delu treće ose. Ako je i ovaj korak bio neuspešan (tj. ako su oba koraka duž treće ose bila neuspešna), stavljamo

$$\mathbf{t}_3^1 = \mathbf{t}_2^1$$

i nastavljamo iz $\mathbf{t}_3^1$ sa pretraživanjem duž četvrte ose.

Potrebno je da se istaknu tri činjenice: Prvo, ako je korak duž pozitivnog dela neke ose uspešan, tada se ne pravi korak duž negativnog dela te ose. Drugo, ako su oba koraka duž jedne koordinatne ose bila neuspešna, tada se za približnu vrednost optimalne tačke uzima vrednost iz prethodne iteracije. Treće, potrebno je odrediti svih n tačaka $\mathbf{t}_1^k, \mathbf{t}_2^k, \ldots, \mathbf{t}_n^k$ između dve

bazične tačke $\mathbf{x}_B^{k-1}$ i $\mathbf{x}_B^k$. Poslednja tačka $\mathbf{t}_n^k$ se uzima za novu bazičnu tačku $\mathbf{x}_B^k = \mathbf{t}_n^k$. Na taj način, prva iteracija se završava posle pretraživanja svih n koordinatnih osa, čime se dobija nova bazična tačka

$$\mathbf{x}_B^1 = \mathbf{t}_n^1.$$

Sada se proces optimizacije može nastaviti iz bazične tačke $\mathbf{x}_B^1$. Međutim, prvo se čini pokušaj da se bazična tačka pomeri u smeru $\mathbf{x}_B^1 - \mathbf{x}_B^0$, čime se dobija tačka

$$\mathbf{t}_0^2 = \mathbf{x}_B^1 + (\mathbf{x}_B^1 - \mathbf{x}_B^0).$$

Za ovakav postupak postoji opravdanje: budući da je vrednost $Q(\mathbf{x}_B^1)$ "bolja" od vrednosti $Q(x_B^0)$, postoji mogućnost da se vrednosti funkcije i dalje poboljšavaju u smeru $\mathbf{x}_B^1 - \mathbf{x}_B^0$ počev od tačke $\mathbf{x}_B^1$. Ukoliko je vrednost $Q(\mathbf{t}_0^2)$ "bolja" od vrednosti $Q(\mathbf{x}_B^1)$ "skok" je bio uspešan. Sada se tačka $\mathbf{t}_0^2$ može proglasiti za novu bazičnu tačku. Međutim, nova bazična tačka se određuje posle prvog uspešnog koraka iz $\mathbf{t}_0^2$. Zbog toga iz te tačke ponovo počinje pretraživanje duž koordinatnih osa. Ako su u nekoj bazičnoj tački $\mathbf{x}_B^k$ svi koraci dužine δ bili neuspešni, tada se dužina δ smanjuje (na primer na polovinu ili desetinu) i koraci se ponavljaju sa ovom manjom dužinom. Ako su svi koraci bili neuspešni, ponovo se smanjuje dužina koraka. Proces se prekida kada dužina koraka bude manja od zadate vrednosti $\varepsilon > 0$. Tada je poslednja bazična tačka željena aproksimacija ekstremne tačke.

Hooke-Jeevesov metod se može ukratko opisati na sledeći način: To je iterativni proces u kome se formira niz bazičnih tačaka $\mathbf{x}_B^0$, $\mathbf{x}_B^1, \ldots$ koje konvergiraju prema nekoj lokalnoj ekstremnoj vrednosti $\mathbf{x}^*$. Počev od svake bazične tačke $\mathbf{x}_B^k$ pretraživanje se nastavlja po svim koordinatnim osama, posle čega se nalazi nova bazična tačka $\mathbf{x}_B^{k+1}$. Tada se pravi "skok"

$$\mathbf{t}_0^{k+2} = \mathbf{x}_B^{k+1} + (\mathbf{x}_B^{k+1} - \mathbf{x}_B^k)$$

i prvi uspešan korak iz $\mathbf{t}_0^{k+2}$ uzima se za novu bazičnu tačku $\mathbf{x}_B^{k+2}$. Ukoliko nijedan od koraka iz $\mathbf{t}_0^{k+2}$ nije uspešan, vraćamo se u $\mathbf{x}_B^{k+1}$. Sada se nastavlja pretraživanje iz nove bazične tačke. Ukoliko nijedan korak iz $\mathbf{x}_B^{k+1}$ nije uspešan, smanjuje se dužina koraka. Proces se prekida kada korak postane manji od zadatog pozitivnog realnog broja.

U funkciji *Postoji* formira se nova "bazična tačka" i ispituje da li je ona "bolja" od prethodne.

```
Postoji[q_,prom_List,t_List,delta_]:=
```

```
   Block[{q0=q,qm,i,j,n=Length[prom],uspesan=False, xm=t},
      Do[q0=q0/.prom[[j]]->xm[[j]],{j,n}];
      For[i=1,i<=n,i++,
          qm=q;
          Do[qm=qm/.prom[[j]]->xm[[j]],{j,i-1}];
          qm=qm/.prom[[i]]->(xm[[i]]+delta);
          Do[qm=qm/.prom[[j]]->xm[[j]],{j,i+1,n}];
          If[qm<q0,
            xm[[i]]+=delta;
            q0=q1;
            uspesan=True,
            qm=q;
            Do[qm=qm/.prom[[j]]->xm[[j]],{j,i-1}];
            qm=qm/.prom[[i]]->(xm[[i]]-delta);
            Do[qm=qm/.prom[[j]]->xm[[j]],{j,i+1,n}];
            If[qm<q0,
              xm[[i]]-=delta;
              q0=q1;
              uspesan=True
            ]
          ]
      ];
      {xm,uspesan}
   ]
```

Sada se Hooke-Jeevesov metod može jednostavno implementirati:

```
HookeJeeves[q_,prom_List,xb_List,delta_,eps_]:=
   Block[{n=Length[prom],x0=x1=xb,t=xb,usp=False,q0=q,
          del=delta,pn,it=0, Lista={} },
      Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
      Lista=Append[Lista,x0];
      (* Formiranje nove bazicne tacke*)
      While[Abs[del]>=eps && !usp,
        pn=Postoji[q,prom,t,del];
        usp=pn[[2]];
        Print["usp = ",usp];
        If[usp, x1=pn[[1]];Lista=Append[Lista,x1], del/=2]
      ];
      While[Abs[del]>=eps,
```

```
        (*Skok u novu tacku t*)
        t=x1+(x1-x0);
        Print[" t = ",t];
        pn=Postoji[q,prom,t,del];
        usp=pn[[2]];
        If[usp, x0=x1;x1=pn[[1]];Lista=Append[Lista,x1],
            del/=2
        ];
        Print["x1= ",x1];
        q0=q; Do[q0=q0/.prom[[i]]->x1[[i]],{i,n}];
        Print["q=",q0,"delta=",del];
        it+=1
    ];
    q0=q; Do[q0=q0/.prom[[i]]->x1[[i]],{i,n}];
    Print["it= ",it];
    {x1,q0, Lista}
]
```

Izložićemo sada jedno poboljšanje Hooke-Jeevesove metode iz [55], koje se sastoji u tome da se nova tačka t_0^{k+2} izračunava pomoću

$$\mathbf{t}_0^{k+2} = \mathbf{x}_B^{k+1} + h_k(\mathbf{x}_B^{k+1} - \mathbf{x}_B^k),$$

gde je korak h_k definisan pomoću

$$h_k = \min_h Q\left(\mathbf{x}_B^{k+1} + h(\mathbf{x}_B^{k+1} - \mathbf{x}_B^k)\right) = \min_h F(h)$$

i izračunava se nekom od metoda jednodimenzionalne optimizacije.

Najvažniji detalj u implementaciji modifikacije Hooke-Jeevesovog metoda jeste konstrukcija funkcije

$$F(h) = Q\left(x_B^{k+1} + h(\mathbf{x}_B^{k+1} - \mathbf{x}_B^k)\right)$$

u simboličkoj formi. Pri tome je zadat analitički izraz $q\_$ ciljne funkcije Q i postoje izračunate dve bazične tačke $x1 = \mathbf{x}_B^{k+1}$ i $x0 = \mathbf{x}_B^k$. Ovaj problem je nepodesan za implementaciju u proceduralnim programskim jezicima. Po svojoj suštini, on je podesniji za simboličku implementaciju ako se h posmatra kao simbol. U programskom paketu MATHEMATICA, funkcija $F(h) = qexp$ se može jednostavno implementirati:

```
t=x1+h*(x1-x0);
```

```
qexp=q;
Do[qexp=qexp/.prom[[i]]->t[[i]],{i,n}];
```

Sada je unutrašnja forma ciljne funkcije za iniciranu jednodimenzionalnu optimizaciju jednaka

$$qexp, \quad \{h\}.$$

Preostali argumenti koji se koriste u pozivu funkcija za jednodimenzionalnu optimizaciju zavise od metoda optimizacije. Potrebno je naglasiti da se koriste nenegativne vrednosti parametra h, što znači da je njegova donja granica jednaka nuli. Ukoliko se u metodu jednodimenzionalne optimizacije koristi jedna početna vrednost, prirodno je da ona bude nula. Nešto je složeniji problem određivanja gornje granice parametra h. U programima koji su navedeni korišćena je vrednost 1, kada se modifikacija svodi na originalni metod. Međutim, mogu se koristiti i vrednosti veće od jedinice. Tada jednodimenzionalna optimizacija ima veći efekat i koristi se manji broj puta.

Osim prednosti **(1M)**–**(3M)** simboličke implementacije metoda višedimenzionalnog pretraživanja, može se navesti i sledeća prednost:

(4M) Funkcionalni programski jezici su pogodniji za implementaciju sledeće transformacije

$$Q(x_1, \ldots, x_n) \mapsto F(h) = R(Q(x_1, \ldots, x_n),\ h),$$

gde je R proizvoljna funkcija.

```
HookeJeevesh[q_,prom_List,xb_List,delta_,eps_]:=
   Block[{n=Length[prom],x0=x1=xb,t=xb,usp=False,q0=q,
          del=delta,pn, qexp,hk,metod,it=0},
      Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
      Lista=Append[Lista,x0];
      (* Formiranje nove bazicne tacke*)
      While[del>=eps && !usp,
          pn=Postoji[q,prom,t,del];
          usp=pn[[2]];
          Print["usp = ",usp];
          If[usp, x1=pn[[1]];Lista=Append[Lista,x1], del/=2]
      ];
      (*Izbor jednodimenzione optimizacije*)
      Print["Izaberi metod jednoimenzionalne optimizacije"];
      Print["<1> skeniranje konstantnim korakom"];
      Print["<2> skeniranje promenljivim korakom"];
```

```
Print["<3> simplexI metod"];
Print["<4> simplexII metod"];
Print["<5> zlatni presek"];
Print["<6> metod dihotomije"];
Print["<7> DSC metod"];
Print["<8> Powelov metod"];
Print["<9> DSC-Powelov metod"];
metod=Input[];
While[Abs[del]>=eps,
  (* Simbilicko formiranje nove tacke t *)
  t=x1+h*(x1-x0);
  qexp=q;
  Do[qexp=qexp/.prom[[i]]->t[[i]],{i,n}];
  (*qexp je funkcija simbola h*)
  Print["qexp = ",qexp];
  (* jednodimenziona optimizacija po h *)
  Which[metod==1,hk=skk[qexp,{h},0,1,0.01],
        metod==2,hk=spk[qexp,{h},0,1,0.5,eps/10],
        metod==3,hk=simplexI[qexp,{h},0,1,0.5,eps/10],
        metod==4,hk=simplexII[qexp,{h},0,0.1,eps/10],
        metod==5,hk=zlatni[qexp,{h},0,1,eps/10],
        metod==6,hk=dih[qexp,{h},0,1,eps/10],
        metod==7,hk=Dsk[qexp,{h},0.,0.1,eps/10],
        metod==8,hk=Powel[qexp,{h},0.,0.1,eps/10],
        metod==9,hk=dskpowel[qexp,{h},0.,0.1,eps/10]
  ];
  hk=hk[[1]]; Print["hk= ",hk];
  (*Formiranje tacke t prema optimalnoj vrednosti h*)
  t=x1+hk*(x1-x0);
  Print["t posle minimiziranja po h = ",t];
  pn=Postoji[q,prom,t,del];
  usp=pn[[2]];
  If[usp, x0=x1;x1=pn[[1]];Lista=Append[Lista,x1],
       del/=2
  ];
  Print["x1= ",x1];
  q0=q; Do[q0=q0/.prom[[i]]->x1[[i]],{i,n}];
  Print["q=",q0,"delta=",del];
  it+=1
```

```
    ];
    q0=q; Do[q0=q0/.prom[[i]]->x1[[i]],{i,n}];
    Print["it= ",it];
    {x1,q0, Lista}
]
```

Numerički rezultati. Ilustrovaćemo metod kroz nekoliko primera.

Primer 1. Ako se za minimizaciju funkcije $Q(x,y) = x^2 + y^2 - 3\,Sin[x-y]$ koristi originalni Hooke-Jeevesov metod, sa početnom vrednošću minimuma $\{0.5, 0.8\}$, primećuje se njegova divergencija:

In[1]:= HookeJeeves[x^2+y^2-3*Sin[x-y],{x,y},{0.5,0.8},0.1,0.001]

Deo liste generisanih tačaka je:

{{0.8, 0.8},{1.1, 0.8},{1.3, 0.8},{1.4, 0.8},{1.4, 0.8}, {1.3, 0.8}, {1.3, 0.7}, {1.2, 0.6}, {1.2, 0.5},{1.1, 0.4}, {1.1, 0.3},{1., 0.2},{1., 0.1},{0.9, 2.22045×10^-16}, {0.9, -0.1}, ... }

Out[1]= $Aborted

Za iste početne uslove modifikacija Hooke-Jeevesovog metoda konvergira.

Na primer, u startnoj tački, simbolička funkcija $f(h)$ jednaka je

```
                  2                  2
f = (0.8 + 0. h)   +(0.6 + 0.1 h)    + 3 Sin[0.2 - 0.1 h]
```

Metod zlatnog preseka posle minimizacije $h_k = \min_h f(h)$ daje rezultat $hk = 0.999959$,

što implicira

$x1 = \{0.799996, 0.8\}, \quad q = 1.28001, \quad delta = 0.1$

U 22. iteraciji se dobija simbolička funkcija

```
                                  -12    2                                  2
f = (-0.584927 - 5.03952 10     h)  + (0.585755 + 0.00312474 h)    -
> 3 Sin[1.17068 + 0.00312474 h]
```

Jednodimenzionalnom optimizacijom dobija se korak $hk = 0.0000408563$

i rezultat:

$x1 = \{0.585755, -0.584927\}, \quad q = -2.0778, \quad delta = 0.00078125.$

Primer 2. U ovom primeru je dato nekoliko grafičkih ilustracija modifikovanog metoda, koje su dobijene primenom funkcija *ContourPlot* i *ListPlot* u paketu MATHEMATICA.

Rezultujuća lista dobijena izrazom

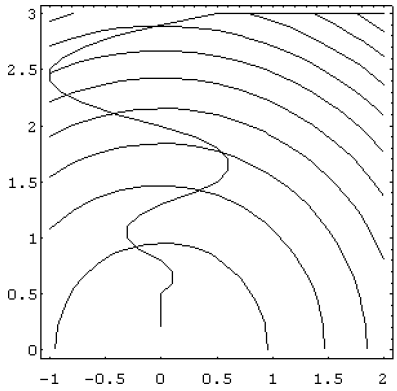
```
HookeJeeves[x^2+y^2,{x,y},{2,3},0.1,0.00000000001]
```

reprezentovana je na slici 1.3.2.

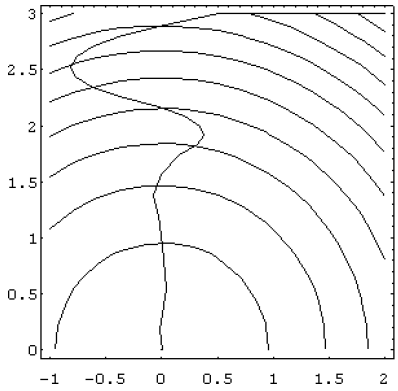
Rezultati modifikovanog Hooke-Jeevesovog metoda

```
HookeJeevesh[x^2+y^2,{x,y},{2,3},0.1,0.00000000001]
```

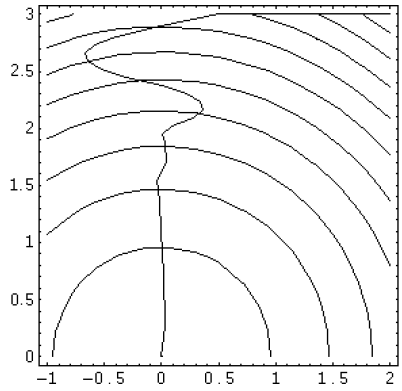
poboljšavaju brzinu konvergencije i tačnost. Na slici 1.3.3 prikazana je trajektorija generisana fiksnim korakom pretraživanja sa tačnošću 10^{-11}.



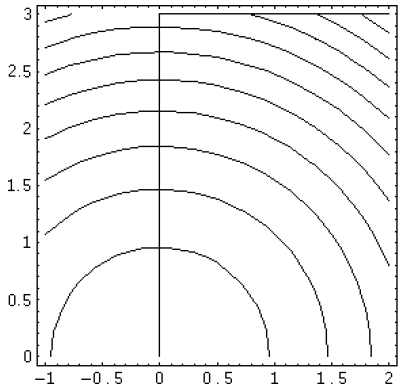
Sl. 1.3.2



Sl. 1.3.3



Sl. 1.3.4



Sl. 1.3.5

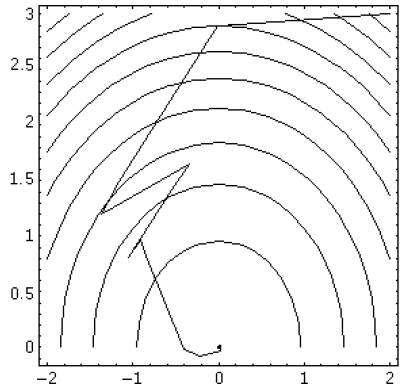
Na slici 1.3.4 prikazana je trajektorija generisana promenljivim korakom pretraživanja sa tačnošću 10^{-11}, dok je na slici 1.3.5 prikazana trajektorija generisana prvom varijantom simpleks metoda sa tačnošću 10^{-11}.

Na slici 1.3.6 prikazana je trajektorija generisana drugom varijantom simpleks metoda sa tačnošću 10^{-7}, dok je na slici 1.3.7 prikazana trajektorija generisana metodom zlatnog preseka sa tačnošću 10^{-7}.

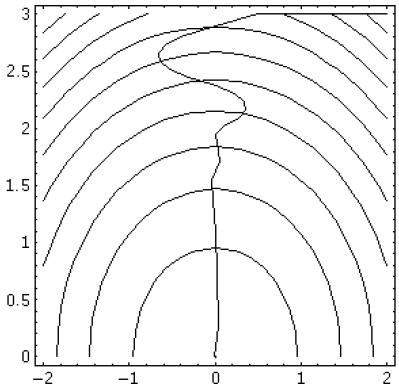
Trajektorije generisane metodom dihotomije sa tačnošću 10^{-7} i Powelovim metodom sa istom tačnošću prikazane su na slikama 1.3.8 i 1.3.9, respektivno.

Primer 3. Sledeće izračunavanje, definisano originalnim Hooke-Jeeves metodom

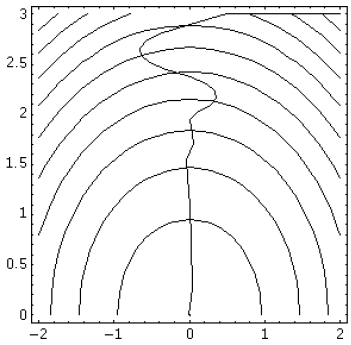
```
HookeJeeves[x^2+y^2,{x,y},{20,10},0.1,0.00000000001]
```



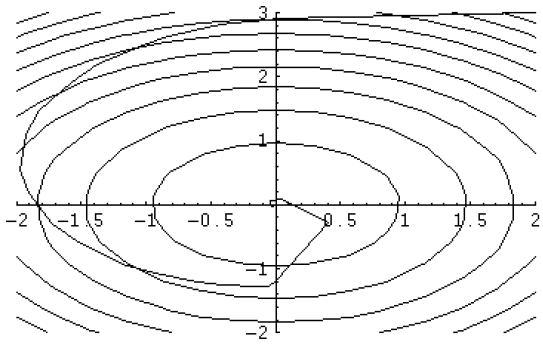
Sl. 1.3.6



Sl. 1.3.7



Sl. 1.3.8



Sl. 1.3.9

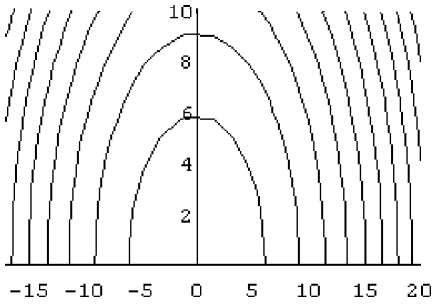
divergira (slika 1.3.10), dok modifikovani metod, pod istim pretpostavkama, konvergira (slika 1.3.11).

1.3.5. SLUČAJNO PRETRAŽIVANJE

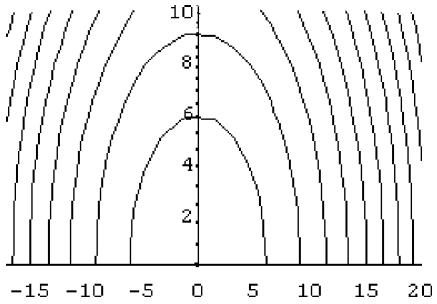
Ovaj metod je analogan metodu skaniranja. Pri primeni metoda prostog slučajnog pretraživanja, vrednosti ciljne funkcije se izračunavaju u sekvencijalno generisanim tačkama $\mathbf{x}$, koje su ravnomerno raspoređene unutar dozvoljene oblasti

$$[\mathbf{xmin},\ \mathbf{xmax}] = [xmin_i, xmax_i], \quad i = 1, \ldots, n.$$

Pretraživanje se prekida na osnovu jednog od sledeća dva kriterijuma:



Sl. 1.3.10



Sl. 1.3.11

1° Dostizanje unapred zadatog broja M_1 izračunavanja vrednosti $Q(\mathbf{x})$;

2° Dostizanje zadatog broja izračunavanja M_2 ciljne funkcije $Q(\mathbf{x})$ nakon poslednjeg izbora tekućeg maksimuma (M_2 je broj "neuspešnih tačaka"). Obično se uzima $10n \leq M_2 \leq 50n$).

Slučajna tačka unutar dozvoljene oblasti $[xmin_i, xmax_i]$, $i = 1, \ldots, n$ generiše se jednakošću

$$x_i = xmin_i + \alpha_i(xmax_i - xmin_i), \quad i = 1, \ldots, n, \tag{1.3.1}$$

gde je α_i pseudoslučajni broj u intervalu $[0, 1]$ dobijen od strane programskog generatora.

Algoritam slučajnog pretraživanja:

Korak 1. Zadati ulazne veličine n, M_1, $xmin_i$, $xmax_i$, $i = 1, \ldots, n$.

Korak 2. Generisati niz slučajnih brojeva α_i i niz koordinata $x_i, i = 1, \ldots, n$, tačke $\mathbf{x}$, na osnovu (1.3.1).

Korak 3. Postaviti $J = 1$ i $xE_i = x_i$, $i = 1, \ldots, n$.

Korak 4. Izračunati $Q_1 = QM = Q(\mathbf{x})$.

Korak 5. Unutar ciklusa, koji se prekida kada je $J > M_1$, izvršiti sledeće korake:

Korak 5.1. Generisati niz slučajnih brojeva α_i a zatim niz slučajnih brojeva x_i, $i = 1, \ldots, n$, koristeći (1.3.1).

Korak 5.2. Izračunati $Q_1 = Q(\mathbf{x})$.

Korak 5.3. Ako je $Q_1 > QM$ postaviti
$QM = Q_1$, $xE_i = x_i$, $i = 1, \ldots, n$.

Korak 5.4. Postaviti $J := J + 1$.

Korak 6. Editovati vrednosti promenljivih QM i xE_i, $i=1,\dots,n$.

U sledećem algoritmu, pretraživanje se prekida kada se dostigne zadati broj M_2 izračunavanja vrednosti ciljne funkcije nakon poslednjeg definisanja tekućeg maksimuma.

Korak 1. Zadati ulazne veličine N, M_2, $xmin_i$, $xmax_i$, $i=1,\dots,n$.

Korak 2. Generisati niz slučajnih brojeva α_i i niz koordinata x_i, $i=1,\dots,n$, tačke $\mathbf{x}$, prema (1.3.1).

Korak 3. Postaviti $L=0$ i $xE_i=x_i, \quad i=1,\dots,n$.

Korak 4. Izračunati $Q_1=QM=Q(\mathbf{x})$.

Korak 5. Unutar ciklusa, koji se prekida kada je $L>M_2$, izvršiti sledeće korake:

Korak 5.1. Koristeći (1.3.1) generisati niz slučajnih brojeva α_i, a zatim niz slučajnih brojeva $x_i, i=1,\dots,n$.

Korak 5.2. Izračunati $Q_1=Q(\mathbf{x})$.

Korak 5.3. Ako je $Q_1>QM$ postaviti
$QM=Q_1$, $xE_i=x_i$, $i=1,\dots,n$, i $L=0$; inače postaviti
$L:=L+1$.

Korak 6. Prikazati vrednosti promenljivih QM i xE_i, $i=1,\dots,n$.

Približna tačnost lokalizacije ekstremuma Δ zavisno od broja ravnomerno generisanih tačaka M_1, izračunava se po formuli

$$\Delta \approx \sqrt{\frac{1}{M_1}}.$$

Pri korišćenju ovog metoda verovatnoća da se pronađe globalni maksimum raste u zavisnosti od broja izračunavanja vrednosti ciljne funkcije.

Metod nije efikasan, ali se koristi kao pomoćni algoritam za nalaženje globalnog ekstremuma ili za izbor početne optimalne tačke pri implementaciji drugih metoda.

1.3.6. SLUČAJNO PRETRAŽIVANJE SA VEĆOM GUSTINOM

U ovom algoritmu nova slučajna tačka se generiše u zavisnosti od tekuće najbolje vrednosti ciljne funkcije. Kriterijum za prekid algoritma je dostizanje zadatog broja M "neuspešnih" tačaka nakon poslednjeg izbora najboljeg rezultata za $Q(x)$.

U toku primene ovog metoda moguće je da vrednost nekog od upravljačkih parametara izađe izvan dozvoljene oblasti $\Gamma_{\mathbf{x}}$. Zbog toga je neophodno izvršiti proveru da li su vrednosti svih parametara unutar zadate oblasti $[\mathbf{xmin}, \mathbf{xmax}]$. Blok šema algoritma *limit*, kojim se vrednosti upravljačkih parametara vraćaju unutar zadate oblasti, posebno je opisana u ovom odeljku.

U algoritmu vrednost broja k predstavlja broj novoodabranih ocena maksimuma. Nova slučajna tačka $\mathbf{x1} = x1_i,\ i = 1, \dots, n$, se generiše prema formulama

$$x1_i = xE_i + (xmax_i - xmin_i)\frac{(2\alpha_i - 1)^k}{k}, \quad i = 1, \dots, n. \tag{1.3.2}$$

Zbog $-1 \leq 2\alpha_i - 1 \leq 1$, zaključujemo da se nove slučajne tačke generišu sa sve većom gustinom sa povećanjem vrednosti k.

Ovaj metod ima karakter adaptivnog metoda, ali sa veoma niskim stepenom adaptivnosti. Sledi opis ovog metoda.

Korak 1. Zadati ulazne veličine $n, M_1,\ xmin_i,\ xmax_i, \quad i = 1, \dots, n$.

Korak 2. Postaviti $L = 0,\ K = 1$.

Korak 3. Generisati niz slučajnih brojeva α_i, a zatim niz brojeva

$$xE_i = xmin_i + \alpha_i(xmax_i - xmin_i), \quad i = 1, \dots, n,$$

koji predstavljaju koordinate tačke $\mathbf{xE}$.

Korak 4. Izračunati $QM = Q(\mathbf{xE})$.

Korak 5. Unutar ciklusa, koji se prekida kada je $J > M_1$, izvršiti sledeće:

Korak 5.1. Generisati niz slučajnih brojeva α_i a zatim niz brojeva $x1_i$, $i = 1, \dots, n$, prema (1.3.2).

Korak 5.2. Poziv procedure *limit*.

Korak 5.3. Izračunati $Q_1 = Q(\mathbf{x1})$.

Korak 5.4. Ako je $Q_1 > QM$ postaviti:
$QM = Q_1,\ xE_i = x1_i,\ i = 1, \dots, n,\ k = k + 1,\ L = 0$;
inače postaviti $L := L + 1$.

Korak 6. Editovati vrednosti QM i xE_i, $i = 1, \dots, n$.

Sledi opis algoritma procedure *limit*.

Ulazne veličine su $n,\ xmin(I),\ xmax(I),\ I = 1, \dots, n$.

Korak 1. Definisati `for` ciklus, za vrednosti promenljive $I = 1, \ldots, n$. Unutar ciklusa uraditi sledeće korake:

Korak 1.1. Ako je $x(I) > xmax(I)$ postaviti $x(I) = xmax(I)$.

Korak 1.2. Ako je $x(I) < xmin(I)$ postaviti $x(I) = xmin(I)$.

```
limit[x_List,xmin_List,xmax_List]:=
  Block[{i,n,x0=x},
      n=Length[x0];
      Do[Which[x0[[i]]<xmin[[i]],x0[[i]]=xmin[[i]],
               x0[[i]]>xmax[[i]],x0[[i]]=xmax[[i]]
           ],{i,n}
      ];
      x0
  ]
```

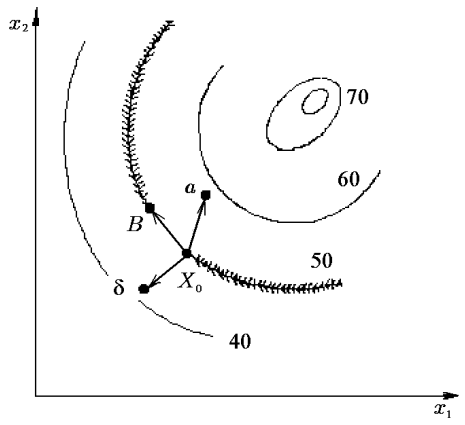
1.3.7. METOD SLUČAJNIH SMEROVA

Osnovna ideja ovog metoda sastoji se u sledećem. Neka je zadata tačka $\mathbf{x}_0 = (x_1^{(0)}, \ldots, x_n^{(0)})$ u dozvoljenoj oblasti Γ_x. Ova tačka leži na jednoj površi (ili liniji) sa konstantnomm vrednošću ciljne funkcije. Ova površ deli prostor na dve oblasti, koje se nazivaju *uspešna* i *neuspešna*, zavisno od smera koji vodi prema ekstremu. Ako se načini neki korak od tačke $\mathbf{x}_0$ u slučajnom smeru postoje tri mogućnosti:

a) korak je unutar *uspešne* oblasti, tj. u smeru poboljšanja vrednosti $Q(\mathbf{x})$;
b) korak je u *neuspešnoj* oblasti, tj. u smeru lošijih vrednosti funkcije $Q(\mathbf{x})$;
c) korak je unutar *neutralne* oblasti, tj. $Q(\mathbf{x})$ ima istu vrednost.

Ako je korak u *uspešnoj* oblasti, nova tačka može da se uzme za novu početnu tačku, a zatim da se načini novi korak u slučajnom pravcu. Ako je tačka u *neuspešnoj* oblasti, ona se izostavlja i pravi se novi korak od prethodne početne tačke, sve dok se ne dobije nova uspešna tačka. Praveći uzastopno korake konstantne dužine Δx na slučajan način i zamenjujući staru početnu tačku novom uspešnom tačkom, stiže se u oblast ekstremne tačke $\mathbf{x}^*$. U toj oblasti korak može da se smanjuje do prethodno zadatih granica. Grafički je metod ilustrovan na slici 1.3.12.

Na osnovu ove ideje razrađeno je mnogo algoritama za slučajno pretraživanje. Osnovna ideja tih algoritama je formiranje koraka u slučajnom pravcu. To se može ostvariti formiranjem slučajnog vektora $\xi = (\xi_1, \ldots, \xi_n)$ jedinične dužine, koji se prostire od zadate tačke u svim mogućim pravcima



Sl. 1.3.12

sa jednakom verovatnoćom, unutar oblasti upravljačkih parametara. Korak od zadate tačke $\mathbf{x}^{(k)}$ u novu tačku $\mathbf{x}^{(k+1)}$ određen je transformacijom

$$x_i^{(k+1)} = x_i^{(k)} + h_i \xi_i^{(k)}, \quad i = 1, 2, \dots, n,$$

gde je h_i veličina koraka po svakom upravljačkom parametru, a $\xi_i^{(k)}$ je i-ta komponenta slučajnog vektora formiranog u k-oj iteraciji. Kriterijum uspešnosti koraka je $Q(\mathbf{x}^{(k+1)}) > Q(\mathbf{x}^{(k)})$ u slučaju maksimuma i $Q(\mathbf{x}^{(k+1)}) < Q(\mathbf{x}^{(k)})$ za slučaj minimuma. U blizini ekstremne tačke, potrebno je sve više smanjivati dužinu koraka h_i da bi korak bio uspešan. Kriterijum za prekid pretraživanja može da bude dostizanje broja M neuspešnih smerova nakon poslednjeg uspešnog koraka. Broj M može da se uzme prema empirijskim formulama

$$M = \begin{cases} 2^n + 4, & \text{za } n \leq 3, \\ 2n + 4, & \text{za } n > 3. \end{cases} \tag{1.3.3}$$

Formiranje slučajnog vektora. Komponente slučajnog n-dimenzionalnog vektora $\xi = (\xi_1, \dots, \xi_n)$ mogu se dobiti korišćenjem vrednosti n slučajnih brojeva β_i $(i = 1, \dots, n)$ ravnomerno raspoređenih unutar intervala $[-C, C]$, prema sledećoj formuli

$$\xi_i = \frac{\beta_i}{\sqrt{\sum_{i=1}^{n} \beta_i^2}}, \qquad i = 1, \dots, n.$$

Pri tome je, evidentno, ispunjen uslov

$$\sum_{i=1}^{n} \xi_i^2 = 1.$$

Ako se raspolaže nizom slučajnih brojeva α_i koji su ravnomerno raspoređeni unutar pozitivnog intervala $[0, C]$, on se transformiše u interval $[-C, C]$ prema formuli

$$\beta_i = \left(\alpha_i - \frac{C}{2}\right) \cdot 2, \quad i = 1, 2, \ldots, n.$$

Na primer, ako je α_i u intervalu $[0, 1]$, on se transformiše u interval $[-1, 1]$, prema formuli

$$\beta_i = \left(\alpha_i - \frac{1}{2}\right) \cdot 2, \qquad i = 1, 2, \ldots, n,$$

posle čega se izračunavaju vrednosti ξ_i. Algoritam za generisanje komponenti n-dimenzionalnog slučajnog jediničnog vektora ξ implementiran je sledećim programom:

```
Sluvec[n_Integer]:=
  Block[{v=Table[Random[],{n}],s},
      s=Sum[v[[i]]^2,{i,n}];
      s=Sqrt[s];
      Do[v[[i]]/=s, {i,n}];
      v
  ]
```

Sledi opis implementacije metoda slučajnih smerova.

Neophodni parametri algoritma su:

$Q(\mathbf{x}) = Q(x_1, \ldots, x_n)$: ciljna funkcija;

n: broj upravljačkih parametara;

$\mathbf{x0} = x0_i$, $i = 1, \ldots, n$: početna tačka;

$\mathbf{h0} = h0_i$, $i = 1, \ldots, n$: početni parametar koraka;

$\mathbf{hmin} = hmin_i$, $i = 1, \ldots, n$: minimalan parametar koraka, odnosno tačnost lokalizacije ekstremuma za svaki upravljački parametar;

LH: vrednost kojom se skraćuju koraci optimizacije h_i, $i = 1, \ldots, n$;

xmin $= xmin_i,\ i = 1, \ldots, n,$ i **xmax** $= xmax_i,\ i = 1, \ldots, n$: vektori kojima je određena oblast optimizacije.

Neophodni potprogrami su:

potprogram za generisanje slučajnog vektora *Sluvec*;

potprogram za proveru ograničenja *limit*.

Pored toga, u algoritmu se koriste sledeće lokalne promenljive:

x1: nova slučajna tačka u pravcu slučajnog vektora;

l-brojač neuspešnih pokušaja ($l \leq M$);

$q0$ i $q1$: pomoćne promenljive (dodeljuje im se vrednost funkcije q u dve uzastopne iteracije **x0** i tački **x1**, respektivno).

Algoritam metoda je sledeći:

Korak 1. Definisati broj M saglasno formuli (1.3.3).

Korak 2. Izračunati vrednost ciljne funkcije u početnoj tački $Q_0 = Q(x_0)$.

Korak 3. Broj neuspešnih smerova postaje nula ($L = 0$).

Korak 4. Generisati komponente slučajnog vektora ξ, koristeći potprogram *Sluvec*.

Korak 5. Načiniti korak u slučajnom pravcu, tj. izračunati **x1** prema

$$x_i^{(k+1)} = x_i^{(k)} + h_i \xi_i^{(k)}, \qquad i = 1, \ldots, n.$$

Korak 6. Proveriti ograničenja za interval (**xmin**, **xmax**) koristeći potprogram *limit*.

Korak 7. Izračunati se vrednost ciljne funkcije $Q_1 = Q(\mathbf{x1})$.

Korak 8. Proveriti da li je načinjeni korak uspešan, tj. uslov $Q_1 > Q_0$.

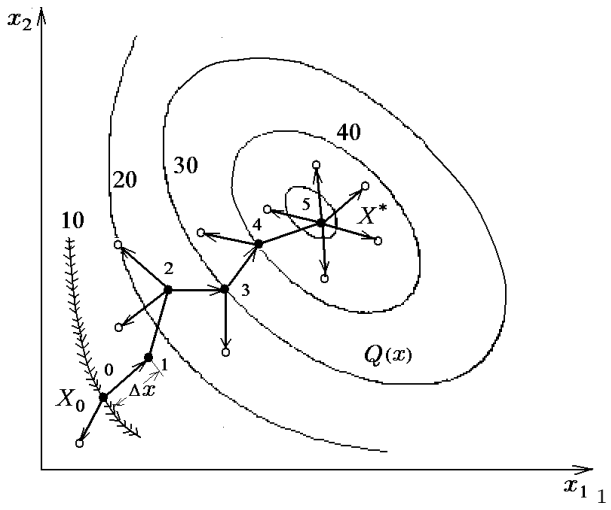
Korak 9. Ako je $Q_1 > Q_0$ početnu tačku premestiti u **x1**, tj. postaviti $\mathbf{x0} = \mathbf{x1}$ i nastaviti algoritam od *Koraka 3*.

Korak 10. Ako je $Q_1 \leq Q_0$ postaviti $L := L+1$, tj. povećati broj neuspešnih smerova za 1.

Korak 11. Proveriti broj neuspešnih smerova. Ako je $L \leq M$, produžiti algoritam od *Koraka 4*, tj. izabrati novi slučajni smer.

Korak 12. Ako je $L > M$, proveriti da li je parametar koraka za svako x_i dostigao zadatu tačnost $hmin_i$, tj.

$$h0_i \leq hmin_i \quad \text{za svako} \quad i = 1, \ldots, n.$$



Sl. 1.3.13

Ako je uslov ispunjen prekinuti pretraživanje i editovati vrednosti za Q_0 i $\mathbf{x0}$.

Korak 13. Ako poslednji uslov nije ispunjen, smanjiti svaki korak h_i deljenjem sa LH, a zatim nastaviti algoritam od *Koraka 3*.

Metod je ilustrovan na slici 1.3.13.

```
SluDir[q_,var_List,x_List,h_List,hmin_List,elha_,
                                    xmin_List,xmax_List]:=
  Block [{x0=x,ksi,M,q0,q1,n,i,L=0,x1=x,lg=True,h0=h,lh=elha,
          izbor, Lista={} },
      izbor=Input["Zelite li minimum(1) ili maksimum(2)?" ];
      n=Length[var];
      If[n<=3, M=2^n+4, M=2*n+4];
      q0=q; Do[q0=q0/.var[[i]]->x0[[i]],{i,n}];
      Lista=Append[Lista,x0];
      q1=q0;
      While[L<=M && lg,
            ksi=Sluvec[n]; ksi=Limit[ksi,xmin,xmax];
            Do[x1[[i]]=x0[[i]]+h0[[i]]*ksi[[i]],{i,n}];
            x1=Limit[x1,xmin,xmax];
            q1=q; Do[q1=q1/.var[[i]]->x1[[i]],{i,n}];
            If[(izbor==1 && q1<q0) || (izbor==2 && q1>q0),
                  q0=q1; x0=x1; L=0; Lista=Append[Lista,x0],
```

```
                L=L+1;
                lg=False;
                Do[Which[h0[[i]]>hmin[[i]], lg=True],{i,n}];
                If[lg, Do[h0[[i]]/=lh,{i,n}]; ]
            ]
        ];
        {x0,q0, Lista}
    ]
```

Metod slučajnih smerova nije tako efikasan u odnosu na brzinu konvergencije, ali je on osnova za niz drugih metoda, kao što su algoritam sa obrnutim korakom, algoritam sa linearnom interpolacijom za ciljnu funkciju, algoritam slučajnog traženja sa samoobučavanjem, itd.

1.3.8. SLUČAJNO PRETRAŽIVANJE SA OBRNUTIM KORAKOM

Značajno ubrzanje konvergencije metoda slučajnih smerova može se postići ako se uvede obrnuti korak, u slučaju da je korak učinjen u neuspešnoj oblasti. Ako se načini neki korak u slučajnom pravcu koji se pokaže neuspešnim, logično je da se proveri obrnuti smer. U slučaju da se i prvi i drugi korak pokažu neuspešnim, definiše se novi slučajan smer.

Algoritam ovog metoda se može opisati na sledeći način:

Korak 1. Izabirati početnu tačku $\mathbf{x}_0$.

Korak 2. Izgraditi slučajni smer, kako je već opisano.

Korak 3. Ako je generisani slučajni vektor odredio neuspešni smer optimizacije, pokušava se sa optimizacijom u suprotnom smeru.

Korak 4. Vratiti se na *Korak 2*.

Korak 5. Kriterijum za prekid algoritma je isti kao i za metod slučajnih smerova.

Ekstremum se lokalizuje sa tačnošću $h0_i$ za svaki upravljački parametar.

```
SluOb[q_,var_List,x_List,h_List,hmin_List,elha_,
                xmin_List,xmax_List]:=
  Block[{x0=x,ksi,M,q0,q1,n,i,L=0,x1=x,lg=True,h0=h,izbor,
                lh=elha, Lista={}},
      izbor=Input["Zelite li minimum(1) ili maksimum(2)?"];
      n=Length[var];
      If[n<=3, M=2^n+4, M=2*n+4];
      q0=q; Do[q0=q0/.var[[i]]->x0[[i]],{i,n}];
      Lista=Append[Lista,x0];
```

```
        q1=q0;
        While[L<=M && lg,
              ksi=Sluvec[n]; ksi=Limit[ksi,xmin,xmax];
              Do[x1[[i]]=x0[[i]]+h0[[i]]*ksi[[i]],{i,n}];
              x1=Limit[x1,xmin,xmax];
              q1=q; Do[q1=q1/.var[[i]]->x1[[i]],{i,n}];
              If[(izbor==2 && q1<q0) || (izbor==1 && q1>q0),
                    Do[x1[[i]]=x0[[i]]-h0[[i]]*ksi[[i]],{i,n}];
                    x1=Limit[x1,xmin,xmax];
                    q1=q; Do[q1=q1/.var[[i]]->x1[[i]],{i,n}];
              ];
              If[(izbor==2 && q1>q0) || (izbor==1 && q1<q0),
                 q0=q1; x0=x1; L=0;Lista=Append[Lista,x0],
                 L=L+1;
                 lg=False;
                 Do[Which[h0[[i]]>hmin[[i]], lg=True],{i,n}];
                 If[lg, Do[h0[[i]]/=lh,{i,n}]; ]
              ];
        ];
        {x0,q0, Lista}
    ]
```

1.3.9. METOD NAMETNUTE SLUČAJNOSTI

U ovom slučaju, koristi se isti pravac generisan slučajnim vektorom sve dok je pretraživanje u tom pravcu uspešno. Posle toga, formira se novi slučajni vektor, odnosno novi pravac za optimizacuju.

Algoritam ovog metoda je sledeći:

Korak 1. Formirati početnu tačku $\mathbf{x}_0$.

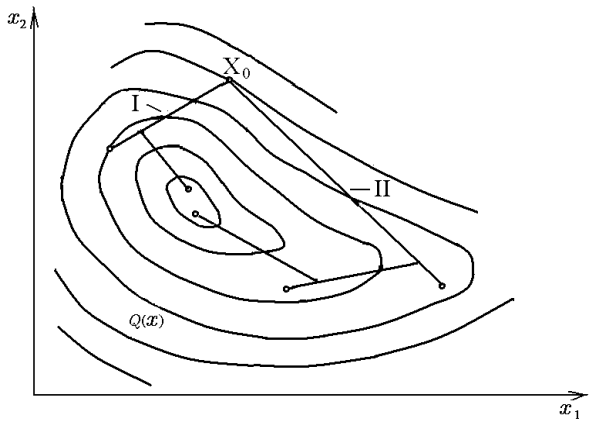
Korak 2. Izgraditi slučajni smer na ranije opisan način.

Korak 3. Lokalizovati ekstremum u tom smeru. Lokalizacija može da se izvrši sa konstantnim korakom dok se ne dobije *neuspešan* rezultat, ili se može izvršiti sa promenljivim korakom.

Korak 4. Poslednji *uspešni* korak u *uspešnom* pravcu se uzima za novu početnu tačku $\mathbf{x}_0$ i algoritam se nastavlja od *Koraka 2*.

Korak 5. Kriterijum za prekid algoritma je isti kao i za metod slučajnih smerova.

Metod je grafički predstavljen na slici 1.3.14. Prikazana su dva moguća smera (I i II) za dostizanje maksimuma od jedne početne tačke $\mathbf{x}_0$.



Sl. 1.3.14

```
SluCon[q_,var_List,x_List,h_List,hmin_List,elha_xmin_List,
                xmax_List]:=
  Block [{x0=x,ksi,M,q0,q1,n,i,L=0,x1=x,lg=True,h0=h,izbor,
                lh=elha, Lista={}},
     izbor=Input["Zelite li minimum (1) ili maksimum(2)?  "];
     n=Length[var];
     If[n<=3, M=2^n+4, M=2*n+4];
     q0=q; Do[q0=q0/.var[[i]]->x0[[i]],{i,n}];
     Lista=Append[Lista,x0];
     q1=q0;
     While[L<=M && lg,
        ksi=Sluvec[n]; ksi=Limit[ksi,xmin,xmax];
        Do[x1[[i]]=x0[[i]]+h0[[i]]*ksi[[i]],{i,n}];
        x1=Limit[x1,xmin,xmax];
        q1=q; Do[q1=q1/.var[[i]]->x1[[i]],{i,n}];
        While[(izbor==1 && q1<q0) || (izbor==2 && q1>q0),
           q0=q1; x0=x1; L=0;
           Do[x1[[i]]=x0[[i]]+h0[[i]]*ksi[[i]],i,n],
           x1=Limit[x1,xmin,xmax];
           q1=q; Do[q1=q1/.var[[i]]->x1[[i]],{i,n}];
           Lista=Append[Lista,x0];
        ];
        L=L+1; lg=False;
        Do[Which[h0[[i]]>hmin[[i]], lg=True],{i,n}];
        If[lg, Do[h0[[i]]/=lh,{i,n}]; ]
```

```
        ];
        {x0,q0, Lista}
    ]
```

Izbor početne tačke pri slučajnom traženju. Brzina konvergencije algoritma suštinski zavisi od izbora početne tačke. Osnovni metodi za izbor početne tačke su:

1° Početna tačka se zadaje u oblasti ekstremuma ako *apriori* postoji takva informacija.

2° Početna tačka **x0** se zadaje u središtu dozvoljene oblasti $xmin_i, xmax_i$, $i = 1, \dots, n$:

$$x0_i = \frac{1}{2}(xmax_i + xmin_i), \qquad i = 1, \dots, n.$$

3° Početna tačka se generiše na slučajan način

$$x0_i = xmin_i + \alpha_i(xmax_i - xmin_i), \quad i = 1, \dots, n,$$

pri čemu je α_i pseudoslučajni broj.

4° Odabira se početna tačka iz skupa slučajnih tačaka. Ovo predstavlja kombinaciju prostog slučajnog traženja i adaptivnog metoda. Generiše se niz slučajnih tačaka, a za početnu se uzima ona u kojoj ciljna funkcija ima najbolju vrednost $Q(\mathbf{x}_0)$. Broj generisanih tačaka N_0 može se definisati empirijskim formulama

$$N_0 = \begin{cases} 2^n + 4, & \text{za } n \leq 3, \\ 2n + 4, & \text{za } n > 3. \end{cases}$$

Ovakvom modifikacijom znatno se povećava konvergencija metoda.

Brzina konvergencije može se dodatno povećati ako se koristi početni algoritam slučajnog pretraživanja sa velikim parametrom u početnoj tački koja je izabrana iz skupa tačaka koje su lokalizovane drugim algoritmom (na primer, gradijentnim algoritmom).

Rezultati testiranja programa.

```
In[1]:= SluDir[x^2+y^2,{x,y},{-1,1},{0.1,0.2}, {0.01,0.02},4,{-5,-3},{2,2}]
   (*maksimum*)
Out[1]= {{-0.627172, 2}, 4.39334, {{-1, 1}, {-0.947012, 1.16961}, {-0.856402, 1.25423},
   {-0.826867, 1.4453}, {-0.791668, 1.6325}, {-0.782184, 1.8316}, {-0.709821, 1.96964},
```

```
    {-0.627172, 2}}}
In[2]:= SluDir[x^2+y^2,{x,y},{-1,1},{0.1,0.2}, {0.01,0.02},4,{-5,-3},{2,2}]
    (*minimum*)
Out[2]= {{-0.976039, 1.01416}, 1.98116, {{-1, 1}, {-0.994116, 1.00422},
    {-0.988058, 1.00729},{-0.982048, 1.01072}, {-0.976039, 1.01416}}}
In[3]:= SluOb[x^2+y^2,{x,y},{-1,1},{0.1,0.2}, {0.01,0.02},4,{-5,-3},{2,2}]
    (*maksimum*)
Out[3]= {{-0.737246, 2}, 4.54353, {{-1, 1}, {-0.917731, 1.1137}, {-0.826847, 1.19713},
    {-0.761856, 1.34913}, {-0.861496, 1.33216}, {-0.960766, 1.30804},
    {-0.909615, 1.47989}, {-0.870186, 1.66369}, {-0.811819, 1.82609},
    {-0.747974, 1.98002}, {-0.737246, 2}}}
In[4]:= SluOb[x^2+y^2,{x,y},{-1,1},{0.1,0.2}, {0.01,0.02},4,{-5,-3},{2,2}]
    (*minimum*)
Out[4]= {{-0.617204, 0.242588}, 0.43979, {{-1, 1}, {-1.01419, 0.802022},
    {-1.03335, 0.60573}, {-0.949694, 0.715303}, {-1.00881, 0.553997},
    {-1.02335, 0.356121}, {-1.04692, 0.161756}, {-0.994721, 0.332349},
    {-0.894751, 0.33724},{-0.80455, 0.423585}, {-0.704648, 0.43241},
    {-0.729485, 0.238678}, {-0.738451, 0.192004}, {-0.745307, 0.14392},
    {-0.73239, 0.186729}, {-0.718912, 0.228841}, {-0.699053, 0.259213},
    {-0.674067, 0.260897}, {-0.678432, 0.211665}, {-0.653432, 0.211801},
    {-0.648969, 0.220551}, {-0.649399, 0.208081}, {-0.643752, 0.213438},
    {-0.637742, 0.216864}, {-0.63403, 0.226921}, {-0.629265, 0.235011},
    {-0.624094, 0.24203}, {-0.619879, 0.251259}, {-0.623398, 0.24093},
    {-0.617204, 0.242588}}}
In[5]:= SluCon[x^2+y^2,{x,y},{-1,1},{0.1,0.2}, {0.01,0.02},4,{-5,-3},{2,2}]
    (*maksimum*)
Out[5]= {{-0.703839, 2}, 4.49539, {{-1, 1}, {-0.95064, 1.17394}, {-0.90128, 1.34788},
    {-0.85192, 1.52181}, {-0.80256, 1.69575}, {-0.753199, 1.86969}, {-0.703839, 2}}}
In[6]:= SluCon[x^2+y^2,{x,y},{-1,1},{0.1,0.2}, {0.01,0.02},4,{-5,-3},{2,2}]
    (*minimum*)
Out[6]= {{-0.836083, 1.12258}, 1.95921, {{-1, 1}, {-0.976583, 1.01751},
    {-0.953167, 1.03502},{-0.92975, 1.05253}, {-0.906333, 1.07004},
    {-0.882917, 1.08755}, {-0.8595, 1.10507},{-0.836083, 1.12258}}}
```

Kako se metod zasniva na slučajnom izboru tačaka iz zadatog intervala, prilikom više poziva za iste parametre dobijaju se različiti rezultati. Greška je relativno velika. Zato se ovakvi metodi najčešće koriste za određivanje početnih vrednosti za neke druge metode.

1.3.10. KOMPLEKS METOD

Ideja metoda je da se unutar dozvoljene oblasti formira "oblak" slučajnih tačaka, koji se naziva *kompleks*, a koji će se na odgovarajući način kretati

prema ekstremnoj vrednosti ciljne funkcije $Q(\mathbf{x})$. Pri premeštanju kompleksa koristi se težiste kompleksa, koje se označava sa $\mathbf{x}^{(c)}$. U ovom poglavlju se izučava kompleks metod koji je primenljiv samo za ograničenja parametara oblika $\Gamma_x = (\mathbf{xmin}, \mathbf{xmax})$.

Algoritam kompleks metoda:

Korak 1. Formirati početni kompleks koji sadrži N_m tačaka $\mathbf{x}^{(j)}$, $j = 1, \ldots, N_m$, formiranih prema formuli

$$\mathbf{x}_i^{(j)} = xmin_i + \alpha_i^{(j)}(xmax_i - xmin_i), \tag{1.3.4}$$

gde su $i = 1, \ldots, n;\ j = 1, \ldots, N_m$. Veličine $\alpha_i^{(j)}$ su ravnomerno raspoređeni slučajni brojevi iz intervala $[0, 1]$.

Broj tačaka u kompleksu se određuje prema empirijskoj formuli

$$N_m = \begin{cases} 2^n + 2, & n \leq 3, \\ 2n + 2, & n > 3. \end{cases}$$

Korak 2. U svakoj tački $\mathbf{x}^{(j)}$ kompleksa izračunati vrednost $Q^{(j)} = Q(\mathbf{x}^{(j)})$.

Korak 3. Determinisati tačku $\mathbf{x}^{(b)}$ unutar kompleksa u kojoj ciljna funkcija ima najbolju vrednost, kao i tačku $\mathbf{x}^{(w)}$ unutar kompleksa u kojoj ciljna funkcija ima najlošiju vrednost, tj.

$$Q^{(b)} = Q(\mathbf{x}^{(b)}) = \max_j Q^{(j)},\ Q^{(w)} = Q(\mathbf{x}^{(w)}) = \min_j Q^{(j)}. \tag{1.3.5}$$

Korak 4. Proveriti kriterijum za prekid algoritma:

$$\sqrt{\sum_{i=1}^{n} \left(x_i^{(b)} - x_i^{(w)} \right)^2} \leq \varepsilon_x, \tag{1.3.6}$$

gde je ε_x mali pozitivan broj koji, očigledno, predstavlja rastojanje između najbolje i najlošije vrednosti ciljne funkcije.

Ako je uslov (1.3.6) ispunjen, algoritam se prekida i edituju se vrednosti $Q^{(b)}$ i $\mathbf{x}^{(b)}$. U protivnom slučaju, algoritam se nastavlja od sledećeg koraka.

Korak 5. Izračunavaju se koordinate nove tačke $x^{(N)}$:

$$x_i^{(N)} = 2x_i^{(b)} - x_i^{(w)}, \quad i = 1, \ldots, n. \tag{1.3.7}$$

Korak 6. Proveriti ograničenja $\mathbf{x} \in \Gamma_x$, koristeći ranije opisanu proceduru *limit*.

Korak 7. Izračunati $Q^{(N)} = Q(\mathbf{x}^{(N)})$.

Korak 8. Ako je $Q^{(N)} > Q^{(w)}$, tačka $\mathbf{x}^{(N)}$ se uzima za novu tačku kompleksa umesto tačke $\mathbf{x}^{(w)}$, tj. $\mathbf{x}^{(w)} = \mathbf{x}^{(N)}$, $Q^{(w)} = Q^{(N)}$. Algoritam se nastavlja od *Koraka 3*.

Korak 9. Ako je $Q^{(N)} \leq Q^{(w)}$, nova tačka $\mathbf{x}^{(N)}$ se izostavlja i izračunavaju se koordinate tačke $\tilde{\mathbf{x}}^{(N)}$, koja se nalazi na sredini između tačaka $\mathbf{x}^{(w)}$ i $\mathbf{x}^{(b)}$:

$$\tilde{\mathbf{x}}_i^{(N)} = \frac{1}{2}\big(x_i^{(b)} + x_i^{(w)}\big), \qquad i = 1, \dots, n.$$

Korak 10. Izračunati

$$\tilde{Q}^{(N)} = Q\big(\tilde{\mathbf{x}}^{(N)}\big).$$

Korak 11. Nova tačka $\tilde{\mathbf{x}}^{(N)}$ se uzima umesto $\mathbf{x}^{(w)}$, tj. postavljaju se smene $\mathbf{x}^{(w)} = \tilde{\mathbf{x}}^{(N)}$, $Q^{(w)} = \tilde{Q}^{(N)}$. Algoritam se nastavlja od *Koraka 3*.

Mogu se istaći sledeće pozitivne osobine kompleks metoda:

1° Algoritam je jednostavan i pogodan za implementaciju na računarima.

2° Može se koristiti kako za izračunavanje bezuslovnog, tako i za izračunavanje uslovnog ekstremuma.

```
KompleksMetod[q_,prom_List,xmin_List,xmax_List,eps_]:=
    Block[{n=Length[prom],xb,xw,Nm,pniz,i,j,vektorq,
           qpom,qb,qw,xN,qN, kompleks,qkompleks={},
           minq,minind,maxq,maxind},
        If[n<=3,Nm=2^n+2,Nm=2n+2];
        kompleks=Table[xmin[[i]]+
             Random[]*(xmax[[i]]-xmin[[i]]),{Nm},{i,n}];
        Print["kompleks= ",kompleks];
        For[j=1,j<=Nm,j++,
          qpom=q;
          Do[qpom=qpom/.  prom[[i]]->kompleks[[j,i]],{i,n}];
          qkompleks=Append[qkompleks,qpom]
        ];
        Print["qkompleks= ",qkompleks];
        minq=qkompleks[[1]]; minind=1;
        maxq=qkompleks[[1]]; maxind=1;
        For[i=2,i<=Length[qkompleks],i++,
```

```
    If[qkompleks[[i]]<minq,
        minq=qkompleks[[i]]; minind=i
    ];
    If[qkompleks[[i]]>maxq,
        maxq=qkompleks[[i]]; maxind=i
    ];
  ];
  xb=kompleks[[minind]];  xw=kompleks[[maxind]];
  qb=minq; qw=maxq;
  While[Sqrt[Sum[(xb[[i]]-xw[[i]])^2,{i,n}]]>=eps,
        xN=2*(xb-xw);
        xN=limit[xN,xmin,xmax];
        qN=q;  Do[qN=qN/.  prom[[i]]->xN[[i]],{i,n}];
        If[qN >= qw,
           xN=(xb+xw)/2;
           xN=limit[xN,xmin,xmax];
        ];
        qN=q;  Do[qN=qN/.  prom[[i]]->xN[[i]],{i,n}];
        kompleks[[maxind]]=xN; qkompleks[[maxind]]=qN;
        Print["kompleks= ",kompleks];
        Print["qkompleks= ",qkompleks];
        minq=qkompleks[[1]]; minind=1;
        maxq=qkompleks[[1]]; maxind=1;
        For[i=2,i<=Length[qkompleks],i++,
            If[qkompleks[[i]]<minq,
                 minq=qkompleks[[i]]; minind=i
            ];
            If[qkompleks[[i]]>maxq,
                 maxq=qkompleks[[i]]; maxind=i
            ]
        ];
        xb=kompleks[[minind]]; xw=kompleks[[maxind]];
        qb=minq; qw=maxq;
  ];
  {xb,qb}
]
```

1.3.11. POWELOV VIŠEDIMENZIONALNI METOD

Powelov metod se može koristiti i za izračunavanje lokalnog minimuma proizvoljne funkcije f sa n promenljivih (videti [73]), pri čemu ima tzv. svojstvo kvadratnog završavanja. Kažemo da neki numerički metod za rešavanje problema bezuslovne optimizacije ima svojstvo kvadratnog završavanja ako za svaku funkciju oblika

$$f(x) = \frac{1}{2}(\mathbf{x}, A\mathbf{x}) + (\mathbf{b}, \mathbf{x}) + a,$$

sa pozitivno definitnom matricom A, metod daje optimalno rešenje u najviše n iteracija. Powellov metod je efikasan u izračunavanju optimalne tačke proizvoljne funkcije f, ali se u opštem slučaju gubi svojstvo kvadratnog završavanja. Potrebno je na početku primene Powellovog metoda odrediti n linearno nezavisnih konjugovanih vektora. Vrlo jednostavan i efikasan metod za određivanje n linearno nezavisnih konjugovanih vektora dao je Powell. U kratkim crtama Powellov metod se može opisati sledećim algoritmom:

Korak 1. Odrediti pravilo zaustavljanja za neki dovoljno mali pozitivan broj ε. Označimo proizvoljnu početnu aproksimaciju sa

$$\mathbf{t}_0^{(1)} = \mathbf{x}^0.$$

Korak 2. Pre k-te iteracije koristi se n linerno nezavisnih (ne nužno konjugovanih) vektora $\mathbf{u}_1^{(k)}$, $\mathbf{u}_2^{(k)}$, ... , $\mathbf{u}_n^{(k)}$. Kao početni vektori obično služe jedinični vektori $\mathbf{e}^1, \dots, \mathbf{e}^n$, tj.
$\mathbf{u}_1^{(1)} = \mathbf{e}^1$, $\mathbf{u}_2^{(1)} = \mathbf{e}^2$, ..., $\mathbf{u}_n^{(1)} = \mathbf{e}^n$.

Korak 3. Početak k-te iteracije ($k = 1, 2, \dots$): Izračunati brojeve θ_1^*, θ_2^*, ... , θ_n^* takve da važi

$$f\big(\mathbf{t}_{i-1}^{(k)} + \theta_i^* \mathbf{u}_i^{(k)}\big) = \min_{\theta_i} f\big(\mathbf{t}_{i-1}^{(k)} + \theta_i^* \mathbf{u}_i^{(k)}\big),$$

gde je

$$\mathbf{t}_i^{(k)} = \mathbf{t}_{i-1}^{(k)} + \theta_i^* \mathbf{u}_i^{(k)}, \qquad i = 1, 2, \dots, n.$$

U prvoj iteracij je $k = 1$. Početi sa $i = 1$, tj. naći optimalnu tačku θ_1^* funkcije $f\big(\mathbf{t}_0^{(1)} + \theta_1 \mathbf{u}_1^{(1)}\big) = f\big(\mathbf{x}^0 + \theta_1 \mathbf{e}^1\big)$. Zatim se izračunava

$$\mathbf{t}_1^{(1)} = \mathbf{t}_0^{(1)} + \theta_1^* \mathbf{u}_1^{(1)}$$

i nastavlja se sa $i = 2$, tj. izračunava se optimalna tačka θ_2^* funkcije $f\big(\mathbf{t}_1^{(1)} + \theta_2 \mathbf{u}_2^{(1)}\big)$, a zatim izračunava

$$\mathbf{t}_2^{(1)} = \mathbf{t}_1^{(1)} + \theta_2^* \mathbf{u}_2^{(1)}(1)$$

i nastavlja sa $i = 3$, itd. Na kraju petlje izračunata su sva optimalna rešenja θ_1^*, θ_2^*, $\ldots$, θ_n^* jednodimenzionih problema minimizacije, kao i svi vektori $\mathbf{t}_1^{(1)}$, $\mathbf{t}_2^{(1)}$, $\ldots$, $\mathbf{t}_n^{(1)}$.

Korak 4. Nastavak k-te iteracije: Zameniti

$$\mathbf{u}_i^{(k+1)} = \mathbf{u}_{i+1}^{(k)}, \qquad i = 1, \ldots, n-1,$$

i staviti

$$\mathbf{u}_n^{(k+1)} = \mathbf{u}_{n+1}^{(k)} = \mathbf{t}_n^{(k)} - \mathbf{t}_0^{(k)}.$$

Ovaj korak opisuje zamenu početnih jediničnih vektora sa konjugovanim vektorima. Dok se u prvoj iteaciji koriste vektori

$$\big\{\mathbf{u}_1^{(1)},\ \mathbf{u}_2^{(1)},\ \ldots,\ \mathbf{u}_n^{(1)}\big\},$$

za izračunavanje prve aproksimacije $\mathbf{x}^1$ koristi se konjugovani vektor $\mathbf{u}_{n+1}^{(1)}$. U izračunavanju druge aproksimacije koristiće se novi skup vektora

$$\big\{\mathbf{u}_2^{(1)},\ \mathbf{u}_3^{(1)},\ \ldots,\ \mathbf{u}_n^{(1)},\ \mathbf{u}_{n+1}^{(1)}\big\} = \big\{\mathbf{u}_1^{(2)},\ \mathbf{u}_2^{(2)},\ \ldots,\ \mathbf{u}_{n-1}^{(2)},\ \mathbf{u}_n^{(2)}\big\}$$

Treba primetiti da je prvi jedinični vektor otpao, zapravo je zamenjen konjugovanim vektorom $\mathbf{u}_{n+1}^{(1)}$. Da bi se moglo lako kontrolisati koji će vektor otpasti i koji su vektori konjugovani, uvedeno je uređenje među vektorima. Novi konjugovani vektor uvek se pojavljuje u skupu tekućih vektora sa desne strane.

Korak 5. Završetak k-te iteracije: Izračunati broj θ_{n+1}^* sa osobinom

$$f\big(\mathbf{t}_n^{(k)} + \theta_{n+1}^*\big(\mathbf{t}_n^{(k)} - \mathbf{t}_0^{(k)}\big)\big) = \min_{\theta_{n+1}} f\big(\mathbf{t}_n^{(k)} + \theta_{n+1}\big(\mathbf{t}_n^{(k)} - \mathbf{t}_0^{(k)}\big)\big),$$

i

$$\mathbf{x}^k = \mathbf{t}_n^{(k)} + \theta_{n+1}^*\big(\mathbf{t}_n^{(k)} - \mathbf{t}_0^{(k)}\big).$$

(Vektor $\mathbf{u}_{n+1}^{(k)} = \mathbf{t}_n^{(k)} - \mathbf{t}_0^{(k)}$ jedan je od konjugovanih vektora. U smeru toga vektora, uzetog iz tačke $\mathbf{t}_n^{(k)}$, nalazi se aproksimacija $\mathbf{x}^k$).

Korak 6. Primena pravila zaustavljanja ($k = 1, \dots$). Ako je

$$\| \mathbf{x}^k - \mathbf{x}^{k-1} \| < \varepsilon$$

proces se zaustavlja jer je $\mathbf{x}^k$ dovoljno dobra aproksimacija lokalnog optimuma $\mathbf{x}^k$. Ako ovo nije slučaj, staviti

$$\mathbf{t}_0^{(k)} = \mathbf{x}^{k-1}$$

i vratiti se na *Korak 3* i nastaviti sa $(k+1)$-vom iteracijom.

Sledi implementacija u jeziku MATHEMATICA:

```
powelvise[q_,pr_,x_,eps_] :=
   Block[{funpom,fjed,vt0=x,vt1=x,vx0=x,vx1=x,rast=2*eps,
     vvu=Table[0,{Length[pr]},{Length[pr]}],
     vnovi=Table[0,{Length[pr]}],
     vpomkonj=Table[0,{Length[pr]}],teta,tetamin,
     pomeraj,n=Length[pr],i,j,jj,iter=0},
     Do[vvu[[i,i]]=1,{i,1,n}];
     (* glavna petlja iteracije *)
     While[rast>eps,
       iter++;
       vt0=vx0;
       Do[vpomkonj=vvu[[i]]; fjed=q;
         Do[fjed=fjed/.pr[[j]]->
             (vt0[[j]]+teta*vpomkonj[[j]]),{j,1,n}];
         tetamin=dskpowel[fjed,{teta},0,N[eps/10]];
         pomeraj=tetamin[[1]];
         Do[vt1[[j]]=N[vt0[[j]]+pomeraj*vpomkonj[[j]]],
            {j,1,n}];
         vt0=vt1,
              {i,1,n}];
         Do[vvu[[i]]=vvu[[i+1]],{i,1,n-1}];
         Do[vpomkonj[[i]]=vt1[[i]]-vx0[[i]],{i,1,n}];
         vvu[[n]]=vpomkonj;
         fjed=q;
         Do[fjed=fjed/.pr[[j]]->
```

```
            (vx0[[j]]+teta*vpomkonj[[j]]),{j,1,n}];
        tetamin=dskpowel[fjed,{teta},0,N[eps/10]];
        pomeraj=tetamin[[1]];
        Do[vx1[[j]]=N[vx0[[j]]+pomeraj*vpomkonj[[j]]],
             {j,1,n}];
        rast=mera[vx1,vx0];
        vx0=vx1;
        funpom=q;
        Do[funpom=N[funpom/.pr[[i]]->vx1[[i]]],{i,1,n}];
      ]
      Return[{vx1, funpom}];
    ]
```

2. GRADIJENTNI METODI

2.1. Opšte napomene

Kao što je rečeno u prvom poglavlju, metodi direktnog pretraživanja (negradijentni metodi) zasnivaju se na upoređivanju vrednosti funkcije cilja $Q(\mathbf{x})$. Početno rešenje $\mathbf{x}$ se proizvoljno odabira, a zatim se svako poboljšanje rešenja u narednom koraku procenjuje na osnovu prethodnih rešenja i vrednosti funkcije cilja. Osnovni nedostatak metoda pretraživanja je u tome što oni imaju slučajni, a ne sistematski karakter. Na taj način se primenom ovih metoda ne dobija garantovano najbolje rešenje, već samo, u izvesnom smislu, zadovoljavajuće rešenje. Osim toga, sve ove metode prati veliki broj računskih operacija pa se zato primenjuju samo za rešavanje zadataka nelinearnog programiranje manjih dimenzija.

Graficki metodi su ograničeni i to samo na zadatke kod kojih figurišu jedna ili dve promenljive. Domen ograničenja (D) se predstavlja tako što se u ravni promenljivih grafički predstavljaju zadata ograničenja. Optimalno rešenje se odabira iz raznih vrednosti dopustivog plana $\mathbf{x}$ iz domena D, vizuelnom inspekcijom grafika funkcije.

Za razliku od ovih metoda, gradijetni metodi predstavljaju metode sistematskog pretraživanja i izračunavanja rešenja. Do rešenja se dolazi, polazeći od izabranog polaznog plana $\mathbf{x}^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$ i težeći da se u jednom koraku izračuna što je moguće efikasniji plan $\mathbf{x}^{(1)} = (x_1^{(1)}, \ldots, x_n^{(1)})$. Da bi se to postiglo potrebno je izračunati najbolji smer promene vrednosti ciljne funkcije, koji se kod gradijentnih metoda izražava u odnosu na smer gradijenta (suprotan smer od smera gradijenta u procesu traženja minimuma, a

upravo smer gradijenta pri traženju maksimuma). Razlike kod pojedinih gradijentnih metoda su u odnosu na veličinu pomeranja u pravcu koji je određen gradijentnim vektorom.

Proces izračunavanja gradijenta funkcije $Q(\mathbf{x})$ je obiman posao pa je ponekad jednostavnije da se to izračunavanje vrši povremeno, što se efikasno koristi u modifikovanim gradijentnim metodama.

Druga klasa metoda se zasniva na pretrazivanju minimuma (maksimuma) funkcije $Q(\mathbf{x})$ duž linije najvećeg pada, odnosno uspona. Algoritam ovih metoda se sastoji u određivanju smera najvećeg pada funkcije $Q(\mathbf{x})$ koji je suprotan pravcu gradijenta funkcije, a zatim se vrši elementarni pomeraj u tom smeru do susedne tačke u kojoj se, takođe, određuje smer najvećeg pada. Ovi metodi se nazivaju metodi najstrmijeg pada (steepest descent).

Analitičku interpretaciju gradijentnih metoda ilustrovaćemo za slučaj funkcije $Q(\mathbf{x}) = Q(x_1, \dots, x_n)$ od n nezavisno promenljivih. Ako se u n-dimenzionalnom prostoru odabere smer $\mathbf{s}$, onda je izvod funkcije $Q(\mathbf{x})$ u tački $\mathbf{x} = (x_1, \dots, x_n)$ i u smeru $\mathbf{s}$ određen izrazom

$$\frac{dQ}{ds} = \sum_{j=1}^{n} \frac{\partial Q}{\partial x_j} \frac{dx_j}{d\mathbf{s}}\,, \tag{2.1.1}$$

gde su:

$\dfrac{\partial Q}{\partial x_j}$ – parcijalni izvodi funkcije $Q(\mathbf{x})$;

$\dfrac{dx_j}{d\mathbf{s}}$ – kosinusi uglova koje smer $\mathbf{s}$ zaklapa sa koordinatnim osama;

$\dfrac{dQ}{d\mathbf{s}}$ – izvod funkcije $Q(\mathbf{x})$ u smeru $\mathbf{s}$.

Uvodeći oznaku

$$z_j = \frac{dx_j}{d\mathbf{s}}\,,$$

izraz (2.1.1) može se napisati u obliku

$$\frac{dQ}{d\mathbf{s}} = \sum_{j=1}^{n} \frac{\partial Q}{\partial x_j} z_j. \tag{2.1.2}$$

Za slučaj kada se funkcija $Q(\mathbf{x})$ minimizira (maksimizira), potrebno je odrediti onaj smer po kome izvod funkcije ima najmanju (najveću) vrednost.

Promenljive z_j nisu međusobno nezavisne jer postoji relacija

$$d\mathbf{s} = \sqrt{\left(\sum_{j=1}^{n} {dx_j}^2\right)},$$

koja se može predstaviti u obliku

$$1 - \sum_{j=1}^{n} z_j^2 = 0. \tag{2.1.3}$$

Na taj način, postavljen je problem minimizacije funkcije $\dfrac{dQ}{d\mathbf{s}}$ definisane jednačinom (2.1.2), uz prisustvo ograničenja (2.1.3). Uopštena Lagrangeova funkcija, uz pretpostavku da postoji samo jedno ograničenje oblika (2.1.3), može se napisati u obliku

$$\Phi = \Phi\left(\mathbf{x}, \Lambda\right) = \sum_{j=1}^{n} \frac{\partial Q}{\partial x_j} z_j + \lambda_0 \left(1 - \sum_{j=1}^{n} {z_j}^2\right),$$

gde je λ_0 Lagrangeov množitelj. Izračunavanjem parcijalnih izvoda uopštene Lagrangeove funkcije Φ po svakoj od promenljivih z_j i izjednačavanjem dobijenih jednačina sa nulom dobija se sistem jednačina

$$\frac{\partial Q}{\partial x_j} - 2\lambda_0 z_j = 0, \quad j = 1, \ldots, n,$$

odakle sleduje

$$z_j = \frac{1}{2\lambda_0} \frac{\partial Q}{\partial x_j}, \qquad j = 1, \ldots, n. \tag{2.1.4}$$

Zamenom vrednosti z_j iz izraza (2.1.4) u (2.1.3) dobija se

$$\lambda_0 = \pm \frac{1}{2} \sqrt{\sum_{j=1}^{n} \left(\frac{\partial Q}{\partial x_j}\right)^2}. \tag{2.1.5}$$

Kada se vrednost za λ_0, data pomoću (2.1.5), zameni u (2.1.4), dobija se vrednost za z_j koja odgovara smeru najvećeg pada ciljne funkcije

$$z_j = -\frac{\partial Q}{\partial x_j} \left[\sum_{i=1}^{n} \left(\frac{\partial Q}{\partial x_i}\right)^2\right]^{-1/2}, \qquad j = 1, \ldots, n,$$

odnosno smeru najvećeg uspona

$$z_j = \frac{\partial Q}{\partial x_j}\left[\sum_{i=1}^{n}\left(\frac{\partial Q}{\partial x_i}\right)^2\right]^{-1/2}, \qquad j = 1, \ldots, n.$$

Metodi bezuslovne optimizacije bazirani na izvodima ciljne funkcije mogu da grubo podeliti u dve klase. U prvu klasu spadaju metodi koji koriste samo prvi izvod ciljne funkcije i oni se nazivaju *gradijentni metodi prvog reda*. Najpoznatiji gradijentni metod prvog reda je *Cauchyev metod najstrmijeg opadanja*. Ovaj metod ima linearnu konvergenciju i odlikuje se dobrim napretkom prema tački optimuma iz "dalekih" početnih aproksimacija, ali i sporom konvergencijom u blizini optimalne tačke. U drugu klasu metoda spadaju oni koji koriste i prvi i drugi izvod ciljne funkcije (ili neke njihove aproksimacije) i nazivaju se *gradijentni metodi drugog reda*. Najpoznatiji metod drugog reda jeste *Newtonov metod*. Newtonov metod se odlikuje kvadratnom konvergencijom, što znači da kada konvergira, on je brži od Cauchyevog metoda. Međutim, Newtonov metod je manje pouzdan od Cauchyevog. Najbolje osobine Cauchyevog i Newtonovog metoda su ujedinjene u tzv. *metodima promenljive metrike*. Ovi metodi imaju najmanje linearni red konvergencije, uz asimptotski kvadratnu konvergenciju. Zato se često kaže da se ti metodi odlikuju kvadratnim završavanjem.

Navodimo osnovne pojmove koji su neophodni za razumevanje gradijentnih metoda optimizacije.

Vektor-gradijent ∇Q u n-dimenzionalnom prostoru ima n komponenti, koje su jednake parcijalnim izvodima po svakom upravljačkom parametru u tački $\mathbf{x}^{(k)}$, tj.

$$\nabla Q(\mathbf{x}^{(k)}) = \operatorname{grad} Q(\mathbf{x}^{(k)}) = \left\{\frac{\partial Q(\mathbf{x}^{(k)})}{\partial x_i}\right\}_{i=1,\ldots,n}. \tag{2.1.6}$$

Zbog jednostavnijeg označavanja usvojićemo oznaku $Q^{(k)} = Q(\mathbf{x}^{(k)})$, tako da je gradijent u tački $\mathbf{x}^{(k)}$ označen na sledeći način

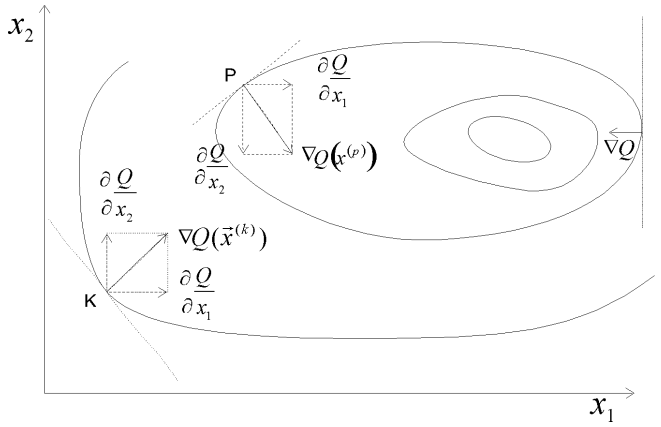
$$\nabla Q(\mathbf{x}^{(k)}) = \left\{\frac{\partial Q^{(k)}}{\partial x_1}, \ \ldots, \frac{\partial Q^{(k)}}{\partial x_n}\right\}. \tag{2.1.7}$$

Hesseova matrica je kvadratna matrica parcijalnih izvoda drugog reda funk-

cije $Q(\mathbf{x})$ u tački $\mathbf{x}^{(k)}$

$$\nabla^2 Q(x^{(k)}) = \mathbb{H}(\mathbf{x}^{(k)}) = \begin{bmatrix} \dfrac{\partial^2 Q^{(k)}}{\partial {x_1}^2} & \cdots & \dfrac{\partial^2 Q^{(k)}}{\partial x_1 \partial x_n} \\ \vdots & & \\ \dfrac{\partial^2 Q^{(k)}}{\partial x_n \partial x_1} & & \dfrac{\partial^2 Q^{(k)}}{\partial {x_n}^2} \end{bmatrix}. \tag{2.1.8}$$

Gradijentni vektor $\nabla Q(\mathbf{x})$ je u svakoj tački $\mathbf{x}^{(k)} = (x_1^{(k)}, \ldots, x_n^{(k)})$ prostora normalan na površ sa konstantnom vrednošću $Q(\mathbf{x})$ (to je linija u slučaju dva upravljačka parametra) i prolazi kroz zadatu tačku. Za slučaj dva upravljačka parametra, ova činjenica se može ilustrovati slikom 2.1.1.



Sl. 2.1.1

Gradijentni vektor u svakoj tački $\mathbf{x}^{(k)}$ je vektor koji ima smer najbržeg rašćenja $Q(\mathbf{x})$ od te tačke. Algoritam gradijentnih metoda optimizacije se sastoji u tome da se od zadate početne ili izračunate tačke $\mathbf{x}^{(k)}$ prelazi na sledeću tačku $\mathbf{x}^{(k+1)}$ sa korakom $\Delta\mathbf{x}^{(k)}$ u smeru gradijenta, pri izračunavanju maksimuma

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)} = \mathbf{x}^{(k)} + \mathbf{h}^{(k)} \nabla Q(\mathbf{x}^{(k)}), \tag{2.1.9}$$

ili u obrnutom smeru od smera gradijenta pri izračunavanju minimuma

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{h}^{(k)} \nabla Q(\mathbf{x}^{(k)}). \tag{2.1.10}$$

Pri zadatom parametru koraka $\mathbf{h}^{(k)} = (h_i^{(k)})$, $i = 1, \dots, n$, kretanje u pravcu gradijenta ostvaruje se formulama

$$x_i^{(k+1)} = x_i^{(k)} + h_i^{(k)} \frac{\partial Q^{(k)}}{\partial x_i}, \qquad i = 1, \dots, n, \tag{2.1.11}$$

pri nalaženju maksimuma, odnosno

$$x_i^{(k+1)} = x_i^{(k)} - h_i^{(k)} \frac{\partial Q^{(k)}}{\partial x_i}, \qquad i = 1, \dots, n, \tag{2.1.12}$$

pri nalaženju minimuma funkcije $Q(\mathbf{x})$.

U formulama (2.1.11) i (2.1.12) kretanje je u smeru gradijenta samo ako su sve veličine $h_i^{(k)}, i = 1, \dots, n$, međusobno jednake. Međutim, u nekim metodima se parametri koraka (tj. veličine $h_i^{(k)}, i = 1, \dots, n$) mogu odabrati proizvoljno.

U gradijentnim metodima mogu se koristiti formule sa koordinatama normalizovanog gradijentnog vektora

$$x_i^{(k+1)} = x_i^{(k)} + h_i^{(k)} \frac{\dfrac{\partial Q^{(k)}}{\partial x_i}}{\sqrt{\sum\limits_{i=1}^{n} \left(\dfrac{\partial Q^{(k)}}{\partial x_i} \right)^2}}, \qquad i = 1, \dots, n. \tag{2.1.13}$$

U formuli (2.1.13), normirani gradijent-vektor samo ukazuje na smer najbrže promene ciljne funkcije, ali ne određuje brzinu napredovanja prema ekstremumu. Ta brzina se zadaje parametrima koraka, $h_i^{(k)}$, $i = 1, \dots, n$. Normalizacija gradijenta doprinosi stabilnosti metoda.

Teorijski se procedura gradijentnog pretraživanja završava u stacionarnoj tački u kojoj su sve koordinate gradijenta jednake nuli, tj. u kojoj je Euklidova norma gradijenta jednaka nuli:

$$\|\nabla Q(\mathbf{x})\| = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial Q}{\partial x_i} \right)^2} = 0. \tag{2.1.14}$$

Prema tome, može se koristiti sledeći kriterijum za prekid gradijentnog pretraživanja:

$$\sqrt{\sum_{i=1}^{n} \left(\frac{\partial Q}{\partial x_i} \right)^2} \leq \varepsilon, \tag{2.1.15}$$

pri čemu je ε zadati mali pozitivan broj. Kao kriterijum za prekid pretraživanja može se uzeti i Čebiševljeva norma gradijenta, tj. kriterijum

$$\sum_{i=1}^{n} \left| \frac{\partial Q^{(k)}}{\partial x_i} \right| \leq \varepsilon. \tag{2.1.16}$$

2.2. O simboličkoj implementaciji

U [2], [13], [16], [37], [60] opisana je implementacija nekih metoda gradijentne optimizacije u FORTRANu. Međutim, u ovim programima nije jednostavno koristiti ciljne funkcije koje nisu definisane potprogramima, kao što je opisano u **(1M)** i **(1U)**. Takođe, u toku implementacije gradijentnih metoda optimizacije, moraju se definisati funkcije kojima se izračunavaju i parcijalni izvodi ciljne funkcije. Na taj način, primena proizvoljnog gradijentnog metoda na novu funkciju je uslovljena ovim potprogramima.

S druge strane, ugrađene funkcije u paketu MATHEMATICA [68], [69] u kojima se koriste gradijentni metodi optimizacije nisu zadovoljavajuće. Na raspolaganju je jedino funkcija *FindMinimum*.

Funkcija *FindMinimum* izračunava lokalni minimum date funkcije prema zadatoj početnoj tački. Polazeći od zadate startne tačke, u ovoj funkciji se minimum zadate funkcije izračunava koristeći *putanju najstrmijeg pada*.

`FindMinimum[f, {x, x0}]` traži lokalni minimum funkcije f, polazeći od tačke $x = x_0$;

`FindMinimum[f, {x,x0}, {y,y0},...]` traži lokalni minimum funkcije f više promenljivih;

`FindMinimum[f, {x,{x0,x1}}]` traži lokalni minimum funkcije f koristeći x_0 i x_1 za prve dve vrednosti x (ovakav oblik se mora koristiti ako simbolički izvodi za f ne mogu biti nađeni);

`FindMinimum[f, {x, xstart,xmin,xmax}]` traži lokalni minimum funkcije f polazeći od $xstart$, a pretraživanje se zaustavlja kada x bilo kada izađe izvan opsega $[xmin,\ xmax]$.

U toku optimizacije u paketu MATHEMATICA, može se koristi jedino metod najstrmijeg pada. Naš cilj je da se razvije programski paket za optimizaciju. Pored uslova **(1M)**–**(3M)** i **(1U)**–**(3U)**, simbolička implementacija gradijentnih metoda povlači neke dodatne prednosti.

(1G) *Automatsko diferenciranje* u proceduralnim programskim jezicima deli se na numeričko diferenciranje (koje produkuje numeričku aproksimaciju

za $f'(x)$) i simboličko diferenciranje koje produkuje formulu za $f'(x)$. Izračunavanje parcijalnih izvoda pomoću konačnih razlika može da se implementira u bilo kom programskom jeziku i za nas nije od interesa. Poznato je da simboličko diferenciranje nije podesan problem za implementaciju u proceduralnim programskim jezicima: simbolički algebarski izraz mora da se podvrgne sintaksnoj analizi u kojoj se formira stablo izraza. Zatim se zadati izraz konvertuje u inverznu poljsku notaciju, i najzad se formira stablo parcijalnih izvoda [6], [31]. S druge strane, ovi problemi se mogu lako rešiti u funkcionalnim programskim jezicima. U paketu MATHEMATICA može se koristiti operator diferenciranja D, koji je na raspolaganju kao ugrađena funkcija. U LISPu se mogu jednostavno implementirati funkcije za simboličko diferenciranje aritmetičkih izraza. To se može uraditi jednostavnom generalizacijom principa koji su opisani u [12], [14], [44], [61].

(2G) Mogućnost efikasne primene generisanih parcijalnih izvoda na proizvoljne argumente. U programskom paketu MATHEMATICA, parcijalni izvodi koji su generisani operatorom diferenciranja D mogu direktno da se koriste kao funkcije. U programskom jeziku LISP, izrazi koji predstavljaju parcijalne izvode ciljne funkcije mogu se jednostavno transformisati u odgovarajuće *lambda funkcije* koje su primenljive na zadatu listu argumenata.

(3G) Može se konstruisati, na prirodan način, nova ciljna funkcija koja zavisi od različitih parametara kao, na primer, dužina koraka koji se koristi kod *metoda najstrmijeg pada* ili kod *metoda konjugovanih pravaca*. Na taj način, *optimizacija po liniji* (*optimization along a line*) je problem koji se prirodno rešava simboličkim procesiranjem.

2.3. Formiranje gradijenta

Komponente gradijenta ciljne funkcije mogu se definisati na različite načine:

1° U slučaju kada je poznat analitički izraz ciljne funkcije, izvodi se mogu definisati analitički iako to ponekad može biti komplikovano.

2° Koordinate gradijenta mogu se definisati eksperimentalnim putem.

3° U velikom broju realnih tehničkih zadataka ne postoji eksplicitan izraz za $Q(\mathbf{x})$ ili je taj izraz veoma komplikovan, tako da je analitičko izražavanje izvoda nemoguće. U takvim slučajevima, koristi se numeričko izračunavanje koordinata gradijenta.

Postoji nekoliko metoda za numeričko diferenciranje.

Metod A. Zadaju se priraštaji za svaki upravljački parametar posebno. Priraštaj koji odgovara upravljačkom parametru x_i označimo sa Δx_i, $i =$

$1, \ldots, n$. Tada se koriste sledeće aproksimacije za parcijalne izvode:

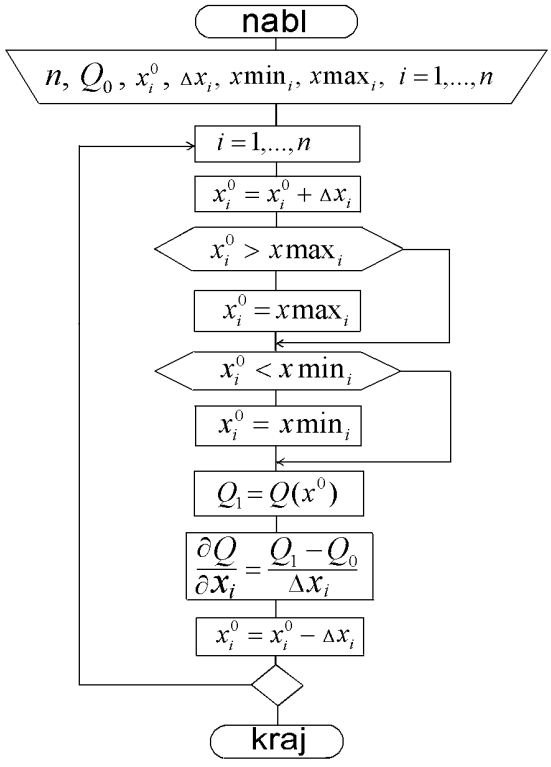
$$\frac{\partial Q}{\partial x_i} \approx \frac{\Delta Q}{\Delta x_i} = \frac{Q(x_1^{(0)}, \ldots, x_i^{(0)} + \Delta x_i, \ldots, x_n^{(0)}) - Q(x_1^{(0)}, \ldots, x_i^{(0)}, \ldots, x_n^{(0)})}{\Delta x_i},$$

gde je $i = 1, \ldots, n$. Za izračunavanje svih parcijalnih izvoda neophodno je $S = n + 1$ izračunavanja vrednosti ciljne funkcije.

Metod B. Parcijalni izvodi $\partial Q/\partial x_i$ mogu se tačnije aproksimirati ako se koriste tzv. dvostrani priraštaji za x_i, tj. tzv. centralne formule

$$\frac{\partial Q}{\partial x_i} \approx \frac{Q(x_1^{(0)}, \ldots, x_i^{(0)} + \Delta x_i, \ldots, x_n^{(0)}) - Q(x_1^{(0)}, \ldots, x_i^{(0)} - \Delta x_i, \ldots, x_n^{(0)})}{2\Delta x_i},$$

gde je $i = 1, \ldots, n$. U ovom slučaju, neophodno je $S = 2n$ izračunavanja vrednosti ciljne funkcije da bi se izračunale sve koordinate gradijenta. Uprkos tome što je metod A grublji u proceni izvoda $\partial Q/\partial x_i$, on se preporučuje zbog manjeg broja izračunavanja vrednosti funkcije $Q(\mathbf{x})$.



Sl. 2.3.1

Pri numeričkom izračunavanju se preporučuje dvostruka tačnost (`double precision`) pri programskoj realizaciji na računskim mašinama.

Osnovni problem pri numeričkom izračunavanju parcijalnih izvoda jeste izbor parametara za numeričko diferenciranje Δx_i. Ne postoje metodi za predviđanje najboljih vrednosti za Δx_i.

Algoritam za numeričko diferenciranje koji koristi jednostrani priraštaj za parametre x_i, $i = 1, \ldots, n$, prikazan je dijagramom na slici 2.3.1. U tom algoritmu je predviđena mogućnost da se provere ograničenja u odnosu na interval $(xmin_i, xmax_i)$, pri dodavanju priraštaja Δx_i za svako $i = 1, \ldots, n$. Implementacija numeričkog diferenciranja u paketu MATHEMATICA je vrlo jednostavna:

```
Numnabla[q_,prom_List,pvred_List,korak_List,xmin,xmax]:=
   Block[{i,n=Length[prom],j,x0=pvred,grad={},k1,k2},
    Do[x0[[i]]=N[x0[[i]]+korak[[i]]];
    If[N[x0[[i]]]>N[xmax[[i]]],
      k1=N[xmax[[i]]-pvred[[i]]]; x0[[i]]=N[xmax[[i]]],
      k1=N[korak[[i]]]
    ];
    qg=q;   Do[qg=qg/.prom[[j]]->x0[[j]],{j,n}];
    x0[[i]]=N[pvred[[i]]];   qd=q;
    x0[[i]]=N[x0[[i]]-korak[[i]]];
    If[N[x0[[i]]]<N[xmin[[i]]],
      k2=N[pvred[[i]]-xmin[[i]]]; x0[[i]]=N[xmin[[i]]],
      k2=N[korak[[i]]]];
      Do[qd=qd/.prom[[j]]->x0[[j]],{j,n}];
      pizv=N[(qg-qd)/(k1+k2)]; grad=Append[grad,N[pizv]];
      x0[[i]]=pvred[[i]],{i,n}
    ];
    N[grad]  ]
```

Numeričko izračunavanje gradijenta u paketu MATHEMATICA se ne razlikuje suštinski od odgovarajućih procedura u programskim jezicima FORTRAN ili C. Za nas je od prvenstvenog interesa simboličko diferenciranje i njegova primena u metodima optimizacije. U tu svrhu, u paketu MATHEMATICA napisana je sledeća funkcija *nabl*, kojom se za zadatu ciljnu funkciju Q u unutrašnjoj formi i za zadatu tačku $\mathbf{x}^{(0)} = x0$ formira gradijent $\nabla Q(x0)$:

```
nabl[q_,prom_List,x0_List]:=
   Block[{n=Length[prom],i,dqdx={}},
     Do[dqdx=Append[dqdx,D[q,prom[[i]]]],{i,n}];
     Do[dqdx=dqdx/.prom[[i]]->x0[[i]],{i,n}];
```

```
    Return[dqdx]
  ]
```

Izrazom

`Do[dqdx=Append[dqdx,D[q,prom[[i]]]],{i,n}];`

formira se vektor parcijalnih izvoda ciljne funkcije $Q(\mathbf{x})$, odnosno formira se gradijent $\nabla Q(\mathbf{x})$. U izrazu

`Do[dqdx=dqdx/.prom[[i]]->x0[[i]],{i,n}];`

izračunava se vrednost gradijenta $\nabla Q(\mathbf{x})$ u tački $x0$, tj. formira se vektor $\nabla Q(\mathbf{x}^{(0)}) = \nabla Q(x0)$.

U jeziku LISP nije komplikovana modifikacija poznatih procedura za pacijalne izvode, koje su date u referencama [12], [14], [44], [61], kako bi one postale primenljive na proizvoljan SCHEME aritmetički izraz. Takođe, poželjno je da se implementira nekoliko procedura za uprošćavanje izraza koji predstavljaju parcijalne izvode [61].

Simboličko diferenciranje izraza u prefiksnoj notaciji u LISPu je implementirano u funkciji *deriv*, ekstenzijom odgovarajućih funkcija iz [12], [14], [44], [61]. Ovim je verifikovana prednost **(1G)** i u LISPu:

```
(define (deriv l x)
   (cond ((number?  l) 0)
    ((atom?  l) (if (equal?  l x) 1 0))
    ((equal?  (car l)  'sin)
     (list '* (append '(cos) (cdr l)) (deriv (cadr l) x)))
    ((equal?  (car l) 'cos)
     (list '* (append '(- sin)x(cdr l))(deriv (cadr l) x)))
    ((equal?  (car l) 'log)
     (append '(/) (list (deriv (cadr l) x))(list (cadr l))))
    ((equal?  (car l) 'expt)
     (list '* (append '(*) (list (caddr l))
              (list (append '(expt) (list (cadr l))
                (list (append '(1-) (list (caddr l))))
              )  )
              )
              (deriv (cadr l) x))
         )
     ((equal?  (car l) '+)
     (izz l x))
     ((equal?  (car l) '-)
       (izr l x))
```

```
     ((equal?  (car l) '*)
       (izp l x))
     ((equal?  (car l) '/)
       (izk l x))
)     )

; izvod zbira
(define (izz l x)
   (let ((nl nl))
     (set!  nl (list '+))
     (set!  l (cdr l))
     (do ()
        ( (null?  l) nl)
        (set!  nl (append nl (list (deriv (car l) x) )))
        (set!  l (cdr l))
) )  )

; izvod razlike
(define (izr l x)
   (let ((nl nl))
   (set!  nl (cons '- nil))   (set!  l (cdr l))
     (do ()
    ( (null?  l) nl)
       (set!  nl (append nl (list (deriv (car l) x))))
       (set!  l (cdr l))
)     )  )

; izvod proizvoda
(define (izp l x)
   (let ((n n))
   (set!  l (cdr l)) (set!  n (length l))
     (cond
       ((= n 1) (deriv (car l) x))
       (else (list '+
   (append '(*) (list (deriv (car l) x)) (cdr l))
   (append '(*)
       (list (car l))
       (list (deriv (append '(*) (cdr l)) x))
)    ) ) )      )

; izvod kolicnika
```

```
(define (izk l x)
   (let ((n n))
    (set!  l (cdr l))
     (cond
       ((= n 1) (deriv (car l) x))
       ((= n 2)
         (list '/
           (list '-
             (append '(*) (list (deriv (car l) x))
    (list (cadr l)))
             (append '(*)
        (list (car l))
        (list (deriv (cadr l) x))
       )  )
      (append '(expt) (list (cadr l)) '(2))
       )     )
      (else (list '/
      (list '-
      (append '(*)
      (list (deriv (car l) x))
            (list (append '(*) (cdr l)))
       )
       x (append '(*)
         (list (car l))
         (list (deriv (append '(*) (cdr l)) x))
       )    )
       (append '(expt)(list (append '(*)(cdr l))) '(2)) )
)  ) )    )
```

U daljem tekstu sledi opis prednosti **(2G)** u LISPu. Neka izraz $(deriv\ f\ x_i)$ predstavlja poziv funkcije kojom se oformljuje simbolički parcijalni izvod funkcije $f(\mathbf{x})$ po nezavisnoj promenljivoj x_i, $1 \leq i \leq n$. Takođe, neka je ciljna funkcija predstavljena u unutrašnjoj formi q, koja je zadata listom

$$(Q(x_1, \ldots, x_n) \qquad (x_1, \ldots, x_n))\,.$$

Tada se gradijent ciljne funkcije $\nabla Q(\mathbf{x})$ može predstaviti vektorom, koji je označen simbolom *nabla* i čija je i-ta koordinata lambda-izraz koji odgovara pozivu funkcije $(deriv\ Q\ x_i)$, $0 \leq i \leq n$:

```
(define (symbder q)
```

```
(let (nabla (make-vector (length (cadr q))))
   (do ((i 0))
      ((= n i) nabla)
      (set!  prom (vector-ref varg i))
      (vector-set!  nabla i (eval (list 'lambda (cadr q)
                     (deriv (car q) prom))))
      (set!  i (+ i 1))
) )   )
```

Elementi vektora $grad = \nabla Q(\mathbf{x}^{(k)})$, koji predstavlja vrednost gradijenta u zadatoj tački (vektoru) $vx0 = \mathbf{x}^{(k)} = \left(x_0^{(k)}, \ldots, x_n^{(k)}\right)$, mogu da se generišu primenjujući redom lambda-izraze sadržane u elementima vektora *nabla* na odgovarajuće vektore $vx0$:

```
(define (nabl q vx0)
  (let (grad (make-vector (length (cadr q))))
    (do ((i 0))
        ((= n i) grad)
        (vector-set!  grad i (apply (vector-ref nabla i)
                                    (vector->list vx0)))
        (set!  i (+ i 1))
)  ) )
```

Sada sledi opis nekoliko pomoćnih funkcija. Procedurom *limit* koordinate vektora $\mathbf{x}$ održavaju se unutar zadatih granica $[\mathbf{xmin}, \mathbf{xmax}]$:

```
limit[x_List,xmin_List,xmax_List]:=
   Block[{x0=x,n=Length[x]},
     Do[Which[x0[[i]]<xmin[[i]],x0[[i]]=xmin[[i]],
              x0[[i]]>xmax[[i]],x0[[i]]=xmax[[i]]
         ],{i,n} ];
     Return[x0] ]
```

Za zadati vektor $vx = vx_i$, $i = 1, \ldots, n$, njegova norma $\|vx\| = \sqrt{\sum_{i=1}^{n} vx_i^2}$ može se izračunati na sledeći način:

```
norma[vx_List]:=Block[{n=Length[vx],i,rezultat=0},
     rezultat=Sum[vx[[i]]^2,{i,n}]; rezultat=Sqrt[rezultat];
     Return[rezultat]    ]
```

2.4. Algoritmi za gradijentne metode prvog reda

Koristeći funkcije za konstrukciju gradijenta ciljne funkcije i na neki način određenu veličinu koraka α_k, može se u k-tom koraku iteracije definisati tranzicija iz tačke $\mathbf{x}^{(k)}$ u sledeću aproksimaciju $\mathbf{x}^{(k+1)}$, koristeći smer gradijenta u tački $\mathbf{x}^{(k)}$ (za maksimizaciju) ili suprotan smer (za minimizaciju):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \pm \alpha_k \frac{\nabla Q\left(\mathbf{x}^{(k)}\right)}{\|\nabla Q\left(\mathbf{x}^{(k)}\right)\|}, \qquad \alpha_k \in \mathbb{R}. \tag{2.4.1}$$

Generalno, primenjivana su dva opšta metoda za izbor veličine koraka α_k. U jednoj grupi metoda, koristi se neoptimalna vrednost koraka, koja može biti fiksirana kroz sve iteracije ili može biti promenljiva u toku iteracija. U metodima *najstrmijeg pada* koristi se optimalna vrednost koraka, što znači da veličinom koraka obezbeđuje optimalna vrednost ciljne funkcije. U ovim metodima, ciljna funkcija $Q(\mathbf{x})$ se transformiše u novu funkciju

$$f(\alpha) = Q\left(\mathbf{x}^{(k)} \pm \alpha \frac{\nabla Q\left(\mathbf{x}^{(k)}\right)}{\|\nabla Q\left(\mathbf{x}^{(k)}\right)\|}\right).$$

Koristeći novu funkciju $f(\alpha)$, izračunava se $\alpha_k = \min\limits_{\alpha} F(\alpha)$ nekim od metoda jednodimenzionalne optimizacije, a zatim se nova aproksimacija $\mathbf{x}^{(k+1)}$, u odnosu na staru $x^{(k)}$, formira prema (2.4.1).

Ako se koriste neoptimalne (fiksne ili promenljive) vrednosti skalara α_k, te vrednosti moraju biti pažljivo kontrolisane da bi se izbeglo neočekivano rašćenje (ili opadanje) ciljne funkcije ili pak veliki broj iteracija. Neočekivano rašćenje (ili opadanje) ciljne funkcije se javlja ako je korak nepodesno veliki s obzirom na blizinu ekstremne tačke. S druge strane, korišćenje malog koraka uzrokuje nepotrebno veliki broj iteracija.

2.4.1. OSNOVNI GRADIJENTNI METOD

Osnovni gradijentni metod je poznat i pod nazivom *modifikacija Cauchyevog metoda*. U originalnom Cauchyevom metodu obično se dosta vremena potroši na jednodimenzionalnu optimizaciju kojom se nalazi optimalna dužina koraka α_k. Takođe, potrebna je izvesna programerska veština za pisanje odgovarajućih programa kojom se funkcija $Q(\mathbf{x})$ prevodi u funkciju $f(\alpha)$. Da bi se izbegle ove poteškoće koristi se osnovni gradijentni metod, koji je

zasnovan na konstantnom parametru koraka $\alpha_k = h$ u svim iteracijama. Preciznije, u slučaju maksimuma, iteracije se računaju prema pravilu

$$x_i^{k+1} = x_i^k \pm h \cdot \frac{\partial Q}{\partial x_i}, \qquad i = 1, \dots, n,\ k = 0, 1, \dots, \tag{2.4.2}$$

Za korak h treba uzeti dovoljno malu vrednost. Jedan od kriterijuma za izbor veličine koraka je

$$0 < h < \frac{2}{\lambda_M},$$

gde je λ_M najveća sopstvena vrednost Hesseove matrice funkcije Q. Ako ovaj metod u slučaju konvergencije, konvergira znatno sporije od originalnog Cauchyevog metoda. Prednost ovakvog metoda je u tome što je izbegnuta konstrukcija nove funkcije i odgovarajuća jednodimenzionalna optimizacija.

Ukazaćemo sada na implementaciju osnovnog gradijentnog metoda.

U algoritmu se koristi ranije opisani potprogram *nabl* za formiranje koordinata gradijenta u zadatoj tački i potprogram *limit* za proveru ograničenja oblika

$$xmin_i \le x_i \le xmax_i, \qquad i = 1, \dots, n.$$

Ulazne veličine:

$q\_$, $prom\_$: ciljna funkcija i lista njenih parametara;

$\mathbf{x}^{(0)} = x0 = (x_1^0, \dots, x_n^0) = x01$: izabrana početna tačka;

$h\_$: lista koja predstavlja fiksirani parametar koraka $\mathbf{h} = \{h_1, \dots, h_n\}$;

$xmin\_$, $xmax\_$: vektori

$$\mathbf{xmin} = \{xmin_1, \dots, xmin_n\} \quad \text{i} \quad \mathbf{xmax} = \{xmax_1, \dots, xmax_n\}$$

koji određuju granice oblasti optimizacije;

$eps\_$: zadata tačnost.

Lokalne promenljive:

izb: parametar koji određuje lokalizaciju minimuma ($izb = 1$), odnosno maksimuma ($izb = 2$).

Algoritam se može iskazati kroz sledeće korake:

Korak 1. Izračunati $q0 = Q(x0)$, $\nabla Q(x0)$, $S = \|\nabla Q(x0)\|$;

Korak 2. `While` ciklus, koji se prekida kada je ispunjen uslov

$$S = \|\nabla Q(x0)\| < eps.$$

Unutar ciklusa izvršiti sledeće korake:

Korak 2.1. Izračunati novu tačku optimuma $x0 = (x_1^0, \ldots, x_n^0)$:

$$x_i^0 = x_i^0 \pm \frac{h_i}{S} \cdot \frac{\partial Q}{\partial x_i}, \qquad i = 1, \ldots, n.$$

Korak 2.2. Izračunati $q0 = Q(x0)$, $\nabla Q(x0)$, $S = \|\nabla Q(x0)\|$.

Korak 3. Izlazne veličine su $x0$ i $Q(x0)$.

Programska realizacija ovog algoritma je veoma jednostavna, ali ona ima određene nedostatke kao što su: veliki broj potrebnih izračunavanja i konstantan parametar koraka koji implicira komplikovano ispunjenje kriterijuma za prekid pretraživanja u slučaju ciljnih funkcija sa velikom osetljivošću ekstremuma. U velikom broju slučaja, konstantni korak postaje nepodesno veliki u okolini ekstremuma.

```
OsnGrad[q_,prom_List,x01_List,h_List,
   xmin_List,xmax_List,eps_]:=
   Block[{i,n,q0,qm,xm,nabla,s,x0=x01,h0=h,it=0,
   izb,Lista={}},
     izb=Input["Zelite li maximum(2) ili minimum(1)?"];
     n=Length[x0];
     q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],i,n]; q0=N[q0];
     qm=q0; xm=x0;
     Lista=Append[Lista,xm];
     nabla=nabl[q,prom,x0];
     s=norma[nabla];
     While[N[s]>=eps && it<=100,
        If[izb==2,
           Do[x0[[i]]+=h0[[i]]*nabla[[i]]/N[s],{i,n}],
           Do[x0[[i]]-=h0[[i]]*nabla[[i]]/N[s],{i,n}]
        ];
     x0=limit[x0,xmin,xmax]; x0=N[x0];
        q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}]; q0=N[q0];
     If[(izb==1 && q0<qm) || (izb==2 && q0>qm),
          qm=q0; xm=x0; Lista=Append[Lista,xm]
     ];
     it+=1;  nabla=nabl[q,prom,x0];   s=norma[nabla]
     ];
     {xm,qm, Lista}
```

```
]
```

Implementacije osnovnog gradijentnog metoda u LISPu opisana je u radovima [49], [50].

2.4.2. MODIFIKACIJA OSNOVNOG GRADIJENTNOG METODA

U osnovnom gradijentnom metodu, u svakoj iteraciji je neophodno da se izračunavaju izvodi ciljne funkcije. Numeričko izračunavanje parcijalnih izvoda ciljne funkcije Q, koja zavisi od n parametara, zahteva minimalno $n+1$ izračunavanja vrednosti $Q(\mathbf{x})$. Za veliki broj upravljačkih parametara n broj izračunavanja brzo raste, dok brzina konvergencije znatno opada. S druge strane, zbog već opisanih poteškoća oko implementacije simboličkog diferenciranja, modifikovani gradijentni metod predstavlja pokušaj da se izbegne izračunavanje gradijenta u svakoj iteraciji. Cilj ovog metoda je da se smanji broj izračunavanja $Q(\mathbf{x})$, odnosno da se simboličko diferenciranje što ređe koristi. Posle izračunavanja gradijenta u tački $\mathbf{x}^{(0)}$, iteracije se izvršavaju u tom pravcu, sve dok se ne dostigne maksimum za taj pravac. Zatim se ponovo izračunava gradijent i lokalizuje ekstremum u novom smeru.

Algoritam modifikacije gradijentnog metoda može se iskazati u sledećim koracima:

Korak 1. U zadatoj početnoj tački $\mathbf{x}^{(0)}$ izračunava se gradijent ciljne funkcije, tj. parcijalni izvodi

$$\frac{\partial Q(\mathbf{x}^{(0)})}{\partial x_i}, \qquad i = 1, \ldots, n.$$

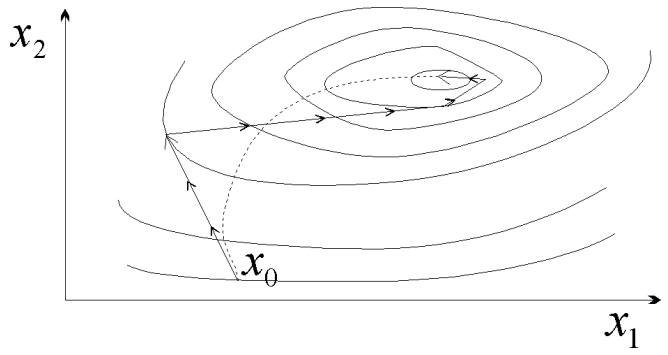
Korak 2. Proverava se neki od kriterijuma za prekid pretraživanja, na primer,

$$S = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial Q(\mathbf{x}^{(0)})}{\partial x_i} \right)^2} \leq \varepsilon.$$

Ako je uslov ispunjen, završiti algoritam, inače preći na sledeći korak.

Korak 3. Izvršavati iterativne korake jedan za drugim, sve dok se vrednosti funkcije poboljšavaju, bez izračunavanja gradijenta, što odgovara formuli

$$x_i^{(k+1)} = x_i^{(k)} \pm \frac{h_i^{(k)}}{S} \cdot \frac{\partial Q(\mathbf{x}^{(0)})}{\partial x_i}, \qquad i = 1, \ldots, n.$$



Sl. 2.4.1

Kada se vrednost ciljne funkcije pogorša, izračunati gradijent u dostignutoj tački, a zatim preći na *Korak 2.*

Na slici 2.4.1 prikazano je kretanje prema maksimumu saglasno ovom metodu. Isprekidanom linijom je označeno pretraživanje prema osnovnom gradijentnom metodu. Put prema ekstremumu koji koristi osnovni gradijentni metod je kraći i ortogonalan je na nivoske linije površi. Međutim, osnovni gradijentni metod zahteva da se u svakoj iteraciji formira gradijent ciljne funkcije, i saglasno tome potrebno je više vremena u svakoj iteraciji u odnosu na modifikovani gradijentni metod.

Implementacija modifikacije osnovnog gradijentnog metoda:

Ulazne veličine i *formalni parametri* su kao i u proceduri za osnovni gradijentni metod.

Korak 1. Izračunati $q0 = Q(x0)$, gradijent $nabla = \nabla Q(x0)$ i normu gradijenta $S = \|\nabla Q(x0)\| = norma(nabla)$.

Korak 2. `While` ciklus, koji se prekida kada je ispunjen uslov

$$S = \|\nabla Q(x0)\| < eps.$$

Unutar ciklusa izvršiti sledeće korake:

Korak 2.1. Izračunati novu tačku optimuma $x1 = (x_1^1, \ldots, x_n^1)$:

$$x_i^1 = x_i^0 \pm \frac{h_i}{S} \cdot \frac{\partial Q}{\partial x_i}, \qquad i = 1, \ldots, n. \tag{2.4.5}$$

Korak 2.2. $x1 = limit(x1, xmin, xmax)$.

Korak 2.3. Izračunati $q1 = Q(x1)$.

Korak 2.4. `While` ciklus, koji se izvršava sve dok je ispunjen jedan od uslova `izb==2 && q1>q0` ili `izb==1 && q1<q0`.

Unutar ciklusa izvršiti sledeće korake:

Korak A. Postaviti $q0 = q1$, $x0 = x1$.

Korak B. Izračunati novu tačku optimuma $x1$ prema (2.4.5).

Korak C. Izračunati $x1 = limit(x1, xmin, xmax)$ i $q1 = q(x1)$.

Korak 3. Izlazne veličine su $x1$ i $Q(x1)$.

Odgovarajuća programska realizacija u paketu MATHEMATICA izgleda:

```
ModGrad[q_,prom_List,x01_List,h_List,
   xmin_List,xmax_List,eps_]:=
   Block[{i,n,nabla,s=eps+1,x0=x01,q0,xm,qm,
   it=0,h0=h,izb, Lista={} },
      izb=Input["Zelite li maximum(2) ili minimum(1)?"];
      n=Length[x0];
      q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
      q0=N[q0];
      qm=q0; xm=x0;
      Lista=Append[Lista,xm];
      While[N[s]>=eps && it<=100,
        nabla=nabl[q,prom,x0];
        s=norma[nabla];
        If[izb==2,
           Do[x1[[i]]=x0[[i]]+h0[[i]]*nabla[[i]]/s,{i,n}],
           Do[x1[[i]]=x0[[i]]-h0[[i]]*nabla[[i]]/s,{i,n}]
        ];
        x0=limit[x0,xmin,xmax]; x0=N[x0];
        q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}]; q0=N[q0];
        it+=1;
        While[(izb==1 && q0<qm)||(izb==2 && q0>qm),
            qm=q0; xm=x0; Lista=Append[Lista,xm];
            If[izb==2,
               Do[x0[[i]]=x0[[i]]+h0[[i]]*nabla[[i]],{i,n}],
               Do[x0[[i]]=x0[[i]]-h0[[i]]*nabla[[i]],{i,n}]
            ];
            x0=limit[x0,xmin,xmax]; x0=N[x0];
            q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],i,n];
            it+=1
```

```
        ]
      ];
      {xm,qm, Lista}
  ]
```

2.4.3. GRADIJENTNI METODI SA AUTOMATSKOM KOREKCIJOM KORAKA

U algoritmu za osnovni gradijentni metod, suštinski nedostatak predstavlja izbor parametra koraka u svim iteracijama, koje su definisane formulama

$$x_i^{(k+1)} = x_i^{(k)} \pm h_i \frac{\dfrac{\partial Q(\mathbf{x}^{(k)})}{\partial x_i}}{\|\nabla Q(\mathbf{x}^{(k)})\|}, \qquad i = 1, \dots, n.$$

Brzina konvergencije zavisi od vrednosti promenljivih h_i, $i = 1, \dots, n$ za svaki upravljački parametar. Pri izboru manjih vrednosti za konstantne parametre h_i, broj iteracija potrebnih za lokalizaciju ekstremuma znatno raste i zahteva veliki broj izračunavanja vrednosti ciljne funkcije. Pri korišćenju većih vrednosti za h_i, u oblasti ekstremuma mogu se dogoditi "preskakanja", naročito u slučaju jako izraženog ekstremuma. Ovo je razlog za uvođenje automatske korekcije parametra koraka. Najčešće se koriste dve modifikacije koje će ovde biti opisane.

A. Jedan od algoritama za automatsku korekciju koraka koristi smanjenje parametra koraka, koje je definisano na sledeći način:

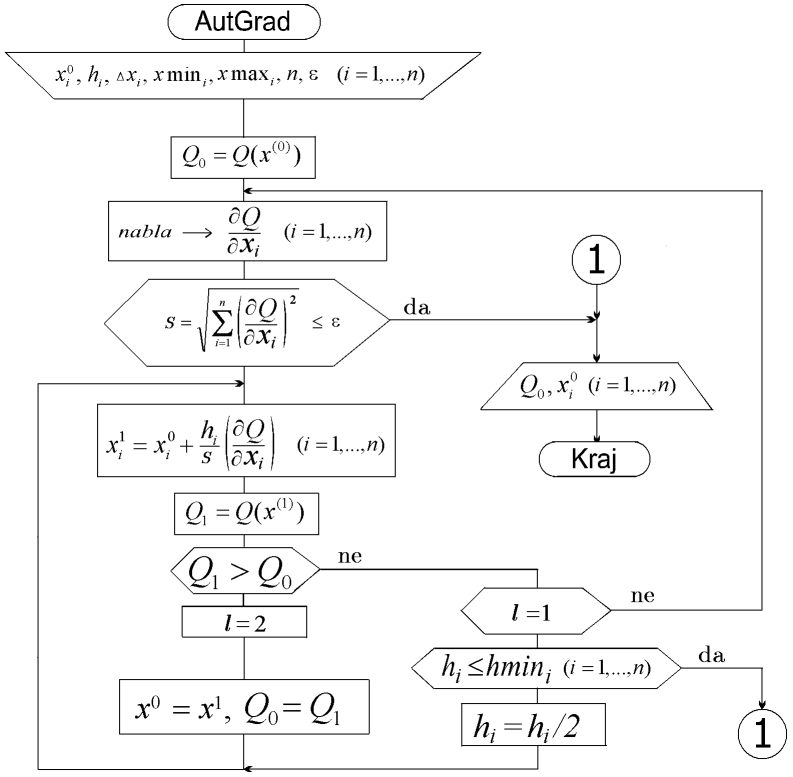
$$h_i^{(k+1)} = \begin{cases} h_i^{(k)} & , \text{ ako } Q^{(k)} > Q^{(k-1)}, \\ 0.5h_i^{(k)} & , \text{ ako } Q^{(k)} \leq Q^{(k-1)}, \end{cases}$$

pri čemu je k redni broj iteracije. Ako je korak u gradijentnom pravcu uspešan, dužina koraka ostaje nepromenjena, dok u slučaju neuspešnog koraka njegova dužina se skraćuje na polovinu.

Kod ovog metoda mogu se predvideti dva kriterijuma za prekid pretraživanja. Prvi je poznati kriterijum (2.1.5) od ranije, a drugi je da parametri koraka postanu manji od unapred zadatih vrednosti $hmin_i$, $i = 1, \dots, n$. Već je ranije napomenuto da je upotreba i jednog i drugog kriterijuma za prekid algoritma sadrži opasnost da se pretraživanje prekine dalje od ekstremuma.

Blok dijagram algoritma je prikazan na slici 2.4.2.

Odgovarajuća programska realizacija u paketu MATHEMATICA izgleda:



Sl. 2.4.2

```
AutGrad[q_,var_List,x_List,h_List,hmin_List,
   xmin_List,xmax_List,eps_]:=
   Block[{x0=x,q0,xm,qm,nabla,l=1,s=eps+1,h0=h,lg=True,
   ind=True,i,n,it=0,izb, Lista={} },
       izb=Input["Zelite li maximum(2) ili minimum(1)?"];
       n=Length[var];
       q0=q; Do[q0=q0/.var[[i]]->x0[[i]],{i,n}];
       xm=x0; qm=q0;
       Lista=Append[Lista,xm];
       While[N[s]>=eps && lg && it<=100,
         If[ind,
               nabla=nabl[q,var,x0];
```

```
            l=1;
            s=norma[nabla];
        ];
        If[izb==2,
         Do[x0[[i]]=x0[[i]]+h0[[i]]*nabla[[i]]/N[s],{i,n}],
         Do[x0[[i]]=x0[[i]]-h0[[i]]*nabla[[i]]/N[s],{i,n}]
         ];
         x0=N[x0]; x01=limit[x0,xmin,xmax]; x0=N[x0];
         q0=q;  Do[q0=q0/.var[[i]]->x0[[i]],{i,n}];
         it+=1;
         If[(izb==2 && q0>qm) || (izb==1 && q0<qm),
             l=2; xm=x0; qm=q0;
             Lista=Append[Lista,xm];ind=False,
             If[l==1,
                lg=False;
                Do[Which[h0[[i]]>hmin[[i]], lg=True],{i,n}];
                If[lg, Do[h0[[i]]/=2,{i,n}] ];
                ind=False,
                ind=True;
             ]
         ]
       ];
       {xm,qm, Lista}
    ]
```

Pri primeni algoritma sa automatskom korekcijom koraka, konvergencija je brža ako pretraživanje započne sa nekim relativno velikim početnim korakom $h0_i$, $i = 1, \ldots, n$.

B. *Korekcija koraka na osnovu vrednosti ciljne funkcije u poslednje tri iteracije*. U slučaju maksimizacije uzimamo

$$h_i^{(k+1)} = \begin{cases} 2h_i^{(k)}, & Q^{(k)} > Q^{(k-1)} > Q^{(k-2)}, \\ 0.5h_i^{(k)}, & Q^{(k)} \leq Q^{(k-1)}, \\ h_i^{(k)}, & \text{u ostalim slučajevima.} \end{cases}$$

Ako su poslednja tri koraka bila uspešna h_i se uvećava dva puta, ako je poslednji korak bio neuspešan h_i se smanjuje dva puta, dok u ostalim slučajevima korak ostaje nepromenjen.

Svi algoritmi sa automatskom korekcijom koraka imaju suštinski nedostatak. U slučaju nekih ciljnih funkcija, na primer kod tzv. *jaružnih* može da se izvrši lokalizacija dalje od ekstremuma zbog prevremenog ispunjenja kriterijuma $h_i \leq hmin_i$ $(i = 1, \ldots, n)$ za prekid pretraživanja.

Numerički eksperimenti.

```
In[1]:= OsnGrad[x^ 2+y^2,{x,y},{1.,-1.2},{0.1,0.2}, {-5,-5},{5,5},0.001]    (*minimum*)
Out[1]= {{0.011051, 0.00153783}, 0.000124489,
   {{1., -1.2}, {0.935982, -1.04636}, {0.869311, -0.897291}, {0.799729, -0.753648},
   {0.726953, -0.616482}, {0.650685, -0.487127}, {0.570633, -0.367266},
   {0.486544, -0.259025}, {0.398273, -0.165039}, {0.305891, -0.0884749},
   {0.209829, -0.0329053}, {0.111036, -0.00192001}, {0.011051, 0.00153783}}}
In[2]:= OsnGrad[x^ 2+y^2,{x,y},{1.,-1.2},{0.1,0.2}, {-5,-5},{5,5},0.001]    (*maksimum*)
Out[2]= {{5., -5.}, 50., {{1., -1.2}, {1.06402, -1.35364}, {1.12582, -1.51088},
   {1.18557, -1.67126}, {1.24343, -1.83438}, {1.29953, -1.99993}, {1.35402, -2.16764},
   {1.407, -2.33726}, {1.45857, -2.50861}, {1.50884, -2.68151}, {1.55788, -2.85581},
   {1.60577, -3.03139}, {1.65258, -3.20812}, {1.69837, -3.38592}, {1.7432, -3.56469},
   {1.78713, -3.74436}, {1.83021, -3.92485}, {1.87247, -4.10611}, {1.91396, -4.28809},
   {1.95472, -4.47072}, {1.99478, -4.65397}, {2.03418, -4.83779}, {2.07294, -5.},
   {2.11124, -5.}, {2.15014, -5.}, {2.18964, -5.}, {2.22976, -5.}, {2.27048, -5.},
   {2.31183, -5.}, {2.3538, -5.}, {2.39639, -5.}, {2.43961, -5.}, {2.48346, -5.},
   {2.52795, -5.}, {2.57307, -5.}, {2.61882, -5.}, {2.66522, -5.}, {2.71226, -5.},
   {2.75994, -5.}, {2.80827, -5.}, {2.85724, -5.}, {2.90685, -5.}, {2.95711, -5.},
   {3.00802, -5.}, {3.05957, -5.}, {3.11176, -5.}, {3.1646, -5.}, {3.21808, -5.},
   {3.2722, -5.}, {3.32696, -5.}, {3.38236, -5.}, {3.43839, -5.}, {3.49505, -5.},
   {3.55235, -5.}, {3.61026, -5.}, {3.6688, -5.}, {3.72796, -5.}, {3.78774, -5.},
   {3.84812, -5.}, {3.90911, -5.}, {3.9707, -5.}, {4.03289, -5.}, {4.09567, -5.},
   {4.15904, -5.}, {4.22299, -5.}, {4.28752, -5.}, {4.35261, -5.}, {4.41827, -5.},
   {4.48449, -5.}, {4.55125, -5.}, {4.61857, -5.}, {4.68642, -5.}, {4.75481, -5.},
   {4.82372, -5.}, {4.89315, -5.}, {4.96309, -5.}, {5., -5.}}}
In[3]:= ModGrad[x^ 2+y^2,{x,y},{1.,-1.2},{0.1,0.2}, {-5,-5},{5,5},0.001]    (*minimum*)
Out[3]= {{1.20669 10^-11 , -0.0289502}, 0.000838114,
   {{1., -1.2}, {0.935982, -1.04636}, {0.871963, -0.892711}, {0.807945, -0.739067},
   {0.743926, -0.585423}, {0.679908, -0.431779}, {0.615889, -0.278134},
   {0.551871, -0.12449}, {0.487852, 0.029154}, {0.423834, 0.182798},
   {0.286772, 0.199846}, {0.213729, 0.0632486}, {0.140686, -0.0733483},
   {0.0369759, -0.0195819}, {0.00261738, -0.0290823}, {-0.000605112, -0.0289481},
   -6 {-0.0000564267, -0.0289503}, {0.0000129496, -0.0289502}, {1.20761 10 , -0.0289502},
   {-2.77137 10^-7, -0.0289502}, {-2.58443 10^-8, -0.0289502},
   {5.93109 10^-9, -0.0289502}, {5.53101 10^-10, -0.0289502},
   {-1.26933 10^-10, -0.0289502}, {1.20669 10^-11, -0.0289502}}}
In[4]:= ModGrad[x^ 2+y^2,{x,y},{1.,-1.2},{0.1,0.2}, {-5,-5},{5,5},0.001]
   (*maksimum*)
Out[4]= {{5., -5.}, 50., {{1., -1.2}, {1.06402, -1.35364}, {1.12804, -1.50729},
```

{1.19206, -1.66093}, {1.25607, -1.81458}, {1.32009, -1.96822}, {1.38411, -2.12187},
{1.44813, -2.27551}, {1.51215, -2.42915}, {1.57617, -2.5828}, {1.64018, -2.73644},
{1.7042, -2.89009}, {1.76822, -3.04373}, {1.83224, -3.19738}, {1.89626, -3.35102},
{1.96028, -3.50466}, {2.0243, -3.65831}, {2.08831, -3.81195}, {2.15233, -3.9656},
{2.21635, -4.11924}, {2.28037, -4.27289}, {2.34439, -4.42653}, {2.40841, -4.58017},
{2.47242, -4.73382}, {2.53644, -4.88746}, {2.60046, -5.}, {2.66448, -5.},
{2.7285, -5.}, {2.79252, -5.}, {2.85653, -5.}, {2.92055, -5.}, {2.98457, -5.},
{3.04859, -5.}, {3.11261, -5.}, {3.17663, -5.}, {3.24065, -5.}, {3.30466, -5.},
{3.36868, -5.}, {3.4327, -5.}, {3.49672, -5.}, {3.56074, -5.}, {3.62476, -5.},
{3.68877, -5.}, {3.75279, -5.}, {3.81681, -5.}, {3.88083, -5.}, {3.94485, -5.},
{4.00887, -5.}, {4.07289, -5.}, {4.1369, -5.}, {4.20092, -5.}, {4.26494, -5.},
{4.32896, -5.}, {4.39298, -5.}, {4.457, -5.}, {4.52101, -5.}, {4.58503, -5.},
{4.64905, -5.}, {4.71307, -5.}, {4.77709, -5.}, {4.84111, -5.}, {4.90512, -5.},
{4.96914, -5.}, {5., -5.}}}

In[5]:= AutGrad[x^ 2+y^2,{x,y},{1.,-1.2},{0.1,0.2},{0.001,0.002}, {-5,-5},{5,5},0.001]
(*minimum*)

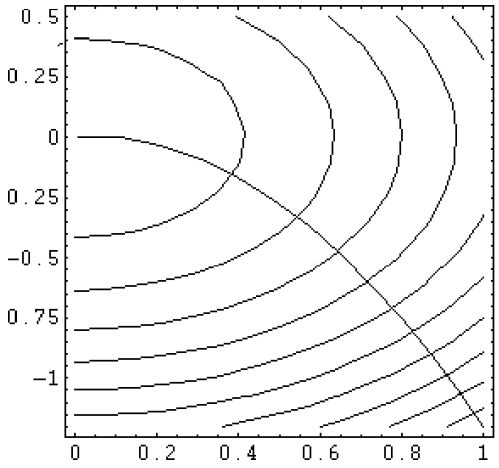
Out[5]= {{-2.29231 10^{-38}, -0.0289502}, 0.000838114,
{{1., -1.2}, {0.935982, -1.04636}, {0.871963, -0.892711}, {0.807945, -0.739067},
{0.743926, -0.585423}, {0.679908, -0.431779}, {0.615889, -0.278134},
{0.551871, -0.12449}, {0.487852, 0.029154}, {0.423834, 0.182798},
{0.286772, 0.199846}, {0.213729, 0.0632486}, {0.140686, -0.0733483},
{0.0369759, -0.0195819}, {0.00261738, -0.0290823}, {-0.000605112, -0.0289481},
{-0.0000564267, -0.0289503}, {0.0000129496, -0.0289502},{1.20761 10^{-6}, -0.0289502},
{-2.77137 10^{-7}, -0.0289502}, {-2.58443 10^{-8} , -0.0289502},
{5.93109 10^{-9} , -0.0289502}, {5.53101 10^{-10} , -0.0289502},
{-1.26933 10^{-10}, -0.0289502}, {-1.18371 10^{-11}, -0.0289502},
{2.71652 10^{-12} , -0.0289502}, {2.53328 10^{-13}, -0.0289502},
{-5.8137 10^{-14} , -0.0289502}, {-5.42154 10^{-15}, -0.0289502},
{1.2442 10^{-15} , -0.0289502}, {1.16028 10^{-16}, -0.0289502},
{-2.66275 10^{-17} , -0.0289502}, {-2.48314 10^{-18}, -0.0289502},
{5.69863 10^{-19},-0.0289502},{5.31423 10^{-20}, -0.0289502},
{-1.21958 10^{-20} , -0.0289502}, {-1.13731 10^{-21}, -0.0289502},
{2.61005 10^{-22} , -0.0289502}, {2.43399 10^{-23} , -0.0289502},
{-5.58584 10^{-24} , -0.0289502}, {-5.20905 10^{-25}, -0.0289502},
{1.19544 10^{-25}, -0.0289502}, {1.1148 10^{-26}, -0.0289502},
{-2.55839 10^{-27}, -0.0289502}, {-2.38582 10^{-28}, -0.0289502},
{5.47528 10^{-29}, -0.0289502}, {5.10595 10^{-30} , -0.0289502},
{-1.17178 10^{-30},-0.0289502}, {-1.09274 10^{-31} , -0.0289502},
{2.50776 10^{-32} , -0.0289502}, {2.3386 10^{-33}, -0.0289502},
{-5.36691 10^{-34} , -0.0289502}, {-5.00489 10^{-35}, -0.0289502},
{1.14859 10^{-35}, -0.0289502}, {1.07111 10^{-36}, -0.0289502},
{-2.45812 10^{-37}, -0.0289502}, {-2.29231 10^{-38}, -0.0289502}}}

In[6]:= AutGrad[x^ 2+y^2,{x,y},{1.,-1.2},{0.1,0.2},{0.001,0.002}, {-5,-5},{5,5},0.001]
(*maksimum*)

Out[6]= {{5., -5.}, 50., {{1., -1.2}, {1.06402, -1.35364}, {1.12804, -1.50729},
{1.19206, -1.66093}, {1.25607, -1.81458}, {1.32009, -1.96822}, {1.38411, -2.12187},

{1.44813, -2.27551}, {1.51215, -2.42915}, {1.57617, -2.5828}, {1.64018, -2.73644},
{1.7042, -2.89009}, {1.76822, -3.04373}, {1.83224, -3.19738}, {1.89626, -3.35102},
{1.96028, -3.50466}, {2.0243, -3.65831}, {2.08831, -3.81195}, {2.15233, -3.9656},
{2.21635, -4.11924}, {2.28037, -4.27289}, {2.34439, -4.42653}, {2.40841, -4.58017},
{2.47242, -4.73382}, {2.53644, -4.88746}, {2.60046, -5.}, {2.66448, -5.},
{2.7285, -5.}, {2.79252, -5.}, {2.85653, -5.}, {2.92055, -5.}, {2.98457, -5.},
{3.04859, -5.}, {3.11261, -5.}, {3.17663, -5.}, {3.24065, -5.}, {3.30466, -5.},
{3.36868, -5.}, {3.4327, -5.}, {3.49672, -5.}, {3.56074, -5.}, {3.62476, -5.},
{3.68877, -5.}, {3.75279, -5.}, {3.81681, -5.}, {3.88083, -5.}, {3.94485, -5.},
{4.00887, -5.}, {4.07289, -5.}, {4.1369, -5.}, {4.20092, -5.}, {4.26494, -5.},
{4.32896, -5.}, {4.39298, -5.}, {4.457, -5.}, {4.52101, -5.}, {4.58503, -5.},
{4.64905, -5.}, {4.71307, -5.}, {4.77709, -5.}, {4.84111, -5.}, {4.90512, -5.},
{4.96914, -5.}, {5., -5.}}}

Rezultati dobijeni paketom MATHEMATICA, kroz zahteve date u Out[1], Out[3] i Out[5], prikazani su na slikama 2.4.3, 2.4.4 i 2.4.5, respektivno.

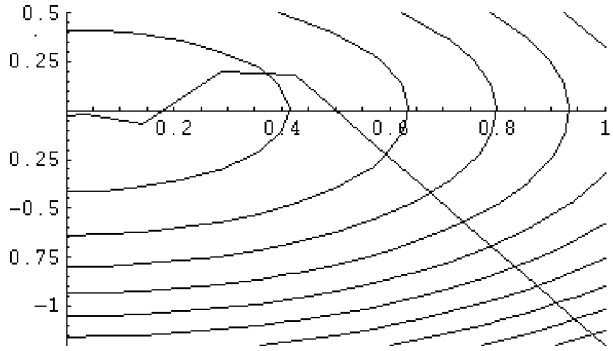


Sl. 2.4.3

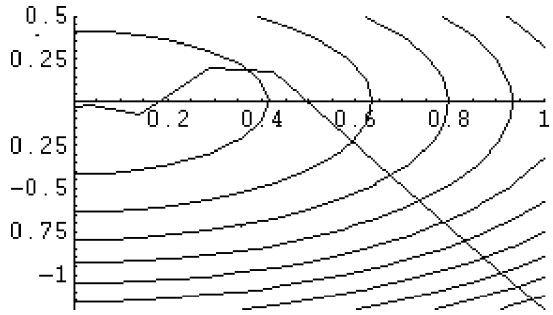
2.4.4. CAUCHYEV METOD NAJSTRMIJEG PADA

Ovaj metod se razlikuje od osnovnog gradijentnog algoritma po tome što se parametar α_k koraka određuje kao rešenje jednodimenzionalnog optimizacionog problema

$$F(\alpha_k) = \min_{\alpha} Q\left(\mathbf{x}^{(k)} \pm \alpha \nabla Q(\mathbf{x}^{(k)})\right) = \min_{\alpha} F(\alpha), \qquad \alpha > 0. \tag{2.4.3}$$



Sl. 2.4.4



Sl. 2.4.5

Napomene. (i) U Cauchyevom metodu se može koristiti normalizovani gradijent

$$\mathbf{u}^k = \frac{\nabla Q(\mathbf{x}^{(k)})}{\|\nabla Q(\mathbf{x}^{(k)})\|}, \qquad k = 0, 1, \ldots$$

(ii) Takođe, može se dogoditi da funkcija $F(h)$ ima nekoliko lokalnih minimuma. Dogovorno se može uzeti lokalni minimum koji je najbliži polaznoj tački.

(iii) Za prekid iterativnog procesa može se uzeti jedan od sledeća dva kriterijuma:

1° Norma gradijenta je manja od zadatog broja ε;

2° Kriterijum koji detektuje smanjenje optimalne vrednosti α

$$\|\alpha_k \mathbf{u}^k\| \leq \|L\alpha_0 \mathbf{u}^0\|,$$

tj.

$$\alpha_k \leq L\alpha_0,$$

gde je $0 < L < 1$ izabrani realan broj.

Algoritam *Cauchyevog metoda* može se iskazati kroz sledeće korake:

Korak 1. Specificirati polaznu aproksimaciju $\mathbf{x}^{(0)}$ i kriterijum zaustavljanja, tj. realan broj $0 < L < 1$. Staviti $k = 0$.

Korak 2. Izračunati $Q(\mathbf{x}^k)$ i $\nabla Q(\mathbf{x}^k)$.

Korak 3. Izračunati normalizovani gradijent:

$$\mathbf{u}^k = \frac{\nabla Q(\mathbf{x}^{(k)})}{\|\nabla Q(\mathbf{x}^{(k)})\|}.$$

Korak 4. Rešiti problem jednodimenzionalne optimizacije (2.4.3), pri čemu se u slučaju maksimizacije uzima znak +, odnosno znak - za slučaj minimizacije.

Korak 5. Izračunati novu aproksimaciju $\mathbf{x}^{(k+1)}$, prema formuli

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \pm \alpha_k \mathbf{u}^k.$$

Korak 6. Ako je $\alpha_k \leq L\alpha_0$, proces se zaustavlja. Približna vrednost minimuma je $\mathbf{x}^* = \mathbf{x}^{k+1}$. U suprotnom, vratiti se na *Korak 2*, pri čemu je $\mathbf{x}^k = \mathbf{x}^{k+1}$.

Ovaj metod ima veću brzinu konvergencije u odnosu na osnovni gradijentni metod, ali ima krupan nedostatak što se u njoj najviše vremena obično potroši na jednodimenzionalnu optimizaciju kojom se izračunava optimalna dužina koraka. Rukovodeći se ovom činjenicom, u ovoj knjizi je dat algoritam za efikasno rešavanje simboličke konstrukcije nove funkcije $F(\alpha)$. Ideja simboličke implementacije je da se konstruiše nova ciljna funkcija

$$F(\alpha) = Q\left(\mathbf{x}^{(k)} \pm \alpha \nabla Q(\mathbf{x}^{(k)})\right),$$

na koju se mogu primeniti ranije implementirani metodi jednodimenzionalne optimizacije, sa ciljem da se izračuna $\min\limits_{\alpha} Q\left(\mathbf{x}^{(k)} \pm \alpha \nabla Q(\mathbf{x}^{(k)})\right)$ u toku minimizacije, ili $\max\limits_{\alpha} Q\left(\mathbf{x}^{(k)} + \alpha \nabla Q(\mathbf{x}^{(k)})\right)$ u toku maksimizacije. Pri tome se uvek mora imati u vidu uslov $\alpha > 0$.

Pored ranijih prednosti **(1G)**, **(2G)** i **(3G)**, može se istaći i sledeća prednost:

(4G) Ako ciljna funkcija nije zadata potprogramom, u proceduralnim programskim jezicima nije jednostavno generisati novu funkciju $f(\alpha)$, koristeći (2.4.5) ili (2.4.6). Poznat je sledeći problem *optimizacije po pravcu*[1)] kao jednodimeznionalni optimizacioni problem u n-dimenzionalnom prostoru [15]:

Za datu tačku $\mathbf{x}^k$ i pravac $\mathbf{s}^k$ naći korak λ^* za koji je

$$Q(\mathbf{x}^k + \lambda^* \mathbf{s}^k) = \min_{\lambda} Q(\mathbf{x}^k + \lambda \mathbf{s}^k). \tag{2.4.4}$$

Niz jednodimenzionalnih optimizacionih potproblema oblika (2.4.4) generiše se u različitim metodima optimizacije za višeargumentne funkcije. Jedan od takvih metoda je Cauchyev metod.

Iz referenci [2], [13], [15], [60] može se zaključiti da problem *optimizacija po pravcu* nije jednostavno rešiv u proceduralnim programskim jezicima. U algoritmima koji su korišćeni u literaturi, funkcija $F(\alpha)$ se može definisati samo pod pretpostavkom da je ciljna funkcija definisana potprogramom. Univerzalni algoritam za automatsko generisanje funkcije $F(\alpha)$, koji koristi proizvoljnu ciljnu funkciju Q, nije jednostavan za implementaciju. U ovoj knjizi dajemo algoritme bazirane na *simboličkom* generisanju funkcije $F(\alpha)$ za proizvoljnu ciljnu funkciju Q. Funkcija Q ne mora da bude zadata potprogramom, već može da se nalazi u listi parametara procedure kojom se implementira neki od metoda optimizacije.

Neka je zadat analitički izraz ciljne funkcije Q u formalnom parametru $q\_$, neka je lista promenljivih zadata parametrom $prom\_$, i neka je tekuća iteracija $\mathbf{x}^{(k)}$ zadata vektorom x. Tada se nova funkcija

$$F(h) = Q\left(\mathbf{x}^{(k)} \pm h \nabla Q(\mathbf{x}^{(k)})\right)$$

može jednostavno formirati sledećim naredbama:

```
qexp=q;
Do[qexp=qexp/.prom[[i]]->N[x[[i]]]-h*N[norgrad[[i]]],{i,n}];
```

u slučaju minimuma, ili naredbama

```
qexp=q;
Do[qexp=qexp/.prom[[i]]->N[x[[i]]]+h*N[norgrad[[i]]],{i,n}];
```

u slučaju maksimuma. Unutrašnja forma funkcije $F(h)$ jeste funkcija čiji je analitički izraz zadat pomoću

$$Q\left(x_1^{(k)} \pm h\big(\nabla Q(\mathbf{x}^{(k)})\big)_1, \ldots, x_n^{(k)} \pm h\big(\nabla Q(\mathbf{x}^{(k)})\big)_n\right),$$

[1)] U anglosaksonskoj literaturi poznat kao *optimization along a line*

dok je lista parametara jednaka $\{h\}$.

Pre implementacije Cauchyevog metoda navodimo potprogramom *jed* proverava jednakost dve liste $l1$ i $l2$.

```
jed[l1_List,l2_List]:=
   Block[{n=Length[l1],i,uslov=True,uslov1},
    Do[uslov1=N[l1[[i]]]==N[l2[[i]]]; uslov=uslov && uslov1,
                  {i,n}];
    Return[uslov]  ]
```

U funkciji kojom se implementira Cauchyev metod, za kriterijum zaustavljanja se uzima jednakost dve uzastopne aproksimacije optimalne tačke.

```
Cauchy[q_,prom_List,x0_List,eps_]:=
   Module[{q1,qexp,i=1,tacka,n=Length[prom],
     grad,it=0,q0,norgrad,metod,qcrt,pom,h,s=eps+1,
     izb,qm,hk,pret,Lista={}},
   izb=Input["Unesi <1> za minimum, a <2> za maksimum"];
   Print["Izaberi metod jednodimenzionalne optimizacije."];
   Print["<1> skeniranje konstantnim korakom"];
   Print["<2> skeniranje promenljivim     korakom"];
   Print["<3> simplexI metod"];
   Print["<4>    simplexII metod"];
   Print["<5> zlatni presek"];
   Print["<6> metod dihotomije"];
   Print["<7> DSC metod"];
   Print["<8> Powelov metod"];
   Print["<9> DSC-Powelov metod"];
   metod=Input[];
   pom=x0; hk=10;
   While[it<50 && N[hk]>eps,
      q0=q; Do[q0=q0/.prom[[i]]->pom[[i]],{i,n}];
      Lista=Append[Lista,pom];    grad=nabl[q,prom,pom];
      s=norma[grad]; norgrad=grad/N[s];
      it+=1;
      (*  Formiranje simbolicke funkcije F(h) *)
      qexp=q;
      Which[izb==1,
        Do[qexp=qexp/.prom[[i]]->
            N[pom[[i]]]-h*N[norgrad[[i]]],{i,n}],
        izb==2,
        Do[qexp=qexp/.prom[[i]]->
```

```
          N[pom[[i]]]+h*N[norgrad[[i]]],{i,n}]
    ];
    (*Izbor metoda jednodimenzionalne optimizacije*)
   Which[metod==1,hk=skk[qexp,{h},0,1,0.01],
         metod==2,hk=spk[qexp,{h},0,1,0.5,eps/10],
         metod==3,hk=simplexI[qexp,{h},0,1,0.5,eps/10],
         metod==4,hk=simplexII[qexp,{h},0,0.1,eps/10],
         metod==5,hk=zlatni[qexp,{h},0,1,eps/10],
         metod==6,hk=dih[qexp,{h},0,1,eps/10],
         metod==7,hk=Dsk[qexp,{h},0.,0.1,eps/10],
         metod==8,hk=Powel[qexp,{h},0.,0.1,eps/10],
         metod==9,hk=dskpowel[qexp,{h},0.,0.1,eps/10]
     ];
     h1=h1[[1]];
     (*Jednodimenzionalni metodi vracaju listu*)
     pret=pom;
     If[izbor==1,
        Do[pom[[i]]=N[pom[[i]]-h1*norgrad[[i]]],{i,n}],
        Do[pom[[i]]=N[pom[[i]]+h1*norgrad[[i]]],{i,n}]
     ];
     qm=q; Do[qm=qm/.prom[[i]]->pom[[i]],i,n];
     Lista=Append[Lista,pom];
     If[jed[pom,pret],it=50];
    ];
    {pom,N[qm], Lista}
   ]
```

Test primeri:

```
In[1]:= Cauchy[x^2+y^2,{x,y},{1.0,-1.2},0.0001]
Unesi <1> za minimum, a <2> za maksimum 1
Izaberi metod jednodimenzionalne optimizacije.
<1> skeniranje konstantnim korakom
<2> skeniranje promenljivim korakom
<3> simplexI metod
<4> simplexII metod
<5> zlatni presek
<6> metod dihotomije
<7> DSC metod
<8> Powelov metod
<9> DSC-Powelov metod
? 2
```

qexp = (1. - 0.640184 h$1) + (-1.2 + 0.768221 h$1)

Rezultat jednodimenzionalne minimizacije $\min_h qexp(h)$ je

h=1.

Optimalna tačka je

pom= {0.359816, -0.431779}

U novim iteracijama dobijaju se redom sledeći rezultati:

qexp = (0.359816 - 0.640184 h$1)^2 + (-0.431779 + 0.768221 h$1)^2
h=0.56205

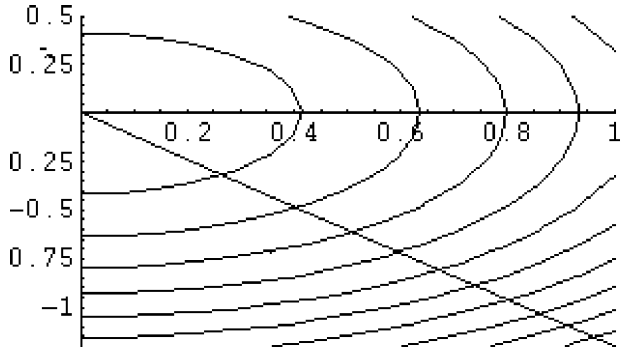
pom= {4.92869 10^-6 ,-5.91442 10^-6 }

qexp = (4.92869 10^-6 - 0.640184 h$1)^2 + (-5.91442 10^-6 + 0.768221 h$1)^2
h= 0.

pom= {4.44694 10^-8 ,-5.33596 10^-8 }

Out[1]={{4.44664 10−8 , -5.33596 10−8 }, 4.82451 10−15 ,
{{1., -1.2}, {0.359816, -0.431779}, {0.359816, -0.431779},
4.44664 10−8,-5.33596 10−8 , 4.44664 10−8, -5.33596 10−8 ,
4.44664 10−8 , -5.33596 10−8 }}}

Ovi rezultati su predstavljeni na slici 2.4.6.



Sl. 2.4.6

In[2]:=Cauchy[8*(x-1)^2+3*(y-2)^2+Cos[(x-1)*(y-2)],{x,y},{0,3},
{-10.,-10.},{10,10.},0.001]
Unesi <1> za minimum, a <2> za maksimum 1
Izaberi metod jednoimenzionalne optimizacije.
<1> skeniranje konstantnim korakom
<2> skeniranje promenljivim korakom
<3> simplexI metod

```
<4> simplexII metod
<5> zlatni presek
<6> metod dihotomije
<7> DSC metod
<8> Powelov metod
<9> DSC-Powelov metod
   ?2
Out[2]= {50, {1.00101, 2.}, 0.0015625, 1.00001} {{1.00101, 2.}, 1.00001}
In[3]:= f1[x_,y_]:=8*(x-1)^2+3*(y-2)^2+Cos[(x-1)*(y-2)]
In[4]:=Cauchy[f1,{x,y},{0,3},{-10.,-10.},{10,10.},0.001]
```

Koristeći u toku jednodimenzionalne optimizacije skeniranje sa promenljivim korakom, dobijaju se sledeće vrednosti za liste $\{it, N[pom], N[qm]\}$:

```
   {1, {-0.946338, 3.32204}, 34.7065}
   {2, {-1.92133, 3.54263}, 75.208}
   {3, {-2.89087, 3.78607}, 131.467}
   {4, {-3.88139, 3.92071}, 200.692}
   {5, {-4.87073, 4.06383}, 289.402}
   {6, {-5.85818, 4.21937}, 390.171}
   {7, {-6.85353, 4.31192}, 510.227}
   {8, {-7.84231, 4.45882}, 642.659}
   {9, {-8.83798, 4.54766}, 794.757}
   {10, {-9.83253, 4.64841}, 958.876}
   {15, {-10., 5.0574}, 995.442}
   {20, {-10., 5.57608}, 1006.3}
   {25, {-10., 6.14525}, 1019.5}
   {30, {-10., 6.7954}, 1036.2}
   {35, {-10., 7.6116}, 1062.92}
   {40, {-10., 8.58989}, 1097.31}
   {45, {-10., 9.74617}, 1147.08}
   {46, {-10., 10.}, 1161.}
Out[4]={{-10., 10.},1161., { {-0.946338, 3.32204},{-1.92133, 3.54263},{-2.89087, 3.78607},
   {-3.88139, 3.92071}, {-4.87073, 4.06383},{-5.85818, 4.21937}, {-6.85353, 4.31192},
   {-7.84231, 4.45882},{-8.83798, 4.54766},{-9.83253, 4.64841}, {-10., 5.0574},
   {-10., 5.57608},{-10., 6.14525},{-10., 6.7954}, {-10., 7.6116},
   {-10., 8.58989}, {-10., 9.74617},{-10., 10.}}}
```

Mogu se istaći sledeći opšti zaključci koji proizilaze posle poređenja rezultata dobijenih softverom koji je opisan u ovoj knjizi sa odgovarajućim rezultatima koji su dobijeni primenom softvera koji je napisan u proceduralnim programskim jezicima.

1. Numeričke mogućnosti jezika SCHEME ne zaostaju za numeričkim mogućnostima proceduralnih programskih jezika FORTRAN ili C. Numeričke mogućnosti programskog jezika MATHEMATICA su daleko veće. S druge strane, u funkcionalnim programskim jezicima ostaje bogatstvo simboličke obrade podataka, koje se itekako može iskoristiti u implementaciji metoda

optimizacije. To je osnovna ideja koja je korišćena pri izradi softvera za simboličku optimizaciju.

2. Implementacija gradijentnih metoda optimizacije u funkcionalnim programskim jezicima uzrokuje ubrzanje konvergencije implementiranih metoda. Poboljšanje konvergencije nastalo je kao posledica simboličke implementacije parcijalnih izvoda i simboličke implementacije problema *optimizacije po pravcu*.

Adekvatni programski jezici za implementaciju metoda optimizacije nisu proceduralni programski jezici, već programski jezici primenljivi istovremeno i u numeričkoj i u simboličkoj obradi podataka. Kao predstavnici takvih jezika izabrani su MATHEMATICA i SCHEME. Naravno, i drugi funkcionalni programski jezici, a verovatno i odgovarajući programski jezici koji će u budućnosti biti razvijeni, mogu da se koriste saglasno principima koji su opisani u ovoj knjizi.

2.4.5. METOD RELAKSACIJE

Ovaj metod je analogon Gauus-Seidelovom metodu, s tom razlikom što se niz upravljačkih parametara po kojima se vrši optimizacija određuje na osnovu optimalno određene koordinate gradijenta.

Za lokalizaciju optimuma, počev od zadate početne tačke, koristi se upravljački parametar x_p za koji ciljna funkcija ima najveću izmenu, a koji je definisan izrazom

$$\frac{\partial Q}{\partial x_p} = \max_i \left| \frac{\partial Q}{\partial x_i} \right|.$$

Kriterijum za prekid pretraživanja je

$$\sqrt{\sum_{i=1}^{n} \left(\frac{\partial Q(\mathbf{x}^{(0)})}{\partial x_i} \right)^2} \leq \varepsilon.$$

Programska implementacija ima oblik:

```
Relaxm[q_,prom_List,x_List,eps_]:=
Block[{Lis={},metod,rad=True,p,pr=prom,
            x0=x1=x,qm=q0=q,del=1,n=Length[prom],naj,gde},
            Lis=Append[Lis,x0];
     Print["Izaberi metod jednoimenzionalne optimizacije."];
     Print["<1> skeniranje konstantnim korakom"];
     Print["<2> skeniranje promenljivim korakom"];
```

```
    Print["<3> simplexI metod"];
    Print["<4> simplexII metod"];
    Print["<5> zlatni presek"];
    Print["<6> metod dihotomije"];
    Print["<7> DSC metod"];
    Print["<8> Powelov metod"];
    Print["<9> DSC-Powelov metod"];
    metod=Input[];
    dqdxp=nabl[q0,pr,x0];
    While[rad && del>=eps,
        naj=Max[dqdxp];
        gde=Position[dqdxp,naj][[1]][[1]];
        qm=q;
        Do[qm=qm/.prom[[i]]->x0[[i]],{i,gde-1}];
        qm=qm/.prom[[gde]]->p;
        Do[qm=qm/.prom[[i]]->x0[[i]],{i,gde+1,n}];
        (*jednodimenzionalna optimizacija po p*)
        Which[metod==1,p0=skk[qm,{p},0,1,0.01],
        metod==2,p0=spk[qm,{p},0,1,0.5,eps/10],
        metod==3,p0=simplexI[qm,{p},0,1,0.5,eps/10],
        metod==4,p0=simplexII[qm,{p},-1,0.1,eps/10],
        metod==5,p0=zlatni[qm,{p},-1,1,eps/10],
        metod==6,p0=dih[qm,{p},-1,1,eps/10],
        metod==7,p0=Dsk[qm,{p},-1,0.1,eps/10],
        metod==8,p0=Powel[qm,{p},-1,0.1,eps/10],
        metod==9,p0=dskpowel[qm,{p},-1,0.1,eps/10]
        ];
        x0[[gde]]=p0[[1]];
        If[jed[x0,x1],rad=False,
           x1=x0;
           Lis=Append[Lis,x0];
           dqdxp=nabl[q0,pr,x0];
           del=Sqrt[Sum[dqdxp[[i]]^2,{i,n}]]
        ]
    ];
    Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
    {x0,q0,Lis}
]
```

2.5. Gradijentni metodi drugog reda

Osim problema **(1U)**–**(3U)**, **(1M)**–**(3M)**, **(1G)**–**(4G)**, koji su teško rešivi bez simboličke manipulacije podacima, osnovni problem vezan za gradijentne metode drugog reda su sledeći:

(1N) Obrazovati Hesseovu matricu parcijalnih izvoda drugog reda ciljne funkcije $Q(\mathbf{x})$ u tački $\mathbf{x}^{(k)}$:

$$\nabla^2 Q(x^{(k)}) = \mathbb{H}(\mathbf{x}^{(k)}) = \begin{bmatrix} \dfrac{\partial^2 Q^{(k)}}{\partial {x_1}^2} & \cdots & \dfrac{\partial^2 Q^{(k)}}{\partial x_1 \partial x_n} \\ \vdots & & \\ \dfrac{\partial^2 Q^{(k)}}{\partial x_n \partial x_1} & & \dfrac{\partial^2 Q^{(k)}}{\partial {x_n}^2} \end{bmatrix}.$$

Parcijalni izvodi drugog reda u paketu MATHEMATICA mogu se simbolički formirati pomoću operatora diferenciranja D. Neka je unutrašnja forma ciljne funkcije Q zadata analitičkim izrazom $q\_$ i listom argumenata $prom\_ = \{x_1, \ldots, x_n\}$. Tada se parcijalni izvod drugog reda $\dfrac{\partial^2 Q}{\partial x_i \partial x_j}$ može generisati *simbolički* izrazom:

```
D[q,prom[[i]],prom[[j]]]
```

Takođe, pomoću ugrađene funkcije *Append* nije problem da se ovako dobijeni simbolički parcijalni izvodi drugog reda ugrade u matricu, koja će predstavljati Hesseovu matricu u simboličkom obliku:

```
hes[q_,prom_List,x0_List]:=
  Block[{n,i,j,hesa={},dqdx},
     n=Length[prom];
     Do[dqdx={};
        Do[dqdx=Append[dqdx,D[q,prom[[i]],prom[[j]]]],{j,n}];
        hesa=Append[hesa,dqdx],
        {i,n}
     ];
     Do[hesa=hesa/.prom[[i]]->x0[[i]],{i,n}];
     hesa
  ]
```

2.5.1. NEWTONOV METOD

Da bi se povećala efektivnost gradijentnog pretraživanja koristi se kvadratna aproksimacija ciljne funkcije $Q(\mathbf{x})$ u oblasti zadate tačke $\mathbf{x}^{(k)}$, tako

da u iterativnoj proceduri učestvuju i drugi izvodi. Kvadratna aproksimacija $\tilde{Q}(\mathbf{x})$ funkcije $Q(\mathbf{x})$, u okolini tačke $\mathbf{x}^{(k)}$, dobija se kada se iz Taylorovog razvoja funkcije $Q(\mathbf{x})$ u tački $\mathbf{x}^{(k)}$ odbace članovi trećeg i višeg reda:

$$\tilde{Q}(\mathbf{x}) = Q(\mathbf{x}^{(k)}) + \nabla Q(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} + \frac{1}{2}(\Delta\mathbf{x}^{(k)})^T\nabla^2 Q(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)}.$$

Na osnovu ove kvadratne aproksimacije funkcije $Q(\mathbf{x})$ dobijamo

$$\nabla\tilde{Q}(\mathbf{x})_{\mathbf{x}=\mathbf{x}^{(k)}} = \nabla Q(\mathbf{x}^{(k)}) + \nabla^2 Q(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)},$$

odakle je

$$\Delta\mathbf{x}^{(k)} = -\left[\nabla^2 Q(\mathbf{x}^{(k)})\right]^{-1}\nabla Q(\mathbf{x}^{(k)}). \tag{2.5.1}$$

Newtonov optimizacioni metod koristi iterativni korak koji je zasnovan na formuli (2.5.1), tako da se Newtonova iterativna procedura definiše na sledeći način:

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} - \left[\nabla^2 Q(\mathbf{x}^{(k)})\right]^{-1}\nabla Q(\mathbf{x}^{(k)}) \\ &= \mathbf{x}^{(k)} - H^{-1}(\mathbf{x}^{(k)})\nabla Q(\mathbf{x}^{(k)}),\end{aligned} \tag{2.5.2}$$

gde je, zbog jednostavnijeg označavanja, uvedena oznaka

$$\nabla^2 Q(\mathbf{x}^{(k)}) = H(\mathbf{x}^{(k)}).$$

Konvergencija Newtonovog metoda je obezbeđena ako je ciljna funkcija $Q(\mathbf{x})$ dvaput diferencijabilna i inverzija Hesseove matrice pozitivno definitna, tj. $H^{-1}(\mathbf{x}^{(k)}) > 0$.

Iz (2.5.2) proizilazi da su pravac napredovanja prema optimamalnoj vrednosti, kao i veličina koraka, u potpunosti definisani pozitivno definitnom Hesseovom matricom $H(\mathbf{x}^{(k)})$. Upoređivanjem Newtonovog optimizacionog metoda sa gradijentnim metodima prvog reda, lako je zaključiti da je kod Newtonovog metoda korak pomeranja u pravcu gradijentnog vektora jednak $H^{-1}(\mathbf{x}^{(k)})$.

Newtonov optimizacioni metod se odlikuje kvadratnom konvergencijom u blizini optimalne vrednosti. Za kvadratnu funkciju $Q(\mathbf{x})$ dovoljan je jedan korak za nalaženje optimuma.

U toku implementacije Newtonovog metoda proizilazi još jedan novi problem, koji je pogodan za simboličku implementaciju, kao i za veliki broj ugrađenih funkcija iz linearne algebre u programskom paketu MATHEMATICA:

(2N) U svakoj iteraciji je neophodno da se izvrši inverzija Hesseove matrice tipa $n \times n$. Osim toga, potrebni su analitički izrazi za parcijalne izvode drugog reda funkcije $Q(\mathbf{x})$, što za neke složenije ciljne funkcije može da bude veliki problem.

U implementaciji Newtonovog metoda imamo:

Ulazne veličine:

$q\_$, *prom_*: ciljna funkcija i lista njenih parametara;

$x01\_$: izabrana početna tačka;

*eps_*: zadata tačnost lokalizacije optimuma.

Lokalne promenljive:

izb: parametar koji određuje lokalizaciju minimuma ili maksimuma;

nabla, *hesa*: gradijent i Hesseova matrica ciljne funkcije.

Algoritam Newtonovog metoda može se iskazati sledećim koracima:

Korak 1. Izračunati $q0 = Q(x0),\ \nabla Q(x0),\ S = \|\nabla Q(x0)\|$.

Korak 2. `While` ciklus, koji se prekida kada je ispunjen uslov

$$S = \|\nabla Q(x0)\| < eps.$$

Unutar ciklusa izvršiti sledeće korake:

Korak 2.1. Formirati Hesseovu matricu $\nabla^2 Q(\mathbf{x}^{(k)}) = H(\mathbf{x}^{(k)})$.

Korak 2.2. Izračunati novu tačku optimuma $x0$:

$$x0 = x0 \pm \left[\nabla^2 Q(\mathbf{x}^{(k)})\right]^{-1} \nabla Q(\mathbf{x}^{(k)}).$$

Korak 2.3. Izračunati $q0 = Q(x0),\ \nabla Q(x0),\ S = \|\nabla Q(x0)\|$.

Korak 3. Izlazne veličine su $x0$ i $q0$.

Zbog očiglednih prednosti koje omogućuje MATHEMATICA u inverziji Hesseove matrice, navedena je samo implementacija u tom jeziku. Detalji vezani za implementaciju u LISPu se mogu naći u [51].

```
newton[q_,prom_List,x01_List,eps_]:=
   Block[{i,n,q0,nabla,hesa,s,x0=x01,xm,qm,it=0,
```

```
      izb,Lista={}},
         izb=Input["Zelite li minimum(1) ili maksimum(2)?  "];
         n=Length[x0];
         q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}]; q0=N[q0];
         xm=x0; qm=q0;  Lista=Append[Lista,xm];
         nabla=nabl[q,prom,x0];
         s=N[norma[nabla]];
         Print[s,xm,qm];
         While[N[s]>=eps && it<=100,
          hesa=hes[q,prom,x0];
          If[Det[hesa]==0,
            Print["Hesseova matrica je singularna "];
            Return[],
            If[izb==2,   x0=x0+
            (Transpose[Inverse[hesa].Transpose[{nabla}]][[1]]),
                        x0=x0-
            (Transpose[Inverse[hesa].Transpose[{nabla}]][[1]]);
            ];
              q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
              If[(izb==2 && q0>qm)||(izb==1 && q0<qm),
                        qm=q0; xm=x0; Lista=Append[Lista,xm];
              ];
               it+=1;
               nabla=nabl[q,prom,x0];
               s=N[norma[nabla]];
          ]
         ];
         {xm,qm, Lista}
      ]
```

Numerički eksperimenti.

U slučaju kvadratne funkcije cilja, Newtonov metod dostiže optimalnu vrednost već u prvoj iteraciji.

In[1]:= newton[x^2/y^2-1,{x,y},{10,21},0.0001] (*minimum*)

Out[1]= {{10, 21}, -0.773243, {{10, 21}}}

Redom su dobijene sledeće vrednosti za normu gradijenta:
$0.0502309, 0.0502309, 0.0251154, 0.0125577, 0.00627886,$
$0.00313943, 0.00156971, 0.000784857, 0.000392429, 0.000196214.$

In[2]:= newton[x^2+y^2-1,{x,y},{10.2,21.3},0.0001] (* minimum *)

Out[2]= {{0., 0.}, -1., { {10.2, 21.3},{0., 0.}}}

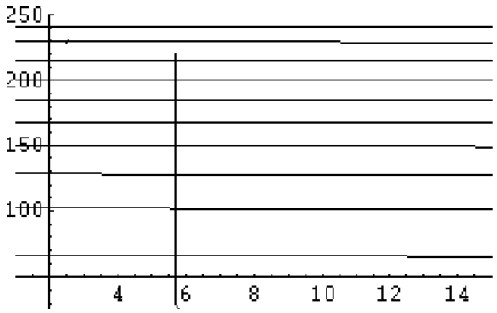
Za nekvadratne funkcije cilja potreban je veći broj iteracija.

```
In[3]:= newton[x^2/y^2-1+3*Sin[x-1],{x,y},{10.2,21.3},0.0001]    (*minimum*)
Out[3]={{5.71234,220.524}, -3.99933, {{10.2,21.3},{5.8583, 22.3557},
{5.70864, 29.4268}, {5.71092, 39.2436}, -3.97882},{5.71157, 52.3278},{5.71193, 69.7725},
{5.71213, 93.0317}, {5.71224, 124.044},{5.71231, 165.392},{5.71234, 220.524}}}
```

Dobijene su sledeće vrednosti za normu gradijenta:

$2.87965, 2.87965, 0.459669, 0.0032172, 0.00321018, 0.00175982, 0.000975228,$
$0.000543868, 0.00030443, 0.000170766.$

Rezultati dobijeni u Out[3] prikazani su na slici 2.5.1.



Sl. 2.5.1

2.5.2. MODIFIKACIJA NEWTONOVOG METODA

Praksa pokazuje da Newtonov metod nije naročito prikladan za nekvadratne funkcije. Newtonov metod je brz (u slučaju konvergencije), ali "hirovit". Ako je početna tačka $\mathbf{x}^{(0)}$ dalje od ekstremuma $\mathbf{x}^*$, korak definisan u (2.5.1) može se pokazati prevelikim i tada metod nema konvergenciju prema rešenju. Osim toga, veliki je problem inverzija Hesseove matrice. Najzad, Hesseova matrica može da bude singularna ili blizu singularne. Iz tih razloga se koristi *modifikovani Newtonovov metod* u kome se uvodi parametar koraka, koji se u k-toj iteraciji označava sa α_k:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \pm \alpha_k \left[\nabla^2 Q(\mathbf{x}^{(k)})\right]^{-1} \nabla Q(\mathbf{x}^{(k)}). \tag{2.5.3}$$

Izbor parametra α_k bi trebalo da garantuje napredovanje prema optimumalnoj vrednosti.

Jedan od načina da se poveća pouzdanost Newtonovog metoda je da se uzme optimalna dužina parametra koraka, koja se dobija kao rešenje sledećeg jednodimenzionalnog optimizacionog problema:

$$G(\alpha_k) = \min_h G(h) = \min_h Q\left(\mathbf{x}^{(k)} \pm h\left[\nabla^2 Q(\mathbf{x}^{(k)})\right]^{-1} \nabla Q(\mathbf{x}^{(k)})\right). \tag{2.5.4}$$

Sada se može istaći još jedna prednost koja se dobija primenom simboličkog diferenciranja:

(3N) Znatno je pogodniji problem za funkcionalne programske jezike da se formira nova funkcija $G(h)$, definisana sa (2.5.4). Funkcija $G(h)$ se formira kao kod Cauchyevog metoda. Navedene su dve funkcije za implementaciju ove modifikacije Newtonovog metoda u jeziku MATHEMATICA:

```
newtonm[q_,prom_List,x01_List,eps_]:=
   Block[{i,n,q0,nabla,hesa,s,v,x0=x01,it=0,h,hk,
             qexp,metod,izb, Lista={} },
      izb=Input["Zelite li minimum(1) ili maksimum(1)?"];
      Print["Izaberi jednoimenzionalnu optimizaciju."];
      Print["<1> skeniranje konstantnim korakom"];
      Print["<2> skeniranje promenljivim korakom"];
      Print["<3> simplexI metod"];
      Print["<4> simplexII metod"];
      Print["<5> zlatni presek"];
      Print["<6> metod dihotomije"];
      Print["<7> DSC metod"];
      Print["<8> Powelov metod"];
      Print["<9> DSC-Powelov metod"];
      metod=Input[];
      n=Length[x0];
      q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
      Lista=Append[Lista,x0];
      nabla=nabl[q,prom,x0];
      s=norma[nabla];
      While[s>=eps && it<=100,
        hesa=hes[q,prom,x0];
        If[Det[hesa]==0,
           Print["Hesseova matrica je singularna "];
           Return[{}],
           v=nmnomatvec[Inverse[hesa], nabla];
```

```
            qexp=q;
            If[izb==1,
                Do[qexp=qexp/.prom[[i]]->
                    N[x0[[i]]]-h*N[v[[i]]],{i,n}],
                Do[qexp=qexp/.prom[[i]]->
                    N[x0[[i]]]+h*N[v[[i]]],{i,n}]
            ];
            Print["qexp = ",qexp];
            Which[metod==1,hk=skk[qexp,{h},0,1,0.01],
              metod==2,hk=spk[qexp,{h},0,1,0.5,eps/10],
              metod==3,hk=simplexI[qexp,{h},0,1,0.5,eps/10],
              metod==4,hk=simplexII[qexp,{h},0,0.1,eps/10],
              metod==5,hk=zlatni[qexp,{h},0,1,eps/10],
              metod==6,hk=dih[qexp,{h},0,1,eps/10],
              metod==7,hk=Dsk[qexp,{h},0.,0.1,eps/10],
              metod==8,hk=Powel[qexp,{h},0.,0.1,eps/10],
              metod==9,hk=dskpowel[qexp,{h},0.,0.1,eps/10]
            ];
            hk=hk[[1]];
            If[izb==1,
              x0=x0-hk*v, x0=x0+hk*v
            ];
            q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
            Lista=Append[Lista,x0];
            it+=1;
            Print[{x0,q0}];
            nabla=nabl[q,prom,x0];
            s=norma[nabla];
          ]
        ];
        {x0,q0, Lista}
    ]
```

Numerički eksperimenti.

Testira se problem minimizacije u sledećem izrazu:

In[1]:= newtonm[x^2+y^2,{x,y},{-2,3},0.0001]

Neka je za rešavanje problema jednodimenzionalne optimizacije $\alpha_k = \min_h G(h)$ izabran metod skeniranja sa promenljivim korakom. Optimalna vrednost se dostiže u prvoj iteraciji. Simbolička funkcija $G(h)$ data je izrazom

qexp = (3. - 3. h)^2 + (-2. + 2. h)^2

Rezultat je
Out[1]= {{0., 0.}, 0.{{0., 0.}}}

Implementacija modifikovanog Newtonovog metoda u jeziku LISP opisana je u [51].

2.6. Metodi promenljive metrike

Metodi promenljive metrike objedinjuju pozitivna svojstva gradijentnih metoda prvog reda i Newtonovog optimizacionog metoda, tj. ovi metodi se odlikuju brzim početnim napredovanjem prema optimalnoj tački i kvadratnom konvergencijom u blizini optimalne tačke. Osnovna ideja je da pretraživanje počne gradijentnim metodom prvog reda, što daje velike početne skokove. Preciznije, početne iteracije su oblika:

$$\mathbf{x}^{k+1} = \mathbf{x}^k \pm \alpha_k \nabla Q(\mathbf{x}^k).$$

U blizini ekstremuma, potrebno je da iteracije imaju sledeći oblik:

$$\mathbf{x}^{k+1} = \mathbf{x}^k \pm \alpha_k \left[\nabla^2 Q(\mathbf{x}^k)\right]^{-1} \nabla Q(\mathbf{x}^k),$$

što garantuje završavanje sa kvadratnom konvergencijom. Međutim, umesto inverzije Hesseove matrice, obično se koristi neka njena aproksimacija. Modifikaciju matrice $\left[\nabla^2 Q(\mathbf{x}^k)\right]^{-1}$ označavaćemo sa H_k. Takve modifikacije Newtonovog metoda se mogu opisati pomoću

$$\mathbf{x}^{k+1} = \mathbf{x}^k \pm \alpha_k H_k \nabla Q(\mathbf{x}^k).$$

U početnim iteracijama se postavlja $H_0 = \mathbf{I}$, što implicira

$$\mathbf{x}^1 = \mathbf{x}^0 \pm \alpha_0 \nabla Q(\mathbf{x}^0).$$

Pri tome se može uzeti da je korak α_0 određen iz jednodimenzionalne optimizacije

$$F(\alpha_0) = \min_{\alpha} Q\left(\mathbf{x}^{(0)} - \alpha \nabla Q(\mathbf{x}^{(0)})\right) = \min_{\alpha} F(\alpha), \qquad \alpha > 0,$$

u slučaju minimuma, odnosno kao rešenje jednodimenzionalnog optimizacionog problema

$$F(\alpha_0) = \max_{\alpha} Q\left(\mathbf{x}^{(0)} + \alpha \nabla Q(\mathbf{x}^{(k)})\right) = \max_{\alpha} F(\alpha), \qquad \alpha > 0,$$

u slučaju maksimuma. Ovim se obezbeđuje početno napredovanje pomoću Cauchyevog metoda. Takođe, može se uzeti fiksirana vrednost parametra α. Međutim, da bi se ostvarila kvadratna konvergencija, potrebno je da H_k u svakoj iteraciji bude približno jednaka sa inverzijom Hesseove matrice, tj. $H_k \approx \left[\nabla^2 Q(\mathbf{x}^k)\right]^{-1}$.

Svaka modifikacija Newtonovog metoda sa konkretnim "pravilom upotpunjavanja" matrice H_k prema gornjem pravilu naziva se *metod promenljive metrike*. Do sada je definisan veći broj metoda promenljive metrike.

2.6.1. METOD MARKUARDA

Ovaj metod predstavlja kombinaciju osnovnog gradijentnog metoda prvog reda i Newtonovog metoda. Prema minimumu se napreduje saglasno formuli

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \pm \left[H(\mathbf{x}^{(k)}) + \lambda^{(k)}\mathbf{I}\right]^{-1} \nabla Q(\mathbf{x}^{(k)}), \tag{2.6.1}$$

Parametar $\lambda^{(k)}$ određuje kretanje prema ekstremumu. Za veliko $\lambda^{(k)}$ ($\lambda^{(k)} > 10^3$) u izrazu (2.6.1) dominira dijagonalna matrica $\lambda^{(k)}\mathbf{I}$, te napredovanje odgovara gradijentnim metodima prvog reda. Za $\lambda^{(k)} = 0$ napredovanje je kao u Newtonovom metodu. U početku se uzima parametar $\lambda^{(0)}$ sa velikom vrednošću, na primer $\lambda^{(0)} = 10^4$ i tada proizvod $\lambda^{(0)}\mathbf{I}$ "potiskuje" uticaj Hesseove matrice $H(\mathbf{x}^{(k)})$, te se proces pretraživanja odvija prema pravcu gradijenta $\nabla Q(\mathbf{x}^{(0)})$. Kada se u k-toj iteraciji poboljša vrednost ciljne funkcije, parametar $\lambda^{(k)}$ se smanjuje. Tada se uzima novi korak $\lambda^{(k+1)} < \lambda^{(k)}$ pa se sa tako određenim korakom načini nova iteracija. U protivnom, uzima se $\lambda^{(k+1)} = \beta\lambda^{(k)}$, pri čemu je $\beta > 1$, a zatim se ponavlja prethodni korak. Generalno posmatrano, svaka uspešna iteracija smanjuje korak optimizacije, a neuspešna povećava korak. Na taj način, što je aproksimacija bliža optimalnoj tački, formula (2.6.1) je sve približnija Newtonovim iteracijama. S druge strane, što je aproksimacija dalja optimalnoj tački, formula (2.6.1) je sve približnija gradijentnim metodima prvog reda. Time se konvergencija ovog metoda menja od linearne do kvadratne.

Metod Markuarda opisan je na sledeći način:

Korak 1. Izračunati

$$q0 = Q(x0) = Q(x^{(0)}),\ \nabla Q(x0) = \nabla Q(\mathbf{x}^{(0)}),\ S = \|\nabla Q(x0)\|.$$

Korak 2. Formirati `While` ciklus, koji se prekida kada je ispunjen uslov

$$S = \|\nabla Q(x0)\| < eps$$

ili kada se prekorači maksimalan broj iteracija.

Unutar ciklusa izvršiti sledeće korake:

Korak 2.1. Ako nova aproksimacija $x1$ nije "bolja" od $x0$ formirati gradijent $\nabla Q(x0) = \nabla Q(\mathbf{x}^{(0)})$, izračunati $S = \|\nabla Q(x0)\|$ i formirati Hesseovu matricu $\nabla^2 Q(\mathbf{x}^{(0)}) = H(x0)$.

Korak 2.2. Izračunati novu tačku $x1$ prema (2.6.1) ili (2.6.2):

$$x1 = x0 \pm [H(x0) + \lambda \mathbf{I}]^{-1} \nabla Q(x0).$$

Korak 2.3. Izračunati $q1 = Q(x1)$.

Korak 2.4. Ako je nova aproksimacija $x1$ "bolja" od $x0$, postaviti $\lambda = \lambda/2$, $q0 = q1$, $x0 = x1$; inače, postaviti $\lambda := 2\lambda$.

Korak 3. Izlazne veličine su $x0$ i $q0$.

U implementaciji metoda predviđena je mogućnost da se algoritam prekine na osnovu maksimalnog broja iteracija $n_k (= 100)$.

```
mark[q_,prom_List,x01_List,eps_]:=
   Block[{x0=x01,lambda=10000,it=0,q0,q1,i,n,s,sk=eps+1,
                    nabla,hesa,izb,e, ind=True,Lista={}},
     izb=Input["Maximum(2) ili minimum(1)?"];
     n=Length[prom]; e=IdentityMatrix[n];
     q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
     Lista=Append[Lista,x0];
     While[sk>=eps && it<=100,
       If[ind,
         nabla=nabl[q,prom,x0]; hesa=hes[q,prom,x0];
         sk=norma[nabla]//N;
       ];
       s=Inverse[hesa-lambda e].Transpose[{grad}];
       s=Transpose[s][[1]];
       If[izb==2, x1=N[x0+s],  x1=N[x0-s] ];
       q1=q; Do[q1=q1/.prom[[i]]->x1[[i]],{i,n}];
       If[(izb==2&&N[q1]>N[q0])||(izb==1&&N[q1]<N[q0]),
         q0=q1; x0=x1; lambda /=2;
         Lista=Append[Lista,x0]; ind=True,
         lambda *=2; ind=False
     ] ];
     {x0,q0,Lista} ]
```

Numerički eksperimenti. U ovom primeru je ilustrovana "hirovitost" Newtonovog metoda, kao i "dostižnost" osnovnog gradijentnog pretraživanja. Osnovni gradijentni metod konvergira sporo, ali dostižno prema optimalnoj vrednosti. Prikazane međuvrednosti sadrže redom normu gradijenta i listu koja sadrži tačku optimuma i optimalnu vrednst.

```
In[1]:= OsnGrad[x^2+y^2+Cos[x*y],{x,y},{1.0,1.0},{0.1,0.2},{-10,-10},{10,10},0.001]
Zelite li maximum(2) ili minimum(1)? 1
1.63841 {{0.884147,0.768294},2.15001}
1.61724 {{0.755585, 0.572067},1.8062}
1.51714 {{0.628432, 0.406545},1.52775}
1.32668 {{0.51302, 0.27569}, 1.32921}
1.09728 {{0.414302, 0.179877},1.20123}
0.879062 {{0.332781, 0.114096}, 1.12304}
0.69548 {{0.266658, 0.0709839},1.07597}
0.54931 {{0.213461, 0.0435998},1.04742}
0.434945 {{0.170809, 0.0265572},1.02987}
0.345485 {{0.136659, 0.0160893},1.01893}
0.275136 {{0.109331, 0.00971366},1.01205}
0.219503 {{0.0874658, 0.00585142},1.00768}
0.175317 {{0.069973, 0.0035198},1.00491}
0.140121 {{0.0559785, 0.00211533},1.00314}
0.112036 {{0.0447828, 0.00127052},1.00201}
0.0896015 {{0.0358262, 0.000762823},1.00128}
0.0716687 {{0.028661, 0.00045789},1.00082}
0.0573293 {{0.0229288, 0.000274809},1.00053}
0.0458609 {{0.018343, 0.000164914},1.00034}
0.0366876 {{0.0146744, 0.0000989597},1.00022}
0.0293495 {{0.0117395, 0.0000593801},1.00014}
0.0234794 {{0.00939164, 0.0000356297},1.00009}
0.0187834 {{0.00751331, 0.0000213784},1.00006}
0.0150267 {{0.00601065, 0.0000128273},1.00004}
0.0120213 {{0.00480852, 7.69648 10^-6},1.00002}
0.0096170 {{0.00384681, 4.61792 10^-6},1.00001}
0.0076936 {{0.00307745, 2.7707710^-6},1.00001}
0.0061549 {{0.00246196, 1.6624710^-6},1.00001}
0.0049239 {{0.00196957,9.97481 10^-7},1.}
0.0039391 {{0.00157565,5.9849 10^-7 },1.}
0.0031513 {{0.00126052,3.59094 10^-7},1.}
0.0025210 {{0.00100842,2.15457 10^-7},1.}
0.0020168 {{0.000806735,1.29274 10^-7},1.}
0.0016134 {{0.000645388,7.75644 10^-8},1.}
0.0012907 {{0.000516311,4.65386 10^-8},1.}
0.0010326 {{0.000413048,2.79232 10^-8},1.}
```

Za iste početne vrednosti Newtonov metod divergira.

```
In[2]:= newton[x^2+y^2+Cos[x*y],{x,y},{1.0,1.0},0.001]
Zelite li minimum(1) ili maksimum(2)? 1
1.63841 {{1., 1.},2.5403}
```

1.63841 {{1., 1.}, 2.5403}
...
21.4296{{1., 1.}, 2.5403}
48.1237 {{1., 1.}, 2.5403}
Out[2]= {{1., 1.}, 2.5403}

Međutim, ako se pođe od početne tačke koja je bliža optimalnoj, Newtonov metod konvergira vrlo brzo:

In[3]:= newton[x^2+y^2+Cos[x*y],{x,y},{0.5,0.5},0.001]
Zelite li minimum(1) ili maksimum(2)? 1
1.23927 {{0.5, 0.5}, 1.46891}
1.23927 {{-0.191011, -0.191011}, 1.0723}
0.530406 {{0.007367, 0.007367}, 1.00011}
0.0208375 {{-3.99921 10^{-7},-3.99921 10^{-7}},1.}
Out[3]= {{-3.99921 10^{-7},-3.99921 10^{-7}}, 1.,
{{0.5,0.5},{-0.191011,-0.191011},{0.007367,0.007367},{-3.9992 10^{-7},-3.9992 10^{-7}}}}

Markuardov metod se ponaša i po Newtonovom i po gradijentnom metodu. Za polaznu tačku $\{1.0, 1.0\}$ divergira:

In[4]:= mark[x^2+y^2+Cos[x*y],{x,y},{1.0,1.0},0.001]
Zelite li maximum(2) ili minimum(1)? 1
1.63841 {{1.01159,1.01159}, 2.56718}
1.63841 {{1.02321, 1.02321},2.59412}
...
1.63841 {{59.7137,59.7137}, 7130.44}
1.63841 {{-38.8171, -38.8171},3013.9}
...
1.63841 {{-13.8673,-13.8673}, 383.82}
Out[4]= {{-13.8673, -13.8673}, 383.82, {{1.01159, 1.01159},{1.02321, 1.02321},..., {{59.7137, 59.7137},{{-38.8171, -38.8171},..., {-13.8673, -13.8673}}}

Kada konvergira, čini to sporije od Newtonovog metoda:

In[5]:= mark[x^2+y^2+Cos[x*y],{x,y},{0.5,0.5},0.001]
Zelite li maximum(2) ili minimum(1)? 1
1.23927 {{0.508876,0.508876}, 1.48457}
1.23927 {{0.517982, 0.517982},1.50083}
1.23927{{0.536925, 0.536925}, 1.53531}
1.23927 {{0.578019, 0.578019}, 1.61292}
1.23927 {{0.675898, 0.675898}, 1.81113}
1.23927 {{0.971925, 0.971925}, 2.47531}
1.23927 {{3.47696, 3.47696}, 25.0668}
1.23927 {{-1.29979, -1.29979}, 3.26052}
1.23927 {{-0.498613, -0.498613}, 1.46648}
1.23927 {{-2.2984, -2.2984}, 11.1051}
1.23678 {{0.493634, 0.493634}, 1.45781}
1.23678 {{2.27569, 2.27569}, 10.8072}
1.22778 {{-0.47622, -0.47622}, 1.42796}
1.22778 {{-2.19697, -2.19697}, 9.76738}

```
1.19552 {{0.420648, 0.420648}, 1.33828}
1.19552 {{1.95225, 1.95225}, 6.83854}
1.08506 {{-0.286505, -0.286505}, 1.1608}
1.08506 {{-1.39148, -1.39148}, 3.51509}
0.777138 {{0.116438, 0.116438}, 1.02702}
0.777138 {{0.681125, 0.681125}, 1.82216}
0.327103 {{-0.0310068, -0.0310068}, 1.00192}
0.327103 {{-0.227341, -0.227341}, 1.10203}
0.0876583 {{0.00757654, 0.00757654}, 1.00011}
0.0876583 {{0.0585557, 0.0585557}, 1.00685}
0.0214291 {{-0.0018397, -0.0018397}, 1.00001}
0.0214291 {{-0.0142744, -0.0142744}, 1.00041}
0.00520346 {{0.000446541, 0.000446541}, 1.}
0.00520346 {{0.00346556, 0.00346556}, 1.00002}
0.00126301 {{-0.000108384, -0.000108384}, 1.}
0.00126301 {{-0.000841169, -0.000841169}, 1.}
0.000306556 {{0.0000263068, 0.0000263068}, 1.}
  Out[5]= {{0.0000263068, 0.0000263068}, 1.,
  {{0.508876, 0.508876},{0.517982, 0.517982},{0.536925, 0.536925},
  {0.578019, 0.578019},{0.675898,0.675898},{0.971925, 0.971925},
  {3.47696, 3.47696},{-1.29979,-1.29979},{-0.498613, -0.498613},
  {-2.2984, -2.2984}, {0.493634,0.493634},{2.27569, 2.27569},
  {-0.47622, -0.47622}, {-2.19697,-2.19697},{0.420648, 0.420648},
  {1.95225, 1.95225}, {-0.286505,-0.286505},{-1.39148, -1.39148},
  {0.116438, 0.116438},{0.681125,0.681125},{-0.0310068, -0.0310068},
  {-0.227341, -0.227341},{0.00757654, 0.00757654},{0.0585557, 0.0585557},
  {-0.0018397, -0.0018397}, {-0.0142744,-0.0142744},
  {0.000446541, 0.000446541},{0.00346556, 0.00346556},
  {-0.00010838,-0.00010838},{-0.0008411,-0.0008411},{0.0000263,0.0000263}}}
```

Osnovni nedostatak Markuardovog metoda je neophodnost da se raspolaže Hessovom matricom $H(\mathbf{x})$ za funkciju $Q(\mathbf{x})$ kao i invertovanje te matrice u svakoj iteraciji, naročito za veliki broj n upravljačkih parametara. U mnogim slučajevima Hesseova matrica je nepoznata ili je njeno izračunavanje komplikovano. Zbog toga su učinjeni mnogi pokušaji da se izgradi metod koji ima preimućstva nad gradijentnim metodom prvog reda i nad Newtonovim metodom, ali da koristi samo prve izvode.

Razrađene su dve grupe takvih metoda: *metod konjugovanih gradijenata* i *kvazi-newtonovski metodi*. Metodi konjugovanih gradijenata se koriste pre svega za kvadratne funkcije. U kvazi-newtonovskim metodima definiše se aproksimacija Hessove matrice ili njene inverzne matrice, ali samo pomoću prvih izvoda funkcije $Q(\mathbf{x})$. Od ovih metoda izložićemo tzv. Davidon-Fletcher-Powell (DFP) metod.

2.6.2. METOD DAVIDON-FLETCHER-POWELL (DFP)

U ovom metodu se umesto matrice $H^{-1}(\mathbf{x}^{(k)})$ koristi njena adekvatna aproksimacija $\eta(\mathbf{x}^{(k)})$, tako da iterativna procedura ima oblik

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \pm h^{(k)}\eta(\mathbf{x}^{(k)})\nabla Q(\mathbf{x}^{(k)}). \tag{2.6.2}$$

Ponekad se kvazi-newtonovski metodi nazivaju *gradijentni metodi sa velikim korakom* ili *metodi sa promenljivom metrikom*, jer se matrica $\eta(\mathbf{x}^{(k)}) = \eta^{(k)}$ menja kroz svaku iteraciju. Velika grupa kvazi-newtonovskih metoda zasnovana je na promeni matrice $\eta(\mathbf{x})$ u svakoj iteraciji, što je definisano pomoću rekurentne relacije

$$\eta^{(k+1)} = \eta^{(k)} + \Delta\eta^{(k)}.$$

U ovoj jednakosti $\Delta\eta^{(k)}$ je matrica koja koriguje smer pretraživanja. Način izračunavanja korigujuće matrice $\Delta\eta^{(k)}$ određuje modifikaciju kvazi-newtonovskog metoda.

U DFP metodu korigujuća matrica $\Delta\eta^{(k)}$ i $\eta^{(k+1)}$ se izračunavaju prema formulama

$$\eta^{(k+1)} = \eta^{(k)} + \mathbf{A}^{(k)} - \mathbf{B}^{(k)}, \tag{2.6.3}$$

gde su

$$\mathbf{A}^{(k)} = \frac{(\Delta\mathbf{x}^{(k)})(\Delta\mathbf{x}^{(k)})^T}{(\Delta\mathbf{x}^{(k)})^T(\Delta\mathbf{q}^{(k)})}, \tag{2.6.4}$$

$$\mathbf{B}^{(k)} = \frac{\eta^{(k)}(\Delta\mathbf{g}^{(k)})(\Delta\mathbf{g}^{(k)})^T(\eta^{(k)})^T}{(\Delta\mathbf{g}^{(k)})^T\eta^{(k)}(\Delta\mathbf{g}^{(k)})}, \tag{2.6.5}$$

$$\Delta\mathbf{x}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}, \tag{2.6.6}$$

$$\Delta\mathbf{g}^{(k)} = \nabla Q(\mathbf{x}^{(k+1)}) - \nabla Q(\mathbf{x}^{(k)}). \tag{2.6.7}$$

Kao i u Newtonovom metodu, neophodno je da matrica $\eta^{(k+1)}$ bude pozitivno definitna.

Algoritam DFP metoda može se iskazati pomoću sledećih koraka:

Korak 1. Zadati početnu tačku $\mathbf{x}^{(0)}$, maksimalan broj iteracija N_k i konstante ε_1 i ε_{2_i}, $i = 1, \ldots, n$.

Korak 2. Staviti $k = 0$.

Korak 3. Za početnu matricu $\eta^{(0)}$ uzeti jediničnu matricu.

Korak 4. Izračunati $\nabla Q(\mathbf{x}^{(0)})$.

Korak 5. $k := k + 1$.

Korak 6. Formirati novu funkciju po simbolu h na sledeći način:

$$f(h) = Q\left(\mathbf{x}^{(k)} - h * \eta(\mathbf{x}^{(k)})\nabla Q(\mathbf{x}^{(k)})\right).$$

Ovo je u skladu sa rekurentnom formulom (2.6.2).

Korak 7. Izračunati $h^{(k)}$ iz uslova minimizacije $f(h)$, tj. $f(h^{(k)}) = \min\limits_{h} f(h)$, pri čemu koristiti neki od metoda jednodimenzionalne optimizacije.

Korak 8. Koristeći formulu (2.6.2) i izračunatu vrednost $h^{(k)}$, odrediti vektor $\mathbf{x}^{(k+1)}$ i vrednost $Q(\mathbf{x}^{(k+1)})$.

Korak 9. Izračunati $\nabla Q(\mathbf{x}^{(k+1)})$.

Korak 10. Na osnovu formula (2.6.6) i (2.6.7), izračunati $\Delta\mathbf{x}^{(k)}$ i $\Delta\mathbf{g}^{(k)}$.

Korak 11. Proveriti jedan od kriterijuma za prekid pretraživanja:

(i) $$\frac{|\Delta Q(\mathbf{x}^{(k)})|}{Q(\mathbf{x}^{(k)})} = \left|\frac{Q(\mathbf{x}^{(k+1)}) - Q(\mathbf{x}^{(k)})}{Q(\mathbf{x}^{(k)})}\right| < \varepsilon_1,$$

(ii) $$|\Delta x_i^{(k)}| < \varepsilon_{2_i}, \text{ ako } x_i \to 0,$$

(iii) $$k > N_k.$$

Korak 12. Ako nije ispunjen kriterijum za prekid pretraživanja izračunati $\eta^{(k+1)}$ po formulama (2.6.3), (2.6.4) i (2.6.5), a zatim se vratiti se na *Korak 4*.

```
Dfp[q_,prom_List,x0p_List,eps_]:=
   Block[{grad1,hk,n=Length[prom],m0,m1,grad2,g01,s,x01,
     ng=eps+1,p,r,Nk=100,i,t,a,b,k=0,x0=x0p,q0,x1,q1,xh,qm,
                  izb,qexp,h,metod, Lista={}},
     izb=Input["Zelite li minimum(1) ili maksimum(1)?"];
     Lista=Append[Lista,x0];
     (* Korak 3 *)
     m0=IdentityMatrix[n];
     (* Korak 4 *)
     grad1=nabl[q,prom,x0];
     izb=Input["Unesi <1> za minimum, <2> za maksimum"];
     Print["Izaberi jednoimenzionalnu optimizaciju."];
```

```
Print["<1> skeniranje konstantnim korakom"];
Print["<2> skeniranje promenljivim korakom"];
Print["<3> simplexI metod"];
Print["<4> simplexII metod"];
Print["<5> zlatni presek"];
Print["<6> metod dihotomije"];
Print["<7> DSC metod"];
Print["<8> Powellov metod"];
Print["<9> DSC-Powellov metod"];
metod=Input[];
(* Korak 11 *)
While[Nk>k && Abs[hk]>=eps && Abs[ng]>=eps,
  q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
  (* Korak 5 *)
  k++;
  (* Korak 6 *)
  If[izb==1, xh=x0-h*(m0.grad1), xh=x0+h*(m0.grad1)  ];
  qexp=q;    Do[qexp=qexp/.prom[[i]]->xh[[i]], {i,n}];
  (* Korak 7 *)
  Which[metod==1,hk=skk[qexp,{h},0,1,0.01],
    metod==2,hk=spk[qexp,{h},0,1,0.5,eps/10],
    metod==3,hk=simplexI[qexp,{h},0,1,0.5,eps/10],
    metod==4,hk=simplexII[qexp,{h},0,0.1,eps/10],
    metod==5,hk=zlatni[qexp,{h},0,1,eps/10],
    metod==6,hk=dih[qexp,{h},0,1,eps/10],
    metod==7,hk=Dsk[qexp,{h},0.,0.1,eps/10],
    metod==8,hk=Powel[qexp,{h},0.,0.1,eps/10],
    metod==9,hk=dskpowel[qexp,{h},0.,0.1,eps/10]
  ];
  hk=hk[[1]];  Print["Korak je hk=",hk];
  (* Korak 8 *)
  If[Abs[hk]>=eps,
      If[izb==1,
         x1=x0-hk*(m0.grad1), x1=x0+hk*(m0.grad1) ];
      q1=q; Do[q1=q1/.prom[[i]]->x1[[i]],{i,n}];
      (* Korak 10 *)
      grad2=nabl[q,prom,x1];   g01=grad2-grad1;
      (* Korak 12 *)
      If[izb==1,x01=-hk*(m0.grad1),x01=hk*(m0.grad1) ];
```

```
            p=Partition[x01,1].{x01}; r=x01.g01;
            a=N[(1/r) p];
            s=m0.Partition[g01,1].{g01}.Transpose[m0];
            t=g01.m0.g01;  b=N[(1/t) s];     m1=m0+a-b;
            If[(izb==2&&q1>q0) || (izb==1&&q1<=q0),
              grad1=grad2; x0=x1; q0=q1; m0=m1;
              Lista=Append[Lista,x0];
      ]   ]
    ];
    {x0,q0,Lista}
]
```

Algoritmi kojima je implementiran DFP metod u programskom jeziku LISP opisani su u [51].

Konvergencija ovog algoritma se može dokazati samo u slučaju kvadratne ciljne funkcije sa pozitivno definitnom Hesseovom matricom. U ovom metodu se mogu koristiti numerički izračunate vrednosti, ali ne i analitički izrazi za $\dfrac{\partial Q}{\partial x_i}$.

U nizu test primera vršena je uporedna analiza kvazi-newtnovskih metoda koji koriste analitičke izraze za parcijalne izvode ciljne funkcije sa kvazi-newtnovskim metodima koji koriste numeričke aproksimacije za parcijalne izvode ciljne funkcije. Praksa pokazuje da u slučaju numeričkih izračunavanja parcijalnih izvoda, ponekad se ne stiže do rešenja zadatka. Ova činjenica ide u prilog opravdanju simboličke implementacije metoda numeričke optimizacije.

2.6.3. METOD KONJUGOVANIH GRADIJENATA

Sukscesivne aproksimacije za minimum ciljne funkcije Q u *metodu konjugovanih smerova* generišu se pomoću iterativne formule

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \sigma_k \mathbf{d}^k$$

pri čemu je σ_k optimalna dužina koraka u smeru $\mathbf{d}^k$, odnosno rešenje jednodimenzionalnog optimizacionog problema

$$\min_{\sigma>0} Q\left(\mathbf{x}^k + \sigma \mathbf{d}^k\right).$$

Osim toga je

$$\mathbf{d}^0 = -\nabla Q\left(\mathbf{x}^0\right),$$
$$\mathbf{d}^k = -\nabla Q\left(\mathbf{x}^k\right) + \gamma_k \mathbf{d}^{k-1},$$
$$\gamma_k = \frac{\|\nabla Q\left(\mathbf{x}^k\right)\|^2}{\|\nabla Q\left(\mathbf{x}^{k-1}\right)\|^2}.$$

Iz poslednje dve jednakosti zaključujemo da je

$$\mathbf{d}^k = -\nabla Q(\mathbf{x}^k) + \frac{\langle \nabla Q(\mathbf{x}^k) \nabla Q(\mathbf{x}^k) \rangle}{\langle \nabla Q(\mathbf{x}^{k-1}) \nabla Q(\mathbf{x}^{k-1}) \rangle} \mathbf{d}^{k-1},$$

gde $\langle \cdot \rangle$ označava skalarni proizvod dva vektora.

Početna aproksimacija $\mathbf{x}^0$ je proizvoljna. Algoritam se prekida u slučaju kada je $\|\mathbf{d}^k\| < \varepsilon$, pri čemu je ε unapred zadat mali pozitivan broj.

Algoritam metoda *konjugovanih gradijenata* može se iskazati sledećim koracima:

Korak 1. Inicijalizacija: postaviti $k = 0$, izabrati startnu tačku $\mathbf{x}^{(0)}$, unapred definisanu toleranciju ε i izračunati $Q(\mathbf{x}^{(0)})$ i $d_0 = \nabla Q(\mathbf{x}^{(0)})$.

Korak 2. Ako je $\|\nabla Q(\mathbf{x}^{(k)})\| < \varepsilon$, prekinuti algoritam; inače, preći na sledeći korak.

Korak 3. U k-toj iteraciji odrediti minimum funkcije $Q(\mathbf{x})$ koristeći jednodimenzionalno pretraživanje u pravcu $\mathbf{d}_k$, tj. izračunati tekuću dužinu koraka α_k koja ispunjava uslov

$$\min_{\alpha>0} Q(\mathbf{x}^{(k)} - \alpha \mathbf{d}_k) = \min_{\alpha>0} Q\left(\mathbf{x}^{(k)} - \alpha \nabla Q(\mathbf{x}^{(k)})\right) \tag{2.4.20}$$

za slučaj minimuma, odnosno

$$\min_{\alpha>0} Q(\mathbf{x}^{(k)} + \alpha \mathbf{d}_k) = \min_{\alpha>0} Q\left(\mathbf{x}^{(k)} + \alpha \nabla Q(\mathbf{x}^{(k)})\right) \tag{2.4.21}$$

za slučaj maksimuma.

Korak 4. Odrediti $\mathbf{x}^{(k+1)}$ pomoću:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \mathbf{d}_k.$$

Korak 5. Izračunati $Q(\mathbf{x}^{(k+1)})$ i $\nabla Q(\mathbf{x}^{(k+1)})$.

Korak 6. Izračunati tekući pravac $\mathbf{d}_{k+1}$:

$$\mathbf{d}_{k+1} = -\nabla Q(\mathbf{x}^{(k)}) + \lambda_k \mathbf{d}_k,$$

gde je $\lambda_0 = \mathbf{0}$ i

$$\lambda_k = \frac{\|\nabla Q(\mathbf{x}^{(k+1)})\|^2}{\|\nabla Q(\mathbf{x}^{(k)})\|^2}.$$

Korak 7. Postaviti $\mathbf{x}^{(k)} = \mathbf{x}^{(k+1)}$, $\mathbf{d}_k = \mathbf{d}_{k+1}$, $k := k+1$, izračunati $Q(\mathbf{x}^{(k)})$ i $\nabla Q(\mathbf{x}^{(k)})$, a zatim se vratiti na *Korak 2*.

Sledi implementacija u paketu MATHEMATICA. Dovoljno je napomenuti da je implementacija *Koraka 3* i *4* analogna opisanoj implementaciji koja je korišćena kod Cauchyevog i DFP metoda.

```
Mkg[q_,prom_List,x01_List,eps_]:=
   Block[{q0,q1,d0,d1,x0=x01,it,A=a,p,x1,grad1,grad2,h,hk,
         n=Length[prom], qm,tacka,izbor,ind,qexp,
         metod,Lista={}},
        izbor=Input["Unesi <1> za minimum, <2> za maksimum"];
        Print["Izaberi jednoimenzionalnu optimizaciju."];
        Print["<1> skeniranje konstantnim korakom"];
        Print["<2> skeniranje promenljivim korakom"];
        Print["<3> simplexI metod"];
        Print["<4> simplexII metod"];
        Print["<5> zlatni presek"];
        Print["<6> metod dihotomije"];
        Print["<7> DSC metod"];
        Print["<8> Powelov metod"];
        Print["<9> DSC-Powelov metod"];
        metod=Input[];
        q0=q;
        Do[q0=q0/.prom[[i]]->x0[[i]],{i,n}];
        Lista=Append[Lista,x0];
        grad1=nabl[q,prom,x0];
        d0=grad1;  it=0;
        While[N[norma[d0]]>=eps && it<50,
          it++; Print["it=",it];
          qexp=q;
          If[izbor==1,
            Do[qexp=qexp/.prom[[i]]->
```

```
            N[x0[[i]]]-h*N[d0[[i]]],{i,n}],
        Do[qexp=qexp/.prom[[i]]->
            N[x0[[i]]]+h*N[d0[[i]]],{i,n}]
      ];
      Which[metod==1,hk=skk[qexp,{h},0,1,0.01],
          metod==2,hk=spk[qexp,{h},0,1,0.5,eps/10],
          metod==3,hk=simplexI[qexp,{h},0,1,0.5,eps/10],
          metod==4,hk=simplexII[qexp,{h},0,0.1,eps/10],
          metod==5,hk=zlatni[qexp,{h},0,1,eps/10],
          metod==6,hk=dih[qexp,{h},0,1,eps/10],
          metod==7,hk=Dsk[qexp,{h},0.,0.1,eps/10],
          metod==8,hk=Powel[qexp,{h},0.,0.1,eps/10],
          metod==9,hk=dskpowel[qexp,{h},0.,0.1,eps/10]
      ];
      hk=hk[[1]];
      Print["Korak =",hk,"x0=",N[x0],"d0=",N[d0]];
      If[s>=eps,
      If[izbor==1, x1=N[x0-hk*d0], x1=N[x0+hk*d0]];
         q0=q; Do[q0=q0/.prom[[i]]->x0[[i]],i,n];
         q1=q; Do[q1=q1/.prom[[i]]->x1[[i]],i,n];
         grad2=nabl[q,prom,x1];
         d1=+grad2-N[(grad2.grad2)/(grad1.grad1)] d0;
         d0=N[d1]; x0=N[x1]; q0=N[q1];
         Lista=Append[Lista,x0];
         grad1=grad2,
         it=50
      ]
    ];
    {N[x0],N[q0],Lista}
]
```

Može se pokazati da je metod konjugovanih gradijenata jedan od metoda promenljive metrike u kojoj se inverzija Hessiana generiše ppomoću

$$H_k = I - \frac{\| \left[\nabla Q\left(\mathbf{x}^k\right)\right]^T \| d^k}{\|\nabla Q\left(\mathbf{x}^{k-1}\right)\|^2}.$$

Metodi konjugovanih gradijenata se odlikuju kvadratnim završavanjem. Međutim, oni su osetljivi na izbor dužine koraka. Ukoliko dužine koraka nisu adekvatne, ovi metodi mogu postati neefikasni i inferiorni u odnosu na druge metode promenljive metrike.

Pozitivna osobina metoda konjugovanih gradijenata je da ne koristi matrice koje mogu da budu loše uslovljene za računanje smerova pretraživanja.

Opis implementacije metoda *konjugovanih gradijenata* u jeziku LISP opisan je u [50].

2.7. Poređenje gradijentnih i negradijentnih metoda

U ovom odeljku navešćemo neke prednosti i nedostatke gradijentnih i negradijentnih metoda, kao i neka dodatna zapažanja.

Prednosti gradijentnih metoda:

1° Brza konvergencija za analitički zadate parcijalne izvode, što se posebno odnosi na algoritme koji koriste parcijalne izvode drugog reda.

2° Gradijentni metodi drugog reda imaju brzu konvergenciju za kvadratne funkcije ili funkcije koje su bliske kvadratnim.

Nedostaci gradijentnih metoda:

1° Gradijentni metodi lokalizuju ekstremum koji je najbliži početnoj tački.

2° Mogućnost konvergencije prema sedlastoj tački za funkcije sa malom osetljivošću ekstremuma (plato) ili za jaružne ciljne funkcije.

3° Teško se odabiraju parametri neophodni za algoritam, kao što su parametri za numeričko diferenciranje δx_i ili parametri za prekid algoritma ε_i.

4° Smer kretanja prema ekstremumu i brzina konvergencije zavise od vrednosti upravljačkih parametara i veličine parametara koraka h_i.

5° Za veliki broj upravljačkih parametara zahtevaju veliki broj izračunavanja vrednosti ciljne funkcije.

Prednosti negradijentnih metoda:

1° Laki su za algoritmizaciju i programsku implementaciju.

2° Kriterijumi za prekid procesa se lako ostvaruju.

3° Neke od tih metoda imaju visoku efektivnost za veliki broj upravljačkih parametara.

4° Ne nameću se uslovi za oblik ciljne funkcije i njene izvode.

Nedostaci negradijentnih metoda:

1° U nekim optimizacionim zadacima broj izračunavanja vrednosti ciljne funkcije je veliki u odnosu na gradijentne metode.

2° Konvergiraju prema lokalnom ekstremumu.

Neka zapažanja za gradijentne metode prvog reda:

1° Osnovni gradijentni metod i njegova modifikacija konvergiraju mnogo sporije i nepouzdanije od metoda najstrmijeg pada. Ako se koristi fiksirana ili podesiva vrednost za skalar h, ta vrednost mora da se pažljivo kontroliše da bi se izbeglo neočekivano udaljavanje od ekstremuma ciljne funkcije ili preterano veliki broj iteracija u dostizanju ekstremuma. Prvi slučaj se događa kod prevelikog koraka h, dok se drugi slučaj sreće ako je h isuviše malo ili ako je h preveliko, tako da dolazi do oscilovanja oko ekstremuma.

2° Tačnost $\|\nabla Q(x^{(k)})\| \leq \varepsilon$ se teško dostiže.

3° Može se dogoditi da metod divergira i tada vrednosti ciljne funkcije osciluju između dve vrednosti.

4° Brzina konvergencije znatno zavisi od izbora početne tačke.

5° Ukoliko se izabere manji priraštaj argumenta prilikom numeričkog diferenciranja, približavanje prema optimalnoj tački je sporije, ali sigurnije, što dovodi do bolje tačnosti rezultata.

6° Cauchyev metod najstrmijeg pada ima mnogo bržu konvergenciju u odnosu na osnovni gradijentni metod i njegove modifikacije. Problem ovog metoda je u tome da se ne može uvek jednostavno naći rešenje jednodimenzionalnog optimizacionog problema.

3. GLOBALNA OPTIMIZACIJA

3.1. Uvod

Ciljna funkcija koja u dozvoljenoj oblasti ima više od jednog ekstremuma naziva se *mnogoekstremalna* ili *multimodalna*. Ovi ekstremumi se nazivaju *lokalni*, a najbolja (najveća ili najmanja) vrednost između njih se naziva *globalni ekstremum*. Funkcija $Q(\mathbf{x})$, definisana unutar dozvoljene n-dimenzionalne oblasti Γ_x, ima apsolutni (globalni) ekstremum, na primer maksimum, u tački $\mathbf{x}^*$, u oznaci

$$Q(\mathbf{x}^*) = \max_{\mathbf{x}} Q(\mathbf{x}), \quad \mathbf{x} \in \Gamma_x,$$

ako je za sve $\mathbf{x} \in \Gamma_x$ ispunjena nejednakost

$$Q(\mathbf{x}^*) > Q(\mathbf{x}).$$

Funkcija Q se naziva mnogoekstremalna i ima lokalni maksimum ako može da se odredi podoblast $\Gamma_\epsilon \subset \Gamma_{\mathbf{x}}$ takva da za svako $\mathbf{x} \in \Gamma_\epsilon$ postoji tačka $\mathbf{x}^l \in \Gamma_\epsilon$ za koju važe sledeće dve nejednakosti:

$$Q(\mathbf{x}^l) > Q(\mathbf{x}), \quad Q(\mathbf{x}^*) > Q(\mathbf{x}^l).$$

Oblast Γ_ϵ se naziva oblast lokalnog ekstremuma $\mathbf{x}^l$. Dva ili više lokalna ekstremuma sa jednakim vrednostima se nazivaju *ekvivalentni*. Prema tome, neki optimizacioni zadatak može da se tretira kao mnogoekstremalni ako postoje ne manje od dva neekvivalentna ekstremuma.

Do sada je razrađen veliki broj metoda za nalaženje globalnog ekstremuma. Za mnoge od njih je matematički dokazana konvergencija, ali do sada nije razvijen numerički algoritam koji za razuman broj iteracija garantuje nalaženje globalnog ekstremuma. Za korektno i pouzdano izračunavanje globalnog ekstremuma neophodno je da se pronađu sve lokalne ekstremne vrednosti i da se između njih izabere najveća. U realnim tehničkim zadacima broj lokalnih ekstremuma može da bude veliki (na stotine), te je takav metod nepraktičan. Neophodno je da se sprovede algoritam koji izračunava globalni ekstremum, bez pronalaženja svih ekstremnih vrednosti.

Pretraživanje globalnog ekstremuma kraće se naziva *globalno pretraživanje*. Metodi za globalnu optimizaciju klasifikuju se po različitim kriterijumima, pri čemu su najvažnije sledeće grupe: *gradijentni, stohastički, kombinovani* i *heuristički*. Čisto gradijentni metodi su malo korišćeni za *globalno pretraživanje*. Najpoznatiji je metod "teškog topa" u različitim modifikacijama. Najrasprostranjeniji su metodi za slučajno pretraživanje. Njihova osnovna prednost nad ostalim metodima je široka moguća oblast za ispitivanje od zadate tačke, mali broj izračunavanja, itd.

Do sada je među metodima za nalaženje lokalnog ekstremuma razrađen jedan metod za nalaženje globalnog ekstremuma - metod skaniranja sa konstantnim i malim korakom. Međutim, neophodnost skaniranja sa veoma malim korakom, koji garantuje da se globalni ekstremum neće propustiti, ograničava primenu ovog metoda. On se može iskoristiti samo za slučaj malog broja upravljačkih parametara ($n \leq 3, 4$) i lako izračunljivom ciljnom funkcijom.

3.2. Metodi i implementacija

3.2.1. SLUČAJNO PRETRAŽIVANJE SA SKANIRANIM POČETNIM TAČKAMA

U ovom metodu su zadate granice $[xmin_i, xmin_i]$, $i = 1, \ldots, n$, za svaki upravljački parametar. Algoritam se može iskazati kroz sledeće korake:

Korak 1. Izračunati korak skaniranja po svakom upravljačkom parametru. Preporučuje se

$$\Delta x_i = \frac{xmax_i - xmin_i}{L}, \qquad i = 1, \ldots, n,$$

za $L = 4,5$.

Korak 2. Sukscesivno izvršavati skeniranje u dozvoljenoj oblasti $xmin_i \leq x_i \leq xmax_i$, $i = 1, \ldots, n$, pri čemu se svaka tačka mreže za skeniranje uzima za početnu tačku $x0_i$.

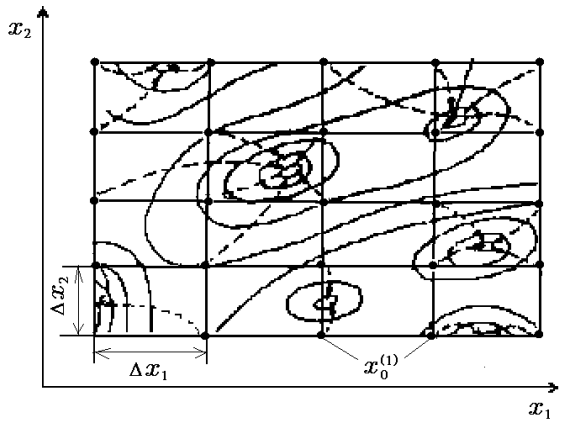
Korak 3. Od svake početne tačke izvršiti slučajno pretraživanje (neko od ranije opisanih) i zapamtiti koordinate koje daju najbolje vrednosti ciljne funkcije.

Korak 4. Globalni ekstremum je najbolja vrednost od svih izračunatih lokalnih ekstremuma.

Implementacija ovog algoritma, za slučaj dve promenljive, u programskom jeziku MATHEMATICA data je sledećim programom:

```
Metod1[q_,var_List,x1min_Real,x2min_Real,
                 x1max_Real,x2max_Real]:=
   Block[{delta1,delta2,q1,W,pom,i,j,izbor,tacka,x1,x2,
          d1=x1min,g1=x1max,d2=x2min,g2=x2max,qnaj,
          a={},b,hmin,x0,r},
    izbor=Input["Zelite li minimum(1) ili maksimum(2)?"];
    delta1=(x1max-x1min)/2; delta2=(x2max-x2min)/2;
    (* r broji pocetne tacke za slucajno pretrazivanje *)
    (* x0 je lista tacaka za skeniranje *)
    r=0; x0={};
    For[x1=d1,x1<=g1,x1+=delta1,
     For[x2=d2,x2<=g2,x2+=delta2, x0=Append[x0,x1,x2]; r+=1]
    ];
    hmin=Input["Unesite minimalnu vrednost hmin u listi"];
    b=Input["Unesite vrednost kojom se koraci skracuju"];
    tacka=Table[0,r,2]; W=Table[0,{r}];
    Do[a=x0[[i]];
       tacka[[i]]=First[SluOb[q,var,a,{delta1,delta2},
              hmin,b,{x1min,x2min},{x1max,x2max},izbor]];
       q1=q; pom=tacka[[i]];
       Do[q1=q1/.var[[j]]->pom[[j]],{j,2}]; W[[i]]=q1;
       {i,r}
    ];
    If[izbor==2, qnaj=Max[W], qnaj=Min[W] ];
    j=1;  While[W[[j]]!=qnaj,j++];
    Return[{tacka[[j]],W[[j]]}]
   ]
```

Grafička interpretacija ovog metoda za dva upravljačka parametra je prikazana na slici 3.2.1. Put od svake početne tačke prema lokalnom ekstremumu, prikazan je isprekidanom linijom.



Sl. 3.2.1

Pri povećanju broja upravljačkih parametara povećava se i broj početnih tačaka, samim tim i opšti broj neophodnih izračunavanja pri traženju ekstremuma. Zbog toga je ovaj metod preporučljiv za mali broj upravljačkih parametara $n \leq 4, 5$.

Numerički rezultati.

```
In[1]:= Metod1[2*x+yˆ2*Sin[x-y],{x,y},0.1,0.4,5.0,7.5]
Zelite li minimum(1) ili maksimum(2)? 1
d1= 2.45 d2 = 3.55
Unesite minimalnu vrednost hmin u obliku liste {0.01,0.01}
Unesite vrednost kojom se koraci skracuju b= 4
x0[[1]] = {0.1, 0.4}
1 za minimum, 2 za maksimum 1
{{1.63946, 2.30492}, -0.00125122}
{{2.26141, 4.33353}, -11.9458}
{{4.4097, 5.75353}, -23.4348}
{{5., 7.17353}, -32.3917}
{{3.69285, 5.71547}, -22.0029}
{{4.59578, 6.81853}, -27.7651}
{{4.89394, 7.08478}, -31.0627}
{{4.97002, 7.15135}, -31.9626}
{{4.99239, 7.1678}, -32.2844}
{{4.99862, 7.17215}, -32.3784}
```

tacka={4.99862, 7.17215}
q[x0[[i]]]= -32.3784
x0[[2]] = {0.1, 3.95}

1 za minimum, 2 za maksimum 1
{{0.1, 2.53}, -3.98005}
{{0.1, 0.4}, 0.152717}
{{0.429579, 2.885}, -4.41427}
. . .

{{3.3986, 5.42637}, -19.6269}
{{3.40006, 5.42878}, -19.6351}
{{3.40102, 5.43017}, -19.6412}
{{3.40188, 5.43156}, -19.6461}
{{3.40299, 5.43294}, -19.6537}
{{3.40222, 5.4304}, -19.6537}

tacka={3.40222, 5.4304}
q[x0[[i]]]= -19.6537 x0[[3]] = {0.1, 7.5}

1 za minimum, 2 za maksimum 1
{{0.1, 5.44077}, 24.1474}
{{0.1, 7.145}, -35.0372}
{{0.1, 7.394}, -46.1212}
{{0.1, 7.47781}, -49.4973}
{{0.1, 7.49114}, -50.0128}
{{0.1, 7.49861}, -50.2994}

tacka={0.1, 7.49861} q[x0[[i]]]= -50.2994 x0[[4]] = {2.55, 0.4}

1 za minimum, 2 za maksimum 1
{{1.52522, 0.4}, 3.19482}
{{0.1, 0.4}, 0.152717}
{{0.1, 0.4}, 0.152717}
{{0.279122, 1.12439}, -0.387614}
{{0.340372, 1.91541}, -2.98801}
{{0.419942, 2.65035}, -4.71097}
{{0.803343, 3.00535}, -5.68507}
{{1.17478, 3.37884}, -6.85335}
. . .
{{1.7646, 3.85844}, -9.36795}
{{1.76597, 3.86077}, -9.3736}
{{1.76555, 3.85776}, -9.37359}

tacka={1.76555, 3.85776} q[x0[[i]]]= -9.37359 x0[[5]] = {2.55, 3.95}

1 za minimum, 2 za maksimum 1
{{3.31281, 5.37}, -18.8668}
{{4.96145, 6.79}, -34.6581}
{{3.23786, 5.37}, -17.9358}
{{4.79576, 6.38095}, -31.1208}
{{5., 6.9501}, -34.8706}

{{4.91416, 6.81316}, -34.113}
{{4.97366, 6.92792}, -34.5629}
{{4.99337, 6.94456}, -34.7929}
{{4.99976, 6.9467}, -34.8834}
{{4.99762, 6.94531}, -34.8564}

tacka={4.99762, 6.94531} q[x0[[i]]]= -34.8564 x0[[6]] = {2.55, 7.5}

1 za minimum, 2 za maksimum 1
{{2.09014, 4.8621}, -4.36022}
{{3.99013, 6.64427}, -12.6968}
{{5., 7.5}, -23.6641}
{{5., 7.5}, -23.6641}
{{5., 7.5}, -23.6641}
{{5., 7.5}, -23.6641}
{{5., 7.5}, -23.6641}
{{5., 7.5}, -23.6641}
{{5., 7.5}, -23.6641}

tacka={5., 7.5}
q[x0[[i]]]= -23.6641 x0[[7]] = {5., 0.4}

1 za minimum, 2 za maksimum 1
{{3.47656, 0.4}, 6.96352}
{{2.95282, 0.4}, 5.99449}
{{2.70782, 0.4}, 5.53411}
{{0.397538, 0.4}, 0.794682}
{{0.1, 0.4}, 0.152717}
{{1.17101, 2.85733}, -5.76787}
{{2.43919, 4.27733}, -12.7672}
{{4.25787, 5.69898}, -23.6899}
{{5., 7.11898}, -33.2538}
{{4.01001, 5.69898}, -24.2319}
{{4.59886, 6.63246}, -30.1643}
{{4.85095, 7.03023}, -30.8512}
{{4.97044, 7.0968}, -32.8492}
{4.99593, 7.11225}, -33.2502}
{{4.99889, 7.1176}, -33.2464}

tacka={4.99889, 7.1176} q[x0[[i]]]= -33.2464 x0[[8]] = {5., 3.95}

1 za minimum, 2 za maksimum 1
{{5., 6.7504}, -34.8349}
{{3.16047, 5.3304}, -17.1434}
{{4.70784, 6.32089}, -30.5023}
{{5., 6.91704}, -35.006}
{{4.92631, 6.82829}, -34.2392}
{{4.98243, 6.88114}, -34.8623}
{{4.99633, 6.91149}, -34.9715}
{{4.99886, 6.91439}, -34.9983}

tacka={4.99886, 6.91439} q[x0[[i]]]= -34.9983 x0[[9]] = {5., 7.5}

```
1 za minimum, 2 za maksimum 1
{{4.00548, 5.29516}, -18.927}
{{4.88917, 6.81154}, -33.7807}
{{4.76936, 6.45654}, -31.8662}
{{4.98222, 6.93043}, -34.686}
{{4.9065, 6.78422}, -34.0618}
{{5., 6.95261}, -34.8579}
{{4.97687, 6.93043}, -34.6014}
{{4.99181, 6.94707}, -34.755}
{{4.99948, 6.94955}, -34.865}
{{4.99911, 6.94693}, -34.8721}
{{5., 6.94904}, -34.8758}
{{4.99891, 6.94701}, -34.8685}
tacka={4.99891, 6.94701}  q[x0[[i]]]= -34.8685
W={-32.3784,-19.6537,-50.2994,-9.37359,-34.8564,-23.6641,-33.2464,-34.9983,-34.8685}
Out[1]= {{0.1, 7.49861}, -50.2994}
```

3.2.2. SLUČAJNO PRETRAŽIVANJE IZ SKUPA SLUČAJNIH POČETNIH TAČAKA

Ako se pretpostavi da su lokalni ekstremumi ravnomerno raspoređeni u nekoj dozvoljenoj oblasti, pretraživanje može da se organizuje počev od ravnomerno raspoređenih slučajnih tačaka u toj oblasti. Algoritam pretraživanja je tada sledeći:

Korak 1. Generisati početnu tačku unutar dozvoljene oblasti $[\mathbf{xmin}, \mathbf{xmax}]$ na slučajan način:

$$x0_i = xmin_i + \alpha_i(xmax_i - xmin_i)\,, \qquad i = 1, \ldots, n,$$

pri čemu je α_i slučajan broj u intervalu $[0, 1]$. Broj neuspešnih pretraživanja postaviti na 0.

Korak 2. Počev od početne tačke $\mathbf{x}_0$ izvršiti slučajno pretraživanje ekstremuma i zapamtiti rezultat.

Korak 3. Generisati novu početnu slučajnu tačku i izvršiti *Korak 2* ovog algoritma. Ako je izračunati lokalni ekstremum bolji od prethodnog, on se pamti umesto prethodnog. Ako je izračunati lokalni ekstremum lošiji od prethodnog, povećava se broj neuspešnih pretraživanja za jedinicu i algoritam se nastavlja od *Koraka 1.*

Korak 4. Kriterijum za prekid pretraživanja je dostizanje zadatog broja neuspešnih pretraživanja (koji ćemo označiti sa K_N) nakon poslednjeg najboljeg rezultata ciljne funkcije. Broj K_N se izračunava

prema empirijskoj formuli

$$K_N = \begin{cases} 2^n + 4, & \text{za } n \leq 3, \\ 2n + 4, & \text{za } n > 3. \end{cases}$$

Ako lokalni ekstremumi nisu ravnomerno raspoređeni, tj. ako se gomilaju u nekoj oblasti, naročito na granicama oblasti, verovatnoća nalaženja globalnog ekstremuma se smanjuje.

Ovaj metod je upotrebljiv i za veći broj upravljačkih parametara.

Sledeći program je implementacija ovog algoritma u programskom jeziku MATHEMATICA:

```
Metod2[q_,var_List,xmin_List,xmax_List]:=
   Block[{n,k,q0=q,q1=q,c,c1,l=0,izbor,alfa},
     Print["Zelite li minimum(1) ili maksimum(2)"];
     izbor=Input[]; n=Length[var];
     c=Table[0,{n}];c1=Table[0,{n}];
     If[n<=3,k=2^n+4,k=2*n+4];
     Do[alfa=Random[];
        c[[i]]=xmin[[i]]+alfa*(xmax[[i]]-xmin[[i]]),
         {i,n}
     ];
     c=limit[c,xmin,xmax];
     Do[q0=q0/.var[[i]]->c[[i]],{i,n}];
     While[l<k,
        Do[alfa=Random[];
          c1[[i]]=xmin[[i]]+alfa*(xmax[[i]]-xmin[[i]]),
          {i,n}
       ];
       c1=limit[c1,xmin,xmax];
       Do[q1=q1/.var[[i]]->c1[[i]],{i,n}];
       If[Or[And[q1>q0,izbor==2],And[q1<q0,izbor==1]],
          q0=q1; c=c1,
          l+=1
       ];
       q1=q;
     ];
     Return[{c,q0}]
   ]
```

Numerički rezultati.

```
In[1]:= Metod2[2*x+yˆ2*Sin[x-y],{x,y},{1.0,2.0},{3.0,4.0}]
Zelite li minimum(1) ili maksimum(2)
Zelite li minimum(1) ili maksimum(2) 1
tacka = {1.81901, 3.11227}
{{1.81901, 3.11227}, -5.67754}
tacka = {2.96403, 3.66739}
tacka = {1.41685, 3.52614}
{{1.41685, 3.52614}, -7.84039}
tacka = {1.49381, 3.17747}
tacka = {1.18545, 2.5532}
tacka = {1.80318, 2.71164}
tacka = {2.72822, 2.80788}
tacka = {2.70974, 2.7814}
tacka = {2.48753, 3.94771}
{{2.48753, 3.94771}, -10.5141}
tacka = {2.73322, 3.39859}
tacka = {2.99058, 2.09387}
Out[1]= {{2.48753, 3.94771}, -10.5141}
```

3.2.3. PRICEOV METOD

Priceov metod je heuristički metod. Unutar dozvoljene oblasti generiše se početni skup od M ravnomerno raspoređenih slučajnih tačaka. Na slučajan način se odabiraju sukcesivno grupe od elemenata tog skupa, analiziraju se i izračunavaju nove tačke, a zatim se zamenjuju najlošije tačke. Na taj način "nivo" najlošijeg rezultata $Q^{(w)}$ neprestano raste (pri nalaženju maksimuma za $Q(\mathbf{x})$). Pri takvom podizanju "nivoa" najlošijeg rezultata, tačke koje ostaju počinju da se grupišu oko globalnog maksimuma.

Priceov algoritam za izračunavanje maksimuma može se opisati na sledeći način:

Korak 1. U dozvoljenoj oblasti generiše se M ravnomerno raspoređenih slučajnih tačaka $\mathbf{x}^{(j)}$, sa koordinatama $x_i^{(j)}, i=1,\dots,n, j=1,\dots,M$, prema formuli

$$x_i^{(j)} = xmin_i + \alpha_i^{(j)}(xmax_i - xmin_i), \quad i = 1,\dots,n;\ j = 1,\dots,M.$$

Korak 2. Izračunava se $Q^{(j)} = Q(\mathbf{x}^{(j)}),\ j = 1,\dots,M.$

Korak 3. Staviti $K = 0,\ IT = 0.$

Korak 4. Odrediti tačku $\mathbf{x}^{(w)}$ koja daje najlošiji rezultat $Q^{(w)}$, kao i tačku $\mathbf{x}^{(b)}$ sa najboljim rezultatom $Q^{(b)}$, za funkciju $Q(\mathbf{x})$.

Korak 5. Proveriti kriterijum za završetak procesa

$$\|\mathbf{x}^{(b)} - \mathbf{x}^{(w)}\| \leq \varepsilon_x$$

Ukoliko je taj uslov ispunjen, prekinuti algoritam, a za izlaz uzeti $\mathbf{x}^{(b)}$ i $Q^{(b)}$.

Korak 6. Od M slučajnih tačaka na slučajan način izabrati $M_1 = n + 1$ tačaka. Od njih se opet slučajno odabira jedna, koja se naziva *pol* i označava sa $\mathbf{x}^{(p)}$. Izračunava se centar težišta $\mathbf{x}^{(c)}$ za ostalih $M_1 - 1 = n$ tačaka pomoću

$$x_i^{(c)} = \frac{1}{M_1 - 1}\left(\sum_{l=1}^{M_1} x_i^{(l)} - x_i^{(p)}\right), \qquad i = 1, \ldots, n.$$

Korak 7. Izračunava se tačka $\mathbf{x}^{(N)}$ pomoću

$$\mathbf{x}^{(N)} = 2\mathbf{x}^{(c)} - \mathbf{x}^{(p)}$$

i postavlja $IT = IT + 1$.

Korak 8. Izračunati vrednost $Q^{(N)} = Q(\mathbf{x}^{(N)})$.

Korak 9. Ako je $Q^{(N)} > Q^{(w)}$ odbaciti $\mathbf{x}^{(w)}$ i zameniti sa $\mathbf{x}^{(N)}$:

$$\mathbf{x}^{(w)} = \mathbf{x}^{(N)} \quad \text{i} \quad Q^{(w)} = Q^{(N)}.$$

Algoritam se nastavlja od *Koraka 5*.

Korak 10. Za $Q^{(N)} \leq Q^{(w)}$, postaviti $K = K + 1$.

Korak 11. Ako je $K < IT/2$, ignorisati $\mathbf{x}^{(N)}$ i produžiti od *Koraka 6*. Inače preći na *Korak 12*.

Korak 12. Izračunava se nova probna tačka

$$\mathbf{x}^{(G)} = \frac{1}{2}\left(\mathbf{x}^{(c)} + \mathbf{x}^{(N)}\right).$$

Korak 13. Izračunava se $Q^{(G)} = Q(\mathbf{x}^{(G)})$.

Korak 14. Ako je $Q^{(G)} \leq Q^{(w)}$ ignorisati $\mathbf{x}^{(G)}$ i produžiti od *Koraka 6*.

Korak 15. Ako je $Q^{(G)} > Q^{(w)}$ tačka $\mathbf{x}^{(w)}$ se zamenjuje tačkom $\mathbf{x}^{(G)}$:

$$\mathbf{x}^{(w)} = \mathbf{x}^{(N)} \quad \text{i} \quad Q^{(w)} = Q^{(N)}.$$

Algoritam se nastavlja od *Koraka 5*.

Potrebne su sledeće pomoćne funkcije:

```
izracunaj[q_,x_List,var_List]:=
    Block[{q1},
          q1=q; Do[q1=q1/.var[[i]]->x[[i]],i,n];
          Return[q1]
    ]
limitn[x_,xmin_,xmax_]:=
    Block[{x0=x},
        Which[x0<xmin,x0=xmin, x0>xmax,x0=xmax];
        Return[x0]
    ]
```

Sledeći program je implementacija prethodnog algoritma u paketu MATHEMATICA:

```
MetodPrice[q1_,var_List,xmin_List,xmax_List,eps_]:=
   Block[{q0,q=q1,uspesno,l,n,x1,M,x1best,p,
          k=0,IT=0,qmax,qmin,r,alfa,b=0,w=0,i,j,j1},
     n=Length[var];  M=n+8;
     x1=Table[0,{M},{n}]; q0=Table[0,{M}];
     Do[ Do[alfa=Random[];
          x1[[j,i]]=xmin[[i]]+alfa*(xmax[[i]]-xmin[[i]]);
          x1[[j,i]]=limit[x1[[j,i]],xmin[[i]],xmax[[i]]],
             {i,n}
         ],
         {j,M-3}
     ];
     Do[q0[[j1]]=izracunaj[q,x1[[j1]],var],{j1,M-3}];
     qmin=q0[[1]];
     Do[If[qmin>q0[[i]],qmin=q0[[i]]],{i,M-3}];
     qmax=Max[q0];
     Do[If[qmax==q0[[j]],b=j]; If[qmin==q0[[j]],w=j],
        {j,M-3}
     ];
     x1best=x1[[b]];
     While[Sqrt[Sum[(x1[[b,i]]-x1[[w,i]])^2,{i,n}]]>=eps,
       uspesno=True;
       While[uspesno,
         Do[ r=Random[Integer,{1,M-3}];
```

```
            Do[x1[[j,i]]=x1[[r,i]],{i,n}],
            {j,n}
          ];
          p=Random[Integer,{1,n+1}];
          Do[x1[[M-2,i]]=
            N[(Sum[x1[[l,i]],{l,n+1}]-x1[[p,i]])/n],
            {i,n}
          ];
          Do[x1[[M-2,i]]=
             limit[x1[[M-2,i]],xmin[[i]],xmax[[i]]],
             {i,n}
          ];
          Do[x1[[M-1,i]]=2*x1[[M-2,i]]-x1[[p,i]],
             {i,n}
          ];
          Do[x1[[M-1,i]]=
             limit[x1[[M-1,i]],xmin[[i]],xmax[[i]]],
             {i,n}
          ];
          IT=IT+1; q0[[M-1]]=izracunaj[q,x1[[M-1]],var];
          If[q0[[M-1]]>qmin,
            x1[[w]]=x1[[M-1]]; qmin=q0[[M-1]]; uspesno=False,
            k=k+1;
            If[k>=(IT/2),
              Do[x1[[M,i]]=
                N[(x1[[M-2,i]]+x1[[M-1,i]])/2],
                {i,n}
              ];
              Do[x1[[M,i]]=
                 limit[x1[[M,i]],xmin[[i]],xmax[[i]]],
                 {i,n}
              ];
              q0[[M]]=izracunaj[q,x1[[M]],var];
              If[q0[[M]]>qmin,
                 x1[[w]]=x1[[M]]; qmin=q0[[M]]; uspesno=False
 ]  ] ] ] ];
 Return[{(x1[[w]]+x1best)/2,(qmin+qmax)/2}]
]
```

Nedostatak ovog algoritma jeste odsustvo kriterijuma za dodeljivanje adekvatne vrednosti M. Price je predložio M između 1 i 100 za $N = 2, 3$. Neophodno je da se pronađe optimalan broj M u zavisnosti od broja upravljačkih parametara n. Uvećanjem broja M uvećava se i verovatnoća za nalaženje globalnog ekstremuma, ali i naglo raste broj potrebnih izračunavanja vrednosti ciljne funkcije Q.

Značajno povećanje brzine konvergencije ovog algoritma može se očekivati ako se algoritam, umesto na skup od M slučajno generisanih tačaka unutar dopustive oblasti, primenjuje na skup izračunatih lokalnih minimuma koji se dobijaju počev od zadatog broja slučajnih početnih tačaka.

3.2.4. METOD "TEŠKOG TOPA"

Za ovaj algoritam je karakteristično da se pri kretanju prema ekstremumu svakoj tački pridodaje masa. Na taj načina se tačka može posmatrati kao "mali top". Koristeći pridodatu masu, može da se preskoči neki od lokalnih minimuma koji nije mnogo izražen.

Rekurentna formula za kretanje prema minimumu je sledeća:

$$x_i^{(k+1)} = x_i^{(k)} - h_i^{(k)} \frac{\partial Q^{(k)}}{\partial x_i} + \beta_i^{(k)} \left(x_i^{(k)} - x_i^{(k-1)} \right), \quad i = 1, 2, \ldots, n,$$

pri čemu je $h_i^{(k)}$ parametar koraka, a $\beta_i^{(k)}$ je *masa* "teškog topa".

Pri pretraživanju maksimuma funkcije $Q(\mathbf{x})$ koristi se

$$x_i^{(k+1)} = x_i^{(k)} + h_i^{(k)} \frac{\partial Q^{(k)}}{\partial x_i} + \beta_i^{(k)} \left(x_i^{(k)} - x_i^{(k-1)} \right), \quad i = 1, 2, \ldots, n.$$

Masa se uzima iz uslova $0 \leq \beta_i^{(k)} \leq 1$ i predstavlja funkciju i-te koordinate gradijenta

$$\beta_i^{(k)} = f \left(\frac{\partial Q^{(k)}}{\partial x_i} \right).$$

Da bi "top" mogao da preskoči lokalni minimum, potrebno je funkciju f definisati tako da ispunjava sledeća dva uslova:

(i) $\beta_i^{(k)}$ je obrnuto proporcionalno vrednosti $\dfrac{\partial Q^{(k)}}{\partial x_i}$;

(ii) $\dfrac{\partial Q^{(k)}}{\partial x_i} \to 0 \Longrightarrow \beta_i^{(k)} \to 1.$

Različite modifikacije ovog metoda se razlikuju prema izboru funkcije f koja određuje "masu". Može se uzeti, na primer

$$\beta_i^{(k)} = \frac{1}{1 + \left| \dfrac{\partial Q^{(k)}}{\partial x_i} \right|}, \quad i = 1, \ldots, n.$$

Ovaj metod daje mogućnost da se preskoči neki od lokalnih ekstremuma, ali ne garantuje nalaženje globalnog ektremuma ako se kreće samo od jedne početne tačke. Algoritam bi trebalo da se ponovi više puta, polazeći od različitih početnih tačaka. Pored teškoća u formiranju masa $\beta_i^{(k)}$, ovaj algoritam ima sve nedostatke gradijentnih metoda.

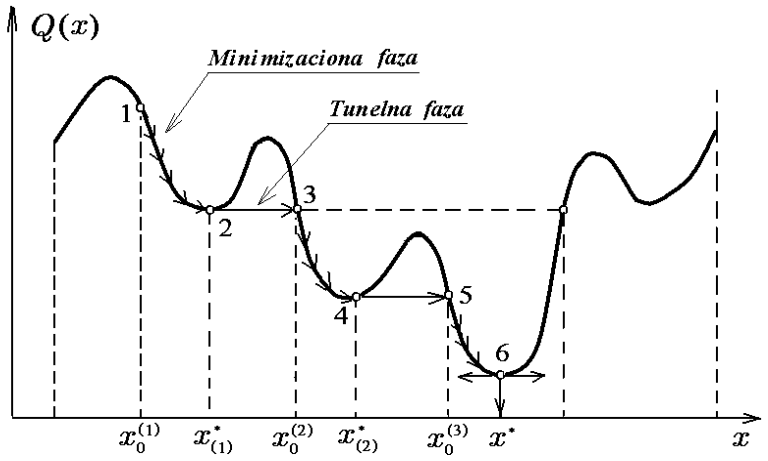
```
metod4[q_,var_List,h_List,epsilon_]:=
  Block[{p,rezultat,pom,qnaj,j=1,z,x0,x1,q0=q},
   Print["Zelite li min(1) ili max(2)?"];
   z=Input[];rezultat=Table[0,{2}];
   p=Table[0,{2}];
   n=Length[var];
   Do[Print["Unesite prvu pocetnu tacku(u obliku liste):  "];
      x0=Input[];
      Print ["Unesite drugu pocetnu tacku(u obliku liste):  "];
      x1=Input[];
      rezultat[[d]]=Last[teskitop[q0,x0,x1,var,h,epsilon,z]];
      p[[d]]=First[teskitop[q0,x0,x1,var,h,epsilon,z]],{d,2}
      ];
   If[z==1,qnaj=Min[p],qnaj=Max[p]];
      While[p[[j]]!=qnaj,j++];
   Return[{p[[j]],rezultat[[j]]}]
  ]

teskitop[q_,c0_List,c1_List,var_List,h_List,epsilon_,z_]:=
  Block[{q0=q,q1=q,q2,Q=q,x2,a,beta,x0=c0,x1=c1},
   n=Length[var];x2=Table[0,{n}];beta=Table[0,{n}];
   Do[q0=q0/.var[[i]]->x0[[i]],{i,n}];
   Do[q1=q1/.var[[i]]->x1[[i]],{i,n}];
   While[Abs[q1-q0]>epsilon,
      a=nabl[Q,var,x1];
      Do[beta[[i]]=N[1.0/(1.0+Abs[a[[i]]])],{i,n}];
      If[z==1,Do[x2[[i]]=x1[[i]]-h[[i]]*a[[i]]+beta[[i]]*
                         (x1[[i]]-x0[[i]]),{i,n}],
```

```
            Do[x2[[i]]=x1[[i]]+h[[i]]*a[[i]]+beta[[i]]*
                       (x1[[i]]-x0[[i]]),{i,n}]];
    q0=q1;q1=q; Do[q1=q1/.var[[i]]->x2[[i]],{i,n}];
    x0=x1; x1=x2;
 ];
 Return[{q1,x1}]
]
```

3.2.5. METOD TUNELA

Ovaj metod su razvili Levy, Montavlo i Gomez osamdesetih godina. Pripada grupi heurističkih metoda. Sastoji se iz dve faze: *minimizaciona* i *tunelna*. Grafička ilustracija ovog metoda, u slučaju jednog upravljačkog parametra, data na slici 3.2.2.



Sl. 3.2.2

Algoritam ovog metoda, u slučaju minimuma, može se opisati na sledeći način:

Korak 1. Izabrati početnu tačku $\mathbf{x}_0$ (tačka 1 sa grafika).

Korak 2. Koristeći neki od metoda za nalaženje lokalnog ekstremuma (gradijentni ili negradijentni) nalazi se minimum funckcije $Q(\mathbf{x})$, što predstavlja tekuće približavanje globalnom minimumu Q^* i odgovarajuće vrednosti x^* upravljačkog parametra (tačka 2 na grafiku). Ovaj korak predstavlja *fazu minimizacije*.

Korak 3. Formirati novu funkciju

$$T(\mathbf{x}) = \frac{(Q(\mathbf{x}) - Q^*)^2}{\sum_{i=1}^{n}(x_i - x_i^*)^2}. \tag{3.2.1}$$

Korak 4. Nekim od poznatih metoda optimizacije nalazi se minimum funkcije $T(\mathbf{x})$, tj. tačka $\mathbf{x}^{(T)}$ u kojoj je $T(\mathbf{x}^{(T)}) = 0$ ili $T(\mathbf{x}^{(T)}) \approx 0$. Ova faza se naziva *tunelna faza*.

Korak 5. Dobijena tačka $\mathbf{x}^{(T)}$ se uzima za novu početnu tačku $x_0 = \mathbf{x}^{(T)}$ i algoritam se ponavlja od *Koraka 2*.

Korak 6. Ove dve faze se ponavljaju opisanim redom, sve dok ne odredi tačka $\mathbf{x}^*$ (tačka 6 na grafiku) u kojoj je $T(\mathbf{x}^{(T)}) = 0$.

Za algoritam je veoma važno da se lokalni minimum funkcije $Q(\mathbf{x})$ što tačnije odredi. Obično se ovaj minimum određuje sa tačnošću ΔQ ciljne funkcije. Konvergencija algoritma se može poboljšati ako se tačnost ΔQ ugradi kao korigujući parametar u $T(\mathbf{x})$, tako da funkcija (3.2.1) dobija oblik

$$T(\mathbf{x}) = \frac{|\, Q(\mathbf{x}) - (Q^* - \Delta Q)\,|^2}{\sum_{i=1}^{n}(x_i - x_i^*)^2}. \tag{3.2.2}$$

Veliki problem u ovom algoritmu je određivanje kriterijuma za prekid.

Sledeći program je implementacija prethodnog algoritma u paketu MATHEMATICA:

```
Metod5[q1_,var_List,xmin_List,xmax_List,epsilon_Real,yy0_List]:=
Block[{T1,q,Q=q1,o,P,P1,n,h,hmin,elha,f,k,T,c=var,broj,y0=yy0},
   Print["Unesite broj iteracija"];
   k=Input[]; n=Length[var]; o=Table[0,{2}];
   Do[q=q1;
      Print["Izaberite jedan od metoda"];
      Print["1-SLUCAJNI SMEROVI"];
      Print["2-SLUCAJNO PRETRAZIVANJE SA OBRNUTIM KORAKOM"];
      Print["3-SLUCAJNO TRAZENJE SA UKAZANOM SLUCAJNOSCU"];
      Print[];Print["Unesite broj metoda"];
   broj=Input[];
   h=Input["Parametar koraka u obliku liste"];
```

```
elha=Input["Parametar kojim se koraci skracuju"];
hmin=Input["Tacnost lokalizacije ekstrema u obliku liste"];
Which[broj==1,
        o=First[SluDir[q,var,y0,h,hmin, elha,xmin,xmax]],
 broj==2,o=First[SluOb[q,var,y0,h,hmin, elha,xmin,xmax]],
 broj==3,o=First[SluCon[q,var,y0,h,hmin, elha,xmin,xmax]]
];
P=q1;
Do[P=P/.var[[j]]->o[[j]],{j,n}];
T=Abs[(Q-(P-0.001))^2]/Sum[(c[[i1]]-(o[[i1]]-
hmin[[i1]]))^2,{i1,n}];
T1=T;
h=Input["Unesite parametar koraka u obliku liste
 (razlicit od param.  koraka za f-ju q)"];
elha=Input["Parametar kojim se koraci skracuju"];
hmin=Input["Tacnost lokalizacije ekstrema(u obliku liste)"];
Print["PAZNJA!:birajte min(2)"];
Which[broj==1,
        o=First[SluDir[T,var,y0,h,hmin, elha,xmin,xmax]],
 broj==2,o=First[SluOb[T,var,y0,h,hmin, elha,xmin,xmax]],
 broj==3,o=First[SluCon[T,var,y0,h,hmin, elha,xmin,xmax]]
 ];
y0=o; P1=T1;
Do[P1=P1/.var[[j]]->y0[[j]],{j,n}]; q=q1;
Do[q=q/.var[[j]]->y0[[j]],{j,n}];
If[P1>epsilon,
        Return[{Print["T:",{o,P1},," q:", {o,q}]}]],{f,k}
 ];
Return[{o,q}]
]
```

III GLAVA

Uslovna optimizacija

U ovoj glavi se izučavaju problemi implementacije metoda uslovne optimizacije u kojima se izračunava minimum ili maksimum ciljne funkcije $Q(\mathbf{x}) = Q(x_1, \dots, x_n), \quad \mathbf{x} \in D_x$, na nekom skupu koji je određen uslovima koji su definisani drugim funkcijama:

$$\begin{aligned} &\varphi_\ell(x_1, \dots, x_n) = \varphi_{0_\ell}, \quad \ell = 1, \dots, n_1 < n, \\ &\Psi_j(x_1, \dots, x_n) \geq \Psi_{0_j}, \quad j = 1, \dots, m_2. \end{aligned} \tag{1.1.1}$$

Prvi uslovi se nazivaju funkcionalna ograničenja tipa jednakosti, dok se drugi uslovi nazivaju ograničenja tipa nejednakosti, ili ograničenja definisana hiperpovršima. Zadatak optimizacije koji uključuje ograničenja svih tipova naziva se *opšti zadatak optimizacije.*

1. O SIMBOLIČKOJ IMPLEMENTACIJI

1.1. Postojeći programi

U literaturi su poznati programi za implementaciju metoda uslovne optimizacije, koji su napisani u proceduralnim programskim jezicima, prvenstveno u FORTRANu ([2], [13], [19], [70]) i C jeziku ([24], [71]). Međutim, proceduralni programski jezici nisu pogodni za implementaciju uslovnih optimizacionih metoda, što će biti izučavano u ovom poglavlju. U radu [52] izučavana je implementacija metoda kaznenih funkcija pomoću metoda simboličkog procesiranja koji su dostupni u programkim jezicima LISP i MATHEMATICA. U ovoj glavi biće implementiran i detaljnije objašnjen veći broj metoda za uslovnu optimizaciju.

Takođe, može se primetiti nedostatak ugrađenih funkcija u jeziku MATHEMATICA za implementaciju metoda uslovne optimizacije. U programskom paketu MATHEMATICA dostupno je nekoliko funkcija za uslovnu numeričku optimizaciju. Funkcije *ConstrainedMin* i *ConstrainedMax* dozvoljavaju da se specificira linearna funkcija cilja koja se minimizira ili maksimizira, zajedno sa skupom linearnih ograničenja tipa nejednakosti. U svim slučajevima se pretpostavlja da promenljive mogu uzimati samo nenegativne vrednosti.

Može se zaključiti da je jedino linearno programiranje implementirano u paketu MATHEMATICA. Sintaksa ovih funkcija se može opisati na sledeći način.

`ConstrainedMin[f, {inequalities}, {x, y, ...}]` nalaženje minimuma funkcije f, u regionu koji je specificiran sa *inequalities*.

`ConstrainedMax[f, {inequalities}, {x, y, ...}]` nalaženje maksimuma funkcije f, u regionu koji je specificiran sa *inequalities*.

Očigledno da ugrađene funkcije nisu dovoljne za veliki broj raznovrsnih metoda uslovne optimizacije.

1.2. Osnovne prednosti

U ovoj glavi se izučava implementacija metoda uslovne optimizacije, koristeći simboličko procesiranje u funkcionalnim programskim jezicima MATHEMATICA i LISP. Praktično, učinjen je pokušaj (prvi ovakve vrste, koliko je autorima poznato), da se ujedine mogućnosti simboličkog i numeričkog procesiranja u implementaciji metoda uslovne optimizacije. Jasno je da se u toku simboličke implementacije metoda uslovne optimizacije podrazumevaju sve prednosti koje proizilaze iz simboličke implementacije metoda bezuslovne optimizacije.

Posmatra se sledeći opšti nelinearni problem matematičkog programiranja:

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \Gamma_{\mathbf{x}} \subseteq \mathbb{R}^n \\ &\text{P.O.: } f_i(\mathbf{x}) \leq 0, \quad i \in \mathcal{P} = \{1, \ldots, p\} \\ &\qquad h_j(\mathbf{x}) = 0, \quad j \in \mathcal{Q} = \{1, \ldots, q\}. \end{aligned} \tag{1.2.1}$$

Mogu se odmah napomenuti sledeće prednosti koje su zajedničke za simboličku implementaciju svih metoda uslovne optimizacije.

(1C) Mogućnost da implementacione procedure sadrže ciljnu funkciju i data ograničenja u listi formalnih parametara.

(2C) Mogućnost da se generiše vektor ili lista čiji su elementi izabrane funkcije. U programskom jeziku MATHEMATICA, funkcije koje su zadate kao elementi vektora mogu da se primenjuju na listu argumenata. Takođe, u programskom jeziku LISP, funkcije kao elementi vektora mogu kasnije da se transformišu u odgovarajuće lambda-izraze i da budu primenjene na zadatu listu argumenata.

Za proceduralne programske jezike je nepodesan problem de se proizvoljne ciljne funkcije i ograničenja ugrade u listu parametara procedure kojom se implementira optimizacioni problem. Obično je ciljna funkcija definisana potprogramom, a ograničenja su definisana potprogramima i ugrađena u

niz ([36], [37], [60]). Prema tome, primena nove ciljne funkcije i novih ograničenja je uslovljena ovim definicijama. Na primer, u [36] je svako ograničenje tipa jednakosti, svako ograničenje tipa nejednakosti i ciljna funkcija identifikovano nekom indeksiranom promenljivom – $R[i]$. Na taj način, problem (1.2.1) je zapamćen na sledeći način:

$$\begin{aligned} R(1) = f_1(\mathbf{x}), \dots, R(p) = f_p(\mathbf{x}), \\ R(p+1) = h_1(\mathbf{x}), \dots, R(p+q) = h_q(\mathbf{x}), \\ R(p+q+1) = Q(\mathbf{x}). \end{aligned}$$

Svaka primena uslovne optimizacione procedure zahteva modifikaciju elemenata niza R, tj. odgovarajuću intervenciju u kodu. Osim toga, dimenzija problema je limitirana dimenzijom niza R.

Primer 1.2.1. Uslovni program

$$\begin{aligned} &\text{Minimizirati: } -x_1 - x_2 \\ &\text{P.O.: } f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 \leq 0, \\ &\qquad h_1(x_1, x_2) = -x_1 + x_2^2 = 0 \end{aligned}$$

je reprezentovan na sledeći način:

$$\begin{aligned} R(1) &= X(1) * *2 + X(2) * *2 - 1, \\ R(2) &= -X(1) + X(2) * *2, \\ R(3) &= -X(1) - X(2). \end{aligned}$$

Primetimo da je definicija ciljne funkcije, kao i ograničenja, striktno vezana za vrednosti globalnog niza X.

Kao efikasniju alternativu, opisaćemo algoritam koji omogućuje korišćenje ciljne funkcije i ograničenja u listi formalnih parametara, u programskim jezicima MATHEMATICA i LISP. U tu svrhu je uvedena *unutrašnja forma* podesna za nelinearne uslovne probleme. Nelinearni uslovni problem (1.2.1) se transformiše u sledeću LISPovsku unutrašnju formu:

’(Q(x) (x) ($f_1(x)$ $\cdots$ $f_p(x)$) ($h_1(x)$ $\cdots$ $h_q(x)$))

Analogna unutrašnja reprezentacija u MATHEMATICA je zadata sledećim izrazima:

Q(x), {x}, { $f_1(x)$ $\cdots$ $f_p(x)$ }, {$h_1(x)$ $\cdots$ $h_q(x)$ }

Ako neki od uslova tipa jednakosti ili nejednakosti odsustvuje, odgovarajuća lista je prazna.

Primer 1.2.2. Nelinearni uslovni optimizacioni problem

$$\begin{aligned} &\text{Minimizirati: } -x_1 - x_2 \\ &\text{P.O.: } x_1^2 + x_2^2 - 1 = 0 \end{aligned}$$

reprezentovan je u sledećoj unutrašnjoj formi u LISPu:

```
' (  (- 0 (+ x1 x2))  (x1 x2)
     ()
     (  (- (+ (* x1 x1) (* x2 x2)) 1) )
)
```

Odgovarajuća unutrašnja forma u MATHEMATICA je

```
-x1-x2, {x1,x2},
     {},
     {x1^2+x2^2-1}
```

Prvi element *unutrašnje forme* je proizvoljna aritmetička funkcija u MATHEMATICA ili SCHEME, drugi predstavlja listu argumenata te funkcije, treći element je lista funkcija koje formiraju ograničenja tipa nejednakosti, dok se četvrti argument interpretira kao lista funkcija koja je sadržana u ograničenjima tipa jednakosti. Prema tome, *funkcija* sadržana u LISPovskoj unutrašnjoj formi q proizvoljnog nelinearnog uslovnog problema optimizacije, može da se selektuje izrazom $(car\ \ q)$, a odgovarajuća *lista parametara* izrazom $(cadr\ \ q)$. Slično, lista funkcija koje formiraju ograničenja tipa jednakosti može da se izdvoji pomoću izraza $(caddr\ \ q)$, a lista funkcija koje formiraju ograničenja tipa nejednakosti koristeći $(cadddr\ \ q)$. Napomenimo da korisnik mora da postavi uslovni optimizacioni zadatak u opisanoj unutrašnjoj formi. Motivacija za izbor takve unutrašnje forme jeste slična unutrašnja forma za linearne uslovne optimizacione probleme u MATHEMATICA (vidi funkciju *ConstrainedMin*.)

Na taj način je opisana prednost **(1C)**.

Vektor vf koji sadrži funkcije sadržane u ograničenjima tipa nejednakosti u unutrašnjoj formi q, koja je primenljiva u SCHEME, može da se konstruiše kako sledi:

```
(set!  vf (make-vector (set!  lf (length (caddr q)))))
(if (< 0 lf) (set!  vf (list->vector (caddr q))))
```

Na sličan način se može generisati vektor vh koji sadrži funkcije iz ograničenja tipa jednakosti:

```
(set!  vh (make-vector (set!  lh (length (cadddr q)))))
(if (< 0 lh) (set!  vh (list->vector (caddr q))))
```

Lista ograničenja tipa jednakosti i nejednakosti u MATHEMATICA sadržana je u trećem i četvrtom delu unutrašnje forme zadatog uslovnog problema, respektivno. U ovim izrazima se koristi sposobnost funkcionalnih programskih jezika da plasiraju proizvoljnu selektovanu funkciju u listu ili vektor, što je

nepodesan problem za proceduralne programske jezike. To je deo prednosti **(2C)**.

2. OGRANIČENJA DATA JEDNAKOSTIMA

2.1. Uvod

Posmatraćemo ciljnu funkciju $Q(\mathbf{x})$, čiji upravljački parametri ispunjavaju uslov

$$\mathbf{x} \in \Gamma_{\mathbf{x}}$$

pri čemu su nametnuta funkcionalna ograničenja zadata jednakostima

$$\varphi_j(x_1, \dots, x_n) = \varphi_{0_j}, \quad j = 1, \dots, n_1,$$

tj.

$$\psi_j(x_1, \dots, x_n) = \varphi_j(x_1, \dots, x_n) - \varphi_{0_j} = 0, \quad j = 1, \dots, n_1. \tag{2.1.1}$$

Svaki vektor $\mathbf{x}$ koji ispunjava ograničenja (2.1.1) naziva se *dopustivo rešenje*. Zadatak uslovne optimizacije je da se izračuna vektor $\mathbf{x}^*$ za koji je $Q(\mathbf{x}^*) = Q_{\min}$ i koji ispunjava zadata ograničenja. Rešenje $\mathbf{x}^*$ se naziva *minimalno rešenje* postavljenog problema ako važi

$$Q(\mathbf{x}^*) \leq Q(\mathbf{x}) \tag{2.1.2}$$

za svako dopustivo rešenje $\mathbf{x}$. Ako je

$$Q(\mathbf{x}^*) \geq Q(\mathbf{x}) \tag{2.1.3}$$

za svako dopustivo rešenje $\mathbf{x}$, tada je $\mathbf{x}^*$ *maksimalno rešenje* postavljenog problema. Ako (2.1.2), odnosno (2.1.3) važi za svako dopustivo rešenje u nekoj okolini $N(\mathbf{x}^*) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{x}^*\| \leq \varepsilon\}$ tačke $\mathbf{x}^*$, tada je $\mathbf{x}^*$ *lokalno minimalno*, odnosno *lokalno maksimalno* rešenje postavljenog problema. Rešenje $\mathbf{x}^*$ se naziva *uslovni ekstremum* funkcije $Q(\mathbf{x})$.

Neophodan uslov za postojanje rešenja ovog zadatka jeste $n_1 < n$. Zaista, u slučaju $n_1 = n$, zadatak se svodi na rešavanje sistema jednačina (2.1.1) i izračunavanje vrednosti ciljne funkcije u vektoru koji je dobijen kao rešenje tog sistema. U nekim slučajevima, tako postavljeni problemi su nekorektni, i nemaju rešenja. Za rešavanje ovakvih problema razvijeno je više metoda: metod eliminacije promenljivih, metod Lagrangeovih množitelja, projekcija na gradijentni vektor, grafičko rešenje za slučaj $n = 2$, itd.

2.2. Metodi eliminacije promenljivih

Problem optimizacije sa zadatim ograničenjima tipa (2.1.1) se može rešiti tako što se iz zadatih ograničenja eliminiše n_1 promenljivih. Dobijene vrednosti upravljačkih parametara se zamenjuju u analitički izraz funkcije $Q(\mathbf{x}) = Q(x_1, \ldots, x_n)$, čime se problem svodi na optimizacioni zadatak funkcije od $n - n_1$ upravljačkog parametara, i to bez ograničenja [60].

2.3. Metodi Lagrangeovih množitelja

Za rešavanje optimizacionih problema sa ograničenjima tipa jednakosti najčešće je korišćen *Metod Lagrangeovih množitelja.* Ideju da se problem sa ograničenjima tipa jednakosti svede na problem bez ograničenja koristio je Lagrange još 1788. godine. Metod se zasniva na sledećem poznatom, potrebnom uslovu optimalnosti.

Teorema 2.3.1. *Neka su funkcije Q i ψ_j, $j = 1, \ldots, n_1$ $(n_1 < n)$, definisane u $\mathbb{R}^n$ i neprekidno diferencijabilne u nekoj okolini $N(\mathbf{x}^*)$ dopustivog rešenja problema sa ograničenjima* (2.1.1) *tipa jednakosti. Pretpostavimo da je rang Jacobieve matrice*

$$\begin{bmatrix} \nabla\psi_1(\mathbf{x}^*) \\ \vdots \\ \nabla\psi_{n_1}(\mathbf{x}^*) \end{bmatrix}$$

jednak n_1. Ako je $\mathbf{x}^*$ lokalno minimalno ili lokalno maksimalno rešenje problema sa ograničenjima tipa jednakosti, tada se gradijent funkcije cilja Q u tački $\mathbf{x}^*$ može izraziti kao linearna kombinacija gradijenata funkcija ograničenja ψ_k, $k = 1, \ldots, n_1$, u tački $\mathbf{x}^*$, tj. postoje realni brojevi $\lambda_1^*, \ldots, \lambda_{n_1}^*$ takvi da je

$$\nabla Q(\mathbf{x}^*) + \sum_{k=1}^{n_1} \lambda_k^* \nabla\psi_k(\mathbf{x}^*) = 0\,. \tag{2.3.1}$$

Činjenica da lokalni optimum $\mathbf{x}^*$ ciljne funkcije Q mora da zadovolji ograničenja tipa (2.1.1) i jednačinu (2.3.1) izražava se pomoću *Lagrangeove funkcije* (ili *Lagrangiana*), koja se definiše na sledeći način:

$$L(\mathbf{x}, \lambda) = Q(\mathbf{x}) + \sum_{k=1}^{n_1} \lambda_k \psi_k(\mathbf{x})\,. \tag{2.3.2}$$

Očigledno je $L(\mathbf{x}, \lambda)$ funkcija od n promenljivih $\mathbf{x} = (x_1, \ldots, x_n)$ i n_1 promenljivih $\lambda = (\lambda_1, \ldots, \lambda_{n_1})$.

Neka su Lagrangeovi množitelji označeni sa

$$\lambda_i = x_{n+i}, \quad i = 1, \dots, n_1.$$

U ovom slučaju, funkcionalni pristup se može upotrebiti za efikasnu manipulaciju sa *izračunljivim* (*slack*) promenljivima, što je izraženo sledećim prednostima simboličke implementacije:

(3C) Jednostavna ekstenzija *liste parametara* ciljne funkcije listom izračunljivih promenljivih. Tako uvedene izračunljive promenljive se mogu koristiti kao tipovi podataka prve vrste.

(4C) Simbolički generisane izračunljive promenljive i težinski koeficijenti mogu se jednostavno ugraditi u unutrašnju formu *transformisane funkcije*.

Lagrangeovi množitelji se u paketu MATHEMATICA mogu označiti pomoću indeksiranih promenljivih $\lambda_i = x[n+i]$, $i = 1, \dots, n_1$. Tada se Lagrangeova funkcija $L(\mathbf{x}, \lambda)$ može formirati sledećom funkcijom:

```
prevedi[jednacine_List,pr_List,np_]:=
   Block[{jed=jednacine,i},
    Do[jed=jed/.pr[[i]]->x[i],{i,Length[pr]}];
    For[i=2,i<np+2,i++, jed[[i]] *= x[n+i-1] ];
    Return[jed] ]
```

Na osnovu Teoreme 2.3.1, potreban uslov za postojanje lokalnog optimuma se može izraziti na sledeći način:

Teorema 2.3.2. *Ako je dopustivo rešenje* $\mathbf{x}^*$ *lokalni optimum problema sa ograničenjima tipa* (1.1.1), *tada postoji vektor* $\lambda^* = (\lambda_1, \dots, \lambda_{n_1})$ *takav da je*

$$\nabla L(\mathbf{x}^*, \lambda^*) = 0. \tag{2.3.3}$$

Ako se parcijalni izvodi funkcije L *posmatraju u odnosu na promenljive* $x_1, \dots, x_n$, *poslednja jednakost postaje*

$$\frac{\partial Q(\mathbf{x}^*)}{\partial x_i} + \sum_{k=1}^{n_1} \lambda_k \frac{\partial \psi_k(\mathbf{x}^*)}{\partial x_i} = 0, \quad i = 1, \dots, n. \tag{2.3.4}$$

Metod *Lagrangeovih množitelja* se sastoji u izračunavanju vektora $\mathbf{x}^*$ i λ^* pomoću (2.3.3) i (2.3.4). Napomenimo da se parcijalni izvodi u (2.3.3) mogu uzeti po svakoj od $n+n_1$ promenljivih $x_1, \dots, x_n, \lambda_1, \dots, \lambda_{n_1}$. U tom slučaju, izvodi po x_k daju (2.3.1), dok izvodi po λ_k daju (2.1.1).

Ako se za Lagrangeove množitelje upotrebe oznake

$$\lambda_j = x_{n+j}, \quad j = 1, \dots, n_1,$$

tada se sistem jednačina (2.1.1), (2.3.4) može zapisati u obliku

$$F_l(\mathbf{x}) = F_l(x_1, \dots, x_{n+n_1}) = 0, \quad l = 1, \dots, n+n_1. \tag{2.3.5}$$

Svi izrazi oblika

$$\frac{\partial Q(\mathbf{x})}{\partial x_i}+\sum_{k=1}^{n_1}\lambda_k\frac{\partial \psi_k(\mathbf{x})}{\partial x_i}=\frac{\partial}{\partial x_i}\left(Q(\mathbf{x})+\sum_{k=1}^{n_1}\lambda_k\psi_k(\mathbf{x})\right),\quad i=1,\dots,n,$$

mogu da se ugrade u listu sledećom funkcijom *flista*, koja se primenjuje na rezultat funkcije *prevedi*:

```
flista[jed_List]:=
   Block[{i,j,p,dqdx={}},
     For[i=1,i<=n+n1,i++,
       p=D[jed[[1]],x[i]];
       For[j=2,j<=n1+1,j++, p+=D[jed[[j]],x[i]] ];
       dqdx=Append[dqdx,p]
     ];
     Return[dqdx] ]
```

Primer 2.3.1. Traži se maksimum funkcije $Q(x_1,x_2,x_3)=Q(\mathbf{x})$, pri zadatim ograničenjima

$$\varphi_1(x_1,x_2,x_3)=\varphi_{01},$$
$$\varphi_2(x_1,x_2,x_3)=\varphi_{02}.$$

Vektor $\mathbf{x}^*=(x_1^*,x_2^*,x_3^*)$ u kojoj je vrednost funkcije maksimalna, kao i parametri λ_1 i λ_2, definišu se kao rešenja sledećeg sistema jednačina:

$$\begin{aligned}
\frac{\partial Q}{\partial x_1}+\lambda_1\frac{\partial \psi_1}{\partial x_1}+\lambda_2\frac{\partial \psi_2}{\partial x_1}&=0,\\
\frac{\partial Q}{\partial x_2}+\lambda_1\frac{\partial \psi_1}{\partial x_2}+\lambda_2\frac{\partial \psi_2}{\partial x_2}&=0,\\
\frac{\partial Q}{\partial x_3}+\lambda_1\frac{\partial \psi_1}{\partial x_3}+\lambda_2\frac{\partial \psi_2}{\partial x_3}&=0,\\
\varphi_1(x_1,x_2,x_3)-\varphi_{01}&=0,\\
\varphi_2(x_1,x_2,x_3)-\varphi_{02}&=0.
\end{aligned}$$

Ne treba izgubiti iz vida činjenicu da su postavljeni uslovi izraženi pomoću sistema Lagrangeovih jednačina samo potrebni, tako da je dobijeno rešenje samo kandidat za lokalni minimum. Za proveru da li je $\mathbf{x}^*$ zaista lokalni minimum, trebalo bi koristiti neki od dovoljnih uslova za optimum. Jedan od najčešćih uslova se može formulisati na sledeći način:

Teorema 2.3.3. *Neka su Q i ψ_k, $k=1,\dots,n_1$, dvaput neprekidno diferencijabilne funkcije. Ako postoji vekor λ^* takav da je*

$$\nabla L(\mathbf{x}^*,\lambda^*)=0$$

i ako za svako $\mathbf{z} \neq \mathbf{0}$ *koje zadovoljava*

$$\nabla \psi_k(\mathbf{x}^*) z = 0, \quad k = 1, \dots, n_1,$$

važi

$$\langle z, \nabla_x^2 L(\mathbf{x}^*, \lambda^*) z \rangle > 0,$$

tada je $\mathbf{x}^*$ *izolovano lokalno minimalno rešenje problema sa ograničenjima tipa* (2.1.1). *Ako se znak* $>$ *u poslednjoj nejednakosti zameni sa* $<$, *tada je* $\mathbf{x}^*$ *izolovano lokalno maksimalno rešenje.*

Simbol $\nabla_x^2 L(\mathbf{x}^*, \lambda^*)$ označava drugi parcijalni izvod funkcije $L(\mathbf{x}, \lambda)$, uzet u tački $(\mathbf{x}^*, \lambda^*)$, ali samo u odnosu na vektor $\mathbf{x}$. Ovaj dovoljan uslov može se izraziti i na sledeći način: Potrebno je da važi uslov (2.3.3) i da kvadratna forma matrice $\nabla_x^2 L(\mathbf{x}^*, \lambda^*)$ bude pozitivno definitna na nula-prostoru Jacobieve matrice ograničenja (osim u koordinatnom početku).

Sistem nelinearnih jednačina (2.3.5) može se rešiti na više različitih načina, na primer korišćenjem *Newtonovog* metoda, *minimizacionog* metoda, itd.

2.3.1. NEWTONOV METOD

Ako se skup funkcija $(F_1, \dots, F_{n+n_1})$ predstavi kao vektor-funkcija $\mathbb{F}$, tada se sistem (2.3.5) može zapisati u sledećoj vektorskoj formi:

$$\mathbb{F}(\mathbf{x}) = \mathbf{0}. \tag{2.3.5a}$$

Ovaj sistem može se rešiti različitim iterativnim postupcima. Neka je k-ta aproksimacija rešenja sistema (2.3.5a) data sa $\mathbf{x}^{(k)} = (x_1^{(k)}, \dots, x_{n+n_1}^{(k)})$, a njena greška $\varepsilon^{(k)}$. Tačno rešenje $\mathbf{x}^*$ je

$$\mathbf{x}^* = \mathbf{x}^{(k)} + \varepsilon^{(k)}. \tag{2.3.6}$$

Zamenom (2.3.6) u (2.3.5a) dobija se

$$\mathbb{F}(\mathbf{x}^{(k)} + \varepsilon^{(k)}) = \mathbf{0}. \tag{2.3.7}$$

Neka je vektor-funkcija $\mathbb{F}(\mathbf{x})$ neprekidna i diferencijabilna u oblasti koja sadrži $\mathbf{x}^*$ i $\mathbf{x}^{(k)}$ i posmatrajmo njen razvoj do linearnog člana:

$$\mathbb{F}(\mathbf{x}^{(k)} + \varepsilon^{(k)}) \approx \mathbb{F}(\mathbf{x}^{(k)}) + \nabla \mathbb{F}(\mathbf{x}^{(k)}) \varepsilon^{(k)} = \mathbf{0}. \tag{2.3.8}$$

U poslednjoj jednakosti $\nabla \mathbb{F}(\mathbf{x}^{(k)})$ predstavlja Jacobievu matricu, koja se obrazuje od parcijalnih izvoda funkcija $F_1, \dots, F_N$ u odnosu na promenljive $x_1, \dots, x_N$,

$$\nabla \mathbb{F}(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial F_1}{\partial x_1} & \cdots & \dfrac{\partial F_1}{\partial x_N} \\ \vdots & & \\ \dfrac{\partial F_N}{\partial x_1} & & \dfrac{\partial F_N}{\partial x_N} \end{bmatrix}, \tag{2.3.9}$$

gde je $N = n + n_1$. Skraćeno koristimo oznaku

$$\nabla\mathbb{F}(\mathbf{x}) = \left\{ \frac{\partial F_i}{\partial x_j} \right\}, \quad i, j = 1, \ldots, N.$$

Pod uslovom da je matrica $\nabla\mathbb{F}(\mathbf{x})$ nesingularna, greška $\varepsilon^{(k)}$ se može odrediti u obliku

$$\varepsilon^{(k)} = -\nabla\mathbb{F}^{-1}(\mathbf{x}^{(k)})\mathbb{F}(\mathbf{x}^{(k)}).$$

Na taj način, dobijamo sledeću iterativnu formulu za nalaženje rešenja jednačine (2.3.5):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \nabla\mathbb{F}^{-1}(\mathbf{x}^{(k)})\mathbb{F}(\mathbf{x}^{(k)}). \tag{2.3.10}$$

Neka je rezultat funkcije *flista* označen listom *novejednacine*. Tada se matrica $\nabla\mathbb{F}(\mathbf{x})$, definisana u (2.3.9), može formirati sledećom funkcijom:

```
nablaFunction[novejednacine_List]:=
   Block[{i,j,nablaF={},nov=novejednacine},
     Do[dqdx={};
        Do[dqdx=Append[dqdx,D[nov[[i]],x[j]]],{j,n+n1}];
        nablaF=Append[nablaF,dqdx],{i,n1+n}
     ];
     Return[nablaF] ]
```

Glavni nedostatak ovog metoda je izbor početne iteracije $\mathbf{x}^{(0)}$. Vektor $\mathbf{x}^{(0)}$ se odabira na slučajan način ili na osnovu nekih apriornih podataka za oblast u kome se nalazi rešenje. Drugi nedostatak ovog metoda je neophodnost da se u svakoj iteraciji izračunava inverzna matrica Jacobieve matrice. Ovaj nedostatak se otklanja tako što se $\nabla\mathbb{F}^{-1}(\mathbf{x})$ izračunava samo u početnoj tački, što dovodi do iterativnog procesa

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \nabla\mathbb{F}^{-1}(\mathbf{x}^{(0)})\mathbb{F}(\mathbf{x}^{(k)}). \tag{2.3.11}$$

Ponekad se koristi još jedna modifikacija Newtonovog metoda u kojoj je neophodno je da se u svakoj iteraciji prati opadanje (rašćenje) funkcije cilja. Dok se vrednosti vektor-funkcije $\mathbb{F}(\mathbf{x})$ poboljšavaju, koriste se iteracije oblika (2.3.11). Kada vrednosti $\mathbb{F}(\mathbf{x}^{(k)})$ počnu da se pogoršavaju, potrebno je da se ponovo izračuna inverzna Jacobieva matrica $\nabla\mathbb{F}^{-1}(\mathbf{x}^{(k)})$, a zatim da se ponovo nastavi sa postupkom (2.3.10). Ovakva ideja podseća na modifikaciju osnovnog gradijentnog metoda koji se primenjuje u gradijentnoj bezuslovnoj optimizaciji. U oblasti ekstremuma može da se potpuno pređe na osnovni Newtonov metod (2.3.10). Sada ćemo dati implementaciju osnovnog Newtonovog metoda. Funkcijom *sjV* izračunava se vrednost liste *jednacine* u datoj tački $x0$.

```
sjV[jednacine_List,x0_List,pr_List,np_]:=
   Block[{sve=jednacine,i},
      Do[sve=sve/.x[i]->x0[[i]],{i,Length[pr]+np}];
      N[sve] ]
```

Osnovni Newtonov metod je implementiran u funkciji *iteracija*. Na osnovu opisane prednosti **(1C)**, unutrašnja forma uslovnog optimizacionog problema sadržana je u parametrima *q_*, *prom_* i *uslovi_*, koji sadrže redom analitički izraz ciljne funkcije, listu parametara te funkcije i listu sa ograničenjima tipa jednakosti. Formalni parametar $x0$ predstavlja vrednost polazne aproksimacije $\mathbf{x}^{(0)}$.

```
iteracija[q_,prom_,uslovi_List,x0_List]:=
   Block[{nabla,duz,n,n1,x1,q1,p,jednacine,f},
      n=Length[prom];n1=Length[uslovi];   duz=Length[x0];
      If[duz!=n1+n,Return["Nije dobra lista brojeva!"]];
      jednacine=Prepend[uslovi,q];
      prevod=prevedi[jednacine,prom,n1];
      f=flista[prevod]; nabla=nablaFunction[f];
      If[Det[nabla]==0,Return["Singularna Matrica"]];
      nablaInverzna=Inverse[nabla];
      For[i=1;x1=x0,i<30,i++, x1=x1-
         sjV[nablaInverzna,x1,prom,n1].sjV[f,x1,prom,n1]
      ];
      q1=q;  Do[q1=q1/.prom[[i]]->x1[[i]],i,n]; q1=N[q1];
      {x1,q1} ]
```

Rezultati testiranja programa.

```
In[1]:= iteracija[x^2+y^2,{x,y},{x-y-3},{1,1,1.}]
 prom = {x, y}
jednacine = {x + y , -3 + x - y}
                2       2
prevod = {x[1] + x[2] , (-3 + x[1] - x[2]) x[3]}
flista = {2 x[1] + x[3], 2 x[2] - x[3], -3 + x[1] - x[2]}
nabla = {{2, 0, 1}, {0, 2, -1}, {1, -1, 0}}
Out[1]= {{1.5, -1.5, -3.},4.5}
```

2.3.2. MINIMIZACIONI METODI

Sistem ograničenja (2.3.5) može da se reši bilo kojom metodom za nalaženje minimuma za funkcije više promenljivih (Newtonov metod, *DFP* metod, Cauchyev metod najstrmijeg pada, metodi za nalaženje globalnog ekstremuma, itd.).

Uočimo najpre funkciju cilja u obliku

$$\Psi(x_1,\dots,x_N)=\sum_{l=1}^{N}\left[F_l(x_1,\dots,x_N)\right]^2. \tag{2.3.12}$$

Iz tog oblika može se zaključiti da se minimalna vrednost funkcije cilja (koja je jednaka nuli) dostiže kada je $F_l(\mathbf{x})=0$, tj. ako su zadovoljene jednačine (2.3.5).

Za ovaj problem bitno je da je minimum ciljne funkcije $\Psi(x_1,\dots,x_N)$ jednak nuli. Zbog toga se za kriterijum kraja optimizacije može iskoristiti uslov

$$\Psi(\mathbf{x}^*)\le\varepsilon,$$

gde je ε mali pozitivan broj.

Da bi se sastavila funkcija cilja (2.3.12) jasno je da mora biti poznat analitički izraz funkcija u (2.3.5). To je moguće da se uradi ako ciljna funkcija $Q(\mathbf{x})$ i ograničenja (2.1.1) imaju relativno jednostavne parcijalne izvode, te se može odrediti analitčki izraz funkcija u sistemu (2.3.5). Međutim, ako su parcijalni izvodi funkcija u (2.3.5) komplikovani, mogu se koristiti numerički parcijalni izvodi ciljne funkcije i funkcija $\psi_j, j=1,\dots,n_1$. U tom slučaju, u svakoj iteraciji, za nalaženje uslovnog ekstremuma $\mathbf{x}^*$ formira se funkcija

$$\Psi(\mathbf{x})=\sum_{l=1}^{N}\left(\frac{\partial Q}{\partial x_i}+\sum_{j=1}^{n_1}\lambda_j\frac{\partial\psi_j}{\partial x_i}\right)^2 + \sum_{j=1}^{n_1}\left[\psi_j(x_1,\dots,x_n)-\psi_{0j}\right]^2. \tag{2.3.13}$$

Tada se u prvoj sumi u (2.3.13) komponente gradijenta izračunavaju numerički. Kao što je napomenuto, numeričko izračunavanje gradijenta nije predmet razmatranja.

3. OPŠTI ZADATAK OPTIMIZACIJE

3.1. Uvod

Sada se rešava optimizacioni zadatak za funkciju $Q(\mathbf{x})$, čiji upravljački parametri ispunjavaju uslov $\mathbf{x}\in\Gamma_{\mathbf{x}}$, kao i funkcionalna ograničenja tipa

nejednakosti, koja mogu biti sledećeg oblika:

$$
\begin{aligned}
\Psi_1(x_1,\dots,x_n) &\geq \Psi_{01},\\
\Psi_2(x_1,\dots,x_n) &\leq \Psi_{02},\\
\Psi'_{03} \leq \Psi_3(x_1,\dots,x_n) &\leq \Psi_{02} \leq \Psi''_{03},\\
&\vdots\\
\Psi_{m_2}(x_1,\dots,x_n) &\geq \Psi_{0m_2}.
\end{aligned}
\tag{3.1.1}
$$

Za rešavanje optimizacionih zadataka sa proizvoljnim ograničenjima razrađen je veći broj metoda: grafičko rešenje, za slučaj $n = 2$, skaniranje u dozvoljenoj oblasti, slučajno pretraživanje, metod "kaznenih" funkcija, gradijentni metodi, kompleks metodi, itd. Zajedničko za sve ove metode je da skoro svi slede ideju konstrukcije odgovarajućeg metoda za nalaženje ekstremuma bez ograničenja, uz stalno proveravanje da li je rešenje generisano bezuslovnom optimizacijom unutar dozvoljene oblasti. U slučaju da su uslovi narušeni, potrebno je da se učine odgovarajući koraci za vraćanje u dozvoljenu oblast.

Gradijentni metodi za bezuslovnu optimizaciju se mogu adaptirati za rešavanje proizvoljnih ograničenja prema sledećim principima:

1. Načiniti korak prema algoritmu osnovnog gradijentnog metoda.

2. U slučaju da se zadati uslovi optimizacije ne narušavaju, koristiti istu tačku kao i za osnovni gradijentni metod, inače učiniti korak u pravcu sumarnih antigradijentnih vektora narušenih ograničenja. Veliki problem u ovakvom algoritmu je određivanje kriterijuma za prekid metoda.

I drugi metodi bezuslovne optimizacije se mogu adaptirati tako da budu primenljivi za rešavanje optimizacionih zadataka sa proizvoljnim ograničenjima. Na primer, u metodima slučajnog pretraživanja se u proceduru *limit* ugrađuju dodatne provere za proveru ograničenja (3.1.1). U slučaju da je bar jedan od uslova narušen, korak se smatra neuspešnim.

3.2. Neki osnovni metodi

3.2.1. KOMPLEKS METOD ZA FUNKCIONALNA OGRANIČENJA

Opisani kompleks metod za bezuslovnu optimizaciju može da se prilagodi, tako da bude primenljiv za optimizaciju programa sa funkcionalnim ograničenjima koja su zadata nejednakosti. Pod *kompleksom* se podrazumeva

"oblak" slučajnih tačaka, ravnomerno raspoređenih unutar dozvoljene oblasti. Koristići heurističke principe, *kompleks* se na odgovarajući način kreće prema ekstremumu ciljne funkcije $Q(\mathbf{x})$ [60]. Neka je zadata ciljna funkcija $Q(\mathbf{x})$, čiji se maksimum traži prema ograničenjima

$$x_{\min_i} \le x_i \le x_{\max_i}, \quad i = 1, \dots, n, \tag{3.2.1}$$

$$\Psi_{\min_j} \le x_i \le \Psi_j(\mathbf{x}) \le \Psi_{\max_j}, \quad j = 1, \dots, m_2. \tag{3.2.2}$$

U slučaju da su neka ograničenja zadata jednostrano, tj. u slučaju izostanka nekih od veličina $\Psi_{\max_j}$ ili $\Psi_{\min_j}$, one se mogu zadati kao fiktivni, veoma veliki, odnosno veoma mali realni brojevi.

Algoritam kompleks metoda sa funkcionalnim ograničenjima može se iskazati sledećim koracima:

Korak 1. Zadati realne konstante ε_x ili ε_Q za proveru uslova za prekid algoritma, saglasno kriterijumu (3.2.5), odnosno (3.2.6).

Korak 2. Definisati broj tačaka kompleksa

$$N_k = \begin{cases} 2^n + 4, & n \le 3, \\ 2n + 4, & n > 3. \end{cases}$$

Korak 3. Unutar dozvoljene oblast $\Gamma_{x,\psi}$ formirati početni kompleks koji sadrži N_k tačaka $\mathbf{x}^{(j)}$, $j = 1, \dots, N_k$:

$$\begin{aligned} \mathbf{x}_i^{(j)} =& x_{\min_i} + \alpha_i^{(j)} \left(x_{\max_i} - x_{\min_i} \right), \\ & i = 1, \dots, n; \quad j = 1, \dots, N_k, \end{aligned} \tag{3.2.3}$$

gde su $\alpha_i^{(j)}$ slučajni brojevi iz intervala $[0, 1]$. Generisana tačka se uključuje u kompleks samo ako nisu narušena ograničenja (3.2.1) i (3.2.2). Ako broj neuspešno generisanih tačaka prevaziđe zadati broj N_v, algoritam se prekida. Broj N_v se određuje empirijski na sledeći način:

$$N_v = 100\, m_2.$$

Korak 4. U svakoj tački $\mathbf{x}^{(j)}$ kompleksa izračunava se vrednost ciljne funkcije $Q^{(j)} = Q(\mathbf{x}^{(j)})$.

Korak 5. Odrediti tačku $\mathbf{x}^{(b)}$ unutar kompleksa u kojoj ciljna funkcija ima najbolju vrednost, kao i tačku $\mathbf{x}^{(w)}$ u kojoj ciljna funkcija ima najlošiju vrednost, tj.

$$\begin{aligned} Q^{(b)} &= Q(\mathbf{x}^{(b)}) = \max_j Q^{(j)}, \\ Q^{(w)} &= Q(\mathbf{x}^{(w)}) = \min_j Q^{(j)}. \end{aligned} \tag{3.2.4}$$

Korak 6. Proveriti jedan od sledeća dva kriterijuma za prekid algoritma. Prvi kriterijum je zadat uslovom

$$\sqrt{\sum_{i=1}^{n}\left(x_i^{(b)}-x_i^{(w)}\right)^2}\leq\varepsilon_x, \tag{3.2.5}$$

gde je ε_x mali pozitivan broj koji, očigledno predstavlja rastojanje između najbolje i najlošije vrednosti ciljne funkcije.
Drugi kriterijum se zadaje uslovom

$$|\,Q^{(b)}-Q^{(w)}\,|\leq\varepsilon_Q. \tag{3.2.6}$$

Ako je jedan od uslova (3.2.5) ili (3.2.6) ispunjen, prekida se algoritam, i edituju se vrednosti $Q^{(b)}$ i $\mathbf{x}^{(b)}$, a inače se algoritam nastavlja od sledećeg koraka.

Korak 7. Izračunavaju se koordinate nove tačke $x^{(N)}$ u pravcu najbolje tačke $\mathbf{x}^{(b)}$:

$$x_i^{(N)}=2x_i^{(b)}-x_i^{(w)}=x_i^{(b)}-\left(x_i^{(w)}-x_i^{(b)}\right), \tag{3.2.7}$$

za $i=1,\ldots,n$.

Korak 8. Proveriti ograničenja (3.2.1), koristeći opisanu proceduru *limit*.

Korak 9. U tački $\mathbf{x}^{(N)}$ se proveravaju ograničenja (3.2.1). Ako je makar jedno od njih narušeno, izračunavaju se koordinate nove tačke $\tilde{\mathbf{x}}^{(N)}$, koja se nalazi na sredini između $\mathbf{x}^{(N)}$ i $\mathbf{x}^{(b)}$:

$$\tilde{\mathbf{x}}_i^{(N)}=\frac{1}{2}\left(x_i^{(b)}+x_i^{(N)}\right),\quad i=1,\ldots,n. \tag{3.2.8}$$

Sve dok tačka $\tilde{\mathbf{x}}^{(N)}$ narušava uslove (3.2.2) primenjivati formulu (3.2.8), pri čemu se koristi $\mathbf{x}^{(N)}=\tilde{\mathbf{x}}^{(N)}$.

Korak 10. Izračunati $Q^{(N)}=Q(\mathbf{x}^{(N)})$ ili $Q^{(N)}=Q(\tilde{\mathbf{x}}^{(N)})$.

Korak 11. Ako je $Q^{(N)}>Q^{(w)}$, tačka $\mathbf{x}^{(N)}$ se uzima za novu tačku kompleksa, na mesto tačke $\mathbf{x}^{(w)}$, tj. staviti smene $\mathbf{x}^{(w)}=\mathbf{x}^{(N)}$, $Q^{(w)}=Q^{(N)}$. Algoritam se nastavlja od *Koraka 5.*

Korak 12. Ako je $Q^{(N)}\leq Q^{(w)}$, nova tačka $\mathbf{x}^{(N)}$ se izostavlja i izračunavaju se koordinate tačke $\mathbf{x}^{(M)}$, koja se nalazi na sredini između $\mathbf{x}^{(w)}$ i $\mathbf{x}^{(b)}$, tj.

$$\mathbf{x}_i^{(M)}=\frac{1}{2}\left(x_i^{(b)}+x_i^{(w)}\right),\quad i=1,\ldots,n.$$

Korak 13. Ako su ograničenja (3.2.1) narušena, definiše se nova tačka $\tilde{\mathbf{x}}^{(M)}$, koja se nalazi na sredini između $\mathbf{x}^{(M)}$ i $\mathbf{x}^{(b)}$,tj.

$$\tilde{\mathbf{x}}_i^{(M)} = \frac{1}{2}\left(x_i^{(b)} + x_i^{(M)}\right), \quad i = 1, \ldots, n.$$

Sve dok su ograničenja (3.2.2) narušena primenjivati transformacije slične opisanim u *Koraku 9.*

Korak 14. Izračunati $Q^{(M)} = Q\left(\mathbf{x}^{(M)}\right)$ ili $Q^{(M)} = Q\left(\tilde{\mathbf{x}}^{(N)}\right)$.

Korak 15. Nova tačka $\tilde{\mathbf{x}}^{(N)}$ se uzima umesto $\mathbf{x}^{(w)}$, tj. postavljaju se smene $\mathbf{x}^{(w)} = \tilde{\mathbf{x}}^{(M)}$, $Q^{(w)} = Q^{(M)}$. Algoritam se nastavlja od *Koraka 5.*

U cilju implementacije ovog metoda napisan je sledeći program:

```
kompleks[q_,epsilon_,epsilonq_,xmin_List,xmax_List,
                    ksiMin_List,ksiMax_List]:=
  Block[{Q,i,ii,n,j,qIzracunato,Nk,Nv,b,x,ispitaj=1,
                    naruseno=True},
     n=Length[Variables[q]];
     If[n<4,Nk=2^ n+4,Nk=2 n+4]; Nv=100 Nk;
     x=Array[0,{n,Nk+1}]; Q=Array[0,Nk];
     x=pocetnikompleks[xmin,xmax,ksiMin,ksiMax];
     If[x=={{}},Return["Nema resenje"]];
     For[i=1,i<=Nk,i++,
          Q[[i]]=izracunajFunkciju[q,kolona[x,i]]];
     b=1;w=1;
     While[True,
     (*IZRACUNAVANJE MAX,MIN,B,W*)
     Q[[b]]=izracunajFunkciju[q,kolona[x,b]];
     Q[[w]]=izracunajFunkciju[q,kolona[x,w]];
     qMax=Max[Q]; qMin=Min[Q];
     For[i=1,i<=Nk,i++,If[qMax==Q[[i]],b=i,{}];
                    If[qMin==Q[[i]],w=i,{}];
     ];
     (*ISPITIVANJE USLOVA IZLAZA*)
     For[i=1,i<n+1,i++,
        If[Or[Sqrt[Sum[(x[[i,b]]-x[[i,w]])^2]]<=epsilon,
           Abs[qMax-qMin]<=epsilonq],
   Return[{izracunajFunkciju[q,kolona[x,b]],kolona[x,b]}]];
     ];
     Do[x[[ii,Nk+1]]=2 x[[ii,b]]-x[[ii,w]],{ii,n}];
```

```
      Do[If[x[[ii,Nk+1]]<xmin[[ii]],x[[ii,Nk+1]]=xmin[[ii]],
         If[x[[ii,Nk+1]]>xmax[[ii]],
              x[[ii,Nk+1]]=xmax[[ii]]]],{ii,n}];
      While[naruseno,
         naruseno=limit1[kolona[x,Nk+1],ksiMin,ksiMax];
         If[naruseno,
      For[i=1,i<=n,i++,x[[i,Nk+1]]=1/2(x[[i,b]]+x[[i,Nk+1]])]];
         ];
      qIzracunato=izr5acunajFunkciju[q,kolona[x,Nk+1]];
      If[qMin<qIzracunato,
         For[i=1,i<=n,i++,x[[i,w]]=x[[i,Nk+1]]];
         qMin=qIzracunato,
      For[i=1,i<=n,i++,
         x[[i,Nk+1]]=1/2(x[[i,b]]+x[[i,w]])]];
      naruseno=True;
      While[naruseno,
         naruseno=limit1[kolona[x,Nk+1],ksiMin,ksiMax];
         If[naruseno,
         For[i=1,i<=n,i++,
              x[[i,Nk+1]]=1/2(x[[i,b]]+x[[i,Nk+1]])]];
       ];
      qIzracunato=izracunajFunkciju[q,kolona[x,Nk+1]];
      For[i=1,i<=n,i++,x[[i,w]]=x[[i,Nk+1]]];
      qMin=qIzracunato;
      ]
  ];

limit1[x_List,ksiMin_List,ksiMax_List]:=
  Block[{i,naruseno=False},
      Do[If[And[ksiMin[[i]]<x[[i]]<ksiMax[[i]],!naruseno],
           naruseno=False,naruseno=True]
         ,{i,n}];
      Return[naruseno] ];

pocetnikompleks[xmin_List,xmax_List,ksiMin_List,ksiMax_List]:=
  Block[{i,j,ispitaj=1,uspesno},
      For[j=1,j<=Nk,j++,
         For[i=1,i<n+1,i++,uspesno=True;
           While[uspesno,
```

```
            alfa=Random[];
            If[ispitaj>Nv,Return[{{}}],{}];
            x[[i,j]]=xmin[[i]]+alfa(xmax[[i]]-xmin[[i]]);
            If[ksiMin[[i]]<x[[i,j]]<ksiMax[[i]],
               uspesno=False,uspesno=True;ispitaj++];
         ]]
      ];
      Return[x] ];

kolona[mat_,kol_]:=
  Block[{niz={},i},
      For[i=1,i<=Dimensions[mat][[1]],i++,
           niz=Append[niz,mat[[i,kol]]]];
      Return[niz] ];

izracunajFunkciju[q_,x0_List]:=
  Block[{j,q1,i},
      q1=q;
      Do[q1=q1/.y[[i]]->x0[[i]],{i,n}];
      Return[q1] ];
```

Rezultati testiranja programa.

In[1]:=kompleks[g^2,1,1,{1},{2},{1},{2}]
Out[1]:={g^2,{1.96875}}
In[2]:=kompleks[g^2,1,1,{1},{2},{1},{2}]
Out[1]:={g^2,{1.53272}}

3.2.2. DODAVANJE OGRANIČENJA

Ovaj metod ima brzu konvergenciju i može da se koristi za nalaženje globalnog ekstremuma multimodalne ciljne funkcije, kako za ograničenja tipa $\mathbf{x}\in\Gamma_{\mathbf{x}}$, tako i za ograničenja zadata funkcionalnim nejednakostima $\psi(\mathbf{x})\geq 0$.

Neka je data ciljna funkcija $Q(\mathbf{x})$ sa ograničenjima

$$\mathbf{x} \in \Gamma_x \tag{3.2.10}$$

i proizvoljnim ograničenjima

$$\psi_j(\mathbf{x}) \geq 0, \quad j = 1, \ldots, m_2. \tag{3.2.11}$$

Ovaj metod, u slučaju maksimuma, koristi sledeći algoritam:

Korak 1. Odrediti početnu tačku $\mathbf{x}_0$ unutar dozvoljene oblasti.

Korak 2. Nekom od poznatih metoda izračunati lokalni maksimum Q^* i odgovarajuću tačku $\mathbf{x}^*$ sa zadatom tačnošću ε_Q, saglasno uslovu (3.2.6).

Korak 3. Uslovima (3.2.10) i (3.2.11) dodaje se novo ograničenje

$$Q(\mathbf{x}) - (Q^* + \varepsilon_Q) \geq 0. \tag{3.2.12}$$

Algoritam se nastavlja od *Koraka 1.*

Korak 4. Algoritam se prekida ako se za M_n pokušaja ne nađe nova tačka $\mathbf{x}_0$ nakon poslednjeg dodavanja uslova.

U primeni ovog metoda najveći problem je da se pronađe početna tačka $\mathbf{x}_0$, kao i ispunjenje kriterijuma za prekid metoda. Ovaj problem može se rešiti ako se uvede efektivna procedura za brzo rešavanje sistema (3.2.10), (3.2.11) i (3.2.12), ili ukoliko se pokaže da takvo rešenje ne postoji. Poznati metodi iz literature nisu dovoljno efikasni za složene multimodalne funkcije.

3.3 Metodi kaznenih funkcija

Cilj ovih metoda je da se izračuna minimum ciljne funkcije $Q(\mathbf{x})$ na skupu dopustivih rešenja S, tj. da se reši problem

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\text{P.O.: } \mathbf{x} \in S. \end{aligned}$$

Ako se u razmatranje uvede funkcija $P(\mathbf{x})$ pomoću

$$P(\mathbf{x}) = \begin{cases} 0, & x \in S, \\ +\infty, & x \notin S, \end{cases}$$

i definiše *proširena funkcija cilja* pomoću

$$F(\mathbf{x}) = Q(\mathbf{x}) + P(\mathbf{x}),$$

tada je jasno da je svaki bezuslovni minimum proširene funkcije cilja istovremeno uslovni minimum funkcije $Q(\mathbf{x})$. Najveći problem je u tome što je funkcija $P(\mathbf{x})$ prekidna upravo na granici skupa S, gde se nalazi većina optimalnih rešenja. Ovaj problem se može ublažiti ako se $P(\mathbf{x})$ aproksimira nizom funkcija $P_k(\mathbf{x})$ od kojih svaka sadrži neki "parametar kazne" $\rho^{(k)} > 0$. Grubo govoreći, kada $k \to \infty$ i $\rho^{(k)} \to 0$, tada $P_k(\mathbf{x}) \to P(\mathbf{x})$.

Posmatrajmo sada opšti nelinearni problem matematičkog programiranja:

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n, \\ &\text{P.O.: } f_i(\mathbf{x}) \le 0, \quad i \in \mathcal{P} = \{1, \dots, p\}, \\ &\qquad h_j(\mathbf{x}) = 0, \quad j \in \mathcal{Q} = \{1, \dots, q\}, \end{aligned} \tag{3.3.1}$$

ili samo

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n, \\ &\text{P.O.: } f_i(\mathbf{x}) \le 0, \quad i \in \mathcal{P} = \{1, \dots, p\}. \end{aligned} \tag{3.3.2}$$

Suština metoda kaznenih funkcija jeste da se opšti zadatak uslovnog nelinearnog programiranja svede na bezuslovni problem ili na niz bezuslovnih programa.

U ovom odeljku se izučava implementacija metoda kaznenih funkcija za rešavanje nelinearnih uslovnih programa, koristeći mogućnosti simboličkog procesiranja u funkcionalnim programskim jezicima LISP i MATHEMATICA. Osim ranije navedenih, može se sugerisati sledeća prednost koja proizilazi iz simboličke implementacije metoda uslovne optimizacije u funkcionalnim programskim jezicima.

(5C) Mogućnosti simboličkog procesiranja u transformaciji unutrašnje forme uslovnog nelinearnog problema u unutrašnju formu odgovarajućeg problema bezuslovne optimizacije, saglasno principima koji proizilaze iz nekih metoda uslovne optimizacije. Generisana unutrašnja forma, primenljiva u bezuslovnoj optimizaciji, može biti plasirana u listu formalnih parametara izabrane procedure za bezuslovnu optimizaciju.

Izbor metoda bezuslovne optimizacije može se ostvariti sledećom funkcijom:

```
(* Izbor metoda bezuslovne optimizacije *)
unconstrained[f_,lprom_]:=
    Block[{metod,rez,arg,xx,lha,hh,hhmin,xxmin,xxmax,eeps},
       Print["Izaberi visedimenzionaln optimizaciju."];
       Print["<1> slucajni smerovi "];
       Print["<2> slucajno trazenje sa obrnutim korakom "];
       Print["<3> nametnuta slucajnost "];
       Print["<4> Powellov visedimenzionalni  "];
       Print["<5> osnovni gradijentni "];
       Print["<6> modifikovani gradijentni "];
       Print["<7> automatska korekcija koraka "];
```

```
Print["<8> steepest descent "];
Print["<9> Newtonov "];
Print["<10> modifikovani Newtonov "];
Print["<11> Markuardov "];  Print["<12> DFP "];
Print["<13> Conjugate gradients "];
metod=Input[];
Which[(metod==1)||(metod==2)||(metod==3),
  xx=Input["Pocetna iteracija?  "];
  hh=Input["Lista koraka?  "];
  lha=Input["Velicina smanjenja koraka?  "];
  hhmin=Input["Lista minimalnih koraka?  "];
  xxmin=Input["Minimalne vrednosti argumenata?"];
  xxmax=Input["Maksimalne vrednosti argumenata?"];
  Which[metod==1,
    rez=SluDir[f,lprom,xx,hh,hhmin,lha,xxmin,xxmax],
        metod==2,
    rez=SluOb[f,lprom,xx,hh,hhmin,lha,xxmin,xxmax],
        metod==3,
    rez=SluCon[f,lprom,xx,hh,hhmin,lha,xxmin,xxmax]
  ],
  (metod==5)||(metod==6)||(metod==7),
  xx=Input["Pocetna iteracija?  "];
  hh=Input["Lista koraka?  "];
  xxmin=Input["Minimalne vrednosti argumenata?"];
  xxmax=Input["Maksimalne vrednosti argumenata?"];
  eeps=Input["tacnost?  "];
  Which[metod==5,
      rez=OsnGrad[f,lprom,xx,hh,xxmin,xxmax,eeps],
          metod==6,
      rez=ModGrad[f,lprom,xx,hh,xxmin,xxmax,eeps],
          metod==7,
      rez=AutGrad[f,lprom,xx,hh,xxmin,xxmax,eeps]
  ],
  (metod==8)||(metod==9)||(metod==10)||(metod==11)||
  (metod==12)||(metod==13),
  xx=Input["Pocetna iteracija?  "];
  eeps=Input["tacnost?  "];
  Which[metod==8,  rez=Cauchy[f,lprom,xx,eeps],
        metod==9,  rez=newton[f,lprom,xx,eeps],
```

```
            metod==10, rez=newtonm[f,lprom,xx,eeps],
            metod==11, rez=mark[f,lprom,xx,eeps],
            metod==12, rez=Dfp[f,lprom,xx,eeps],
            metod==13, rez=Mkg[f,lprom,xx,eeps]
      ]
    ];
    Return[rez]
]
```

3.3.1. METODI SPOLJAŠNJIH KAZNENIH FUNKCIJA

Metodi spoljašnjih kaznenih funkcija se koriste za rešavanje problema tipa (3.3.1). U ovim metodima optimalno rešenje se dostiže pomoću tačaka iz spoljašnosti skupa dopustivih rešenja. Bazična ideja sadržana u ovim metodima jeste rešavanje niza metoda bezuslovne minimizacije, čija rešenja u graničnom procesu generišu minimum nelinearnog uslovnog problema.

U jednom od metoda spoljašnjih kaznenih funkcija, nelinearni problem (3.3.1) se prevodi u sledeći niz bezuslovnih problema [73]:

$$\begin{aligned}\min_{\mathbf{x}} F(\mathbf{x}) &= \min_{\mathbf{x}} \left(Q(\mathbf{x}) + \frac{1}{\rho^{(k)}} P(\mathbf{x}) \right) \\ &= \min_{\mathbf{x}} \left(Q(\mathbf{x}) + \frac{1}{\rho^{(k)}} \left\{ \sum_{i=1}^{p} [f_i(\mathbf{x})]_+^{\alpha} + \sum_{j=1}^{q} |h_j(\mathbf{x})|^{\beta} \right\} \right),\end{aligned} \tag{3.3.3}$$

gde je $[f_i(\mathbf{x})]_+ = \max\{0, f_i(\mathbf{x})\}$ i $\rho^{(k)}$ strogo opadajući niz pozitivnih brojeva, dok su α i β dva cela broja koji ispunjavaju uslov $\alpha, \beta \geq 1$.

Pri implementaciji ovog metoda prvo se specificira parametar kazne $\rho = \rho^{(0)}$ (na primer, $\rho^{(0)} = 1$), a zatim se rešava problem

$$\min_{\mathbf{x}} F(\mathbf{x}).$$

Tada se smanji vrednost parametra ρ (na primer, pomoću $\rho^{(1)} = 0.5\rho^{(0)}$ ili $\rho^{(1)} = 0.1\rho^{(0)}$) i rešava se novi problem bezuslovne optimizacije.

Pretpostavimo da je uslovni optimizacioni problem (3.3.1) u paketu MATHEMATICA zadat sledećim elementima:

$q\_$: ciljna funkcija;
$prom\_$: lista promenljivih;

$vf\_$: lista ograničenja zadatih relacijom $\leq$;
$vh\_$: lista ograničenja zadatih relacijom $=$.

Sada je dat opis odgovarajuće funkcije.

Korak 1. Opadajući niz $\rho^k = \{ro[1], ro[2], \dots\}$ generiše se sledećom rekurzivnom definicijom:

```
ro[1]=1;                ro[n_]:=ro[n-1]/2
```

Takođe, može se koristiti i jedinstveni simbol ro umesto niza. U sledećem izrazu niz ρ^k je predstavljen lokalnom promenljivom ro, čija je početna vrednost 1, a svaka naredna se dobija na osnovu prethodne prema $ro = ro/2$. Tada funkcija $fgoal$ u i-toj iteraciji može da se formira na sledeći način [52]:

```
fgoal=q+1/ro*(Sum[Max[0, vf[[i]]^alpha, {i,Length[vf]}]
            +Sum[(Abs[vh[[i]])^beta, {i,Length[vh]}];
```

Ako se koristi niz $\{ro[1], ro[2], \dots\}$, tada se funkcija $fgoal$ definiše na sledeći način:

```
fgoal=q+1/ro[[i]]* (Sum[Max[0,vf[[i]]^alpha,{i,Length[vf]}]
                 +Sum[(Abs[vh[[i]])^beta,{i,Length[vh]}];
```

Korak 2. Unutrašnja forma Q_u, $\{\mathbf{x}\}$ indukovanog bezuslovnog problema jeste uređeni par $\{fgoal, prom\}$.

Ovim je opisana suština prednosti **(5C)** za slučaj metoda spoljašnjih kaznenih funkcija.

Primer 3.3.1. Unutrašnja forma ciljne funkciji Q_u, koja odgovara nelinearnom problemu iz Primera 1.2.1, jednaka je:

```
-x1-x2 + 1/ro (Abs[-1+x1^2-x2^2])^beta .
```

U sledećem programu niz $\rho^{(k)}$ definisan je lokalnom promenljivom ro:

```
(*METOD SPOLJASNJIH KAZNENIH FUNKCIJA*)
metodaS[q_,prom_List,vf_List,vh_List,alpha_,beta_,eps_]:=
   Module[{f,fp,i,p,l,prva,druga,ro=1},
     p=Length[vf];   l=Length[vh];
     fp=Sum[Max[0,vf[[i]]]^alpha,{i,p}]+
        Sum[(Abs[vh[[i]]])^beta,{i,l}];
     f=q+1/ro*fp;   prva=unconstrained[f,prom];
     ro=ro/2;  f=q+1/ro*fp;  druga=unconstrained[f,prom];
     While[Abs[prva[[2]]-druga[[2]]]>eps,
       prva=druga; ro=ro/2; f=q+1/ro*fp;
       druga=unconstrained[f,prom];
     ];
```

```
      Return[druga] ]
```

Sledi odgovarajući program u slučaju kada je niz $\rho^{(k)}$ definisan funkcijom:

```
metodaS1[q_,prom_List,vf_,vh_,alpha_,beta_,eps_,index_]:=
   Block[{x0,h0,hmin,xmin,xmax,lh,x1,f,i,p,l,nnn,
              prva,druga,ind=index,izbor},
     ro[1]=1; ro[nnn_]:=ro[nnn-1]/2;
     p=Length[vf];  l=Length[vh];
     f=q+1/ro[ind]*(Sum[Max[0,vf[[i]]]^alpha,{i,p}]
           +Sum[(Abs[vh[[i]]])^beta,{i,l}]);
     prva=unconstrained[f,prom];    ind=ind+1;
     f=q+1/ro[ind]*(Sum[Max[0,vf[[i]]]^alpha,{i,p}]
           +Sum[(Abs[vh[[i]]])^beta,{i,l}]);
     prva=unconstrained[f,prom];
     epsilon=Input["Unesi tacnost:"];
     While[Abs[prva[[2]]-druga[[2]]]>epsilon,
           prva=druga;     ind=ind+1;
           f=q+1/ro[ind]*(Sum[Max[0,vf[[i]]]^alpha,{i,p}]
                 +Sum[(Abs[vh[[i]]])^beta,{i,l}]);
           druga=unconstrained[f,prom];
     ];
     Return[druga] ];
```

Test primeri.

```
In[1]:= metodaS[x^2/y^2-1,{x,y},{x-y-3},{x^2-y-2},1,1,0.1]
prva = {{2.02885, 2.10864}, -0.0666493}
                                2
                               x                2
Simbolicka funkcija: -1 + -- + y  + (Abs[-2 + x  - y] + Max[0, -3 + x - y])
                               2
                              y
Izaberi metod visedimenzionalne optimizacije.
? 1
Pocetna iteracija? {1,1}
Lista koraka? {0.1,0.2}
Velicina smanjenja koraka? 4
Lista minimalnih koraka? {0.01,0.02}
Minimalne vrednosti argumenata? {-10,-10}
Maksimalne vrednosti argumenata? {10,10}
1 za minimum, 2 za maksimum 1
{{1.06284, 1.10732}, 1.89897}
{{1.08822, 1.22161}, 1.83093}
...
{{2.03133, 2.10662}, -0.0504978}
```

```
{{2.02885, 2.10864}, -0.0666493}
prva = {{2.02885, 2.10864}, -0.0666493}
                                  2
                                 x                    2
Simbolicka funkcija: -1 + -+ 2 (Abs[-2 + x  - y] + Max[0, -3 + x - y])
                                  2
                                 y
Izaberi metod visedimenzionalne optimizacije.
? 1
Pocetna iteracija? {2.02885,2.10864}
Lista koraka? {0.1,0.2}
Velicina smanjenja koraka? 4
Lista minimalnih koraka? {0.01,0.02}
Minimalne vrednosti argumenata? {-10,-10}
Maksimalne vrednosti argumenata? {10,10}
1 za minimum, 2 za maksimum 1
{{2.06899, 2.26593}, -0.136669}
{{2.0922, 2.44009}, -0.139248}
...
{{2.18169, 2.74605}, -0.34134}
{{2.18119, 2.75072}, -0.357522}
prva[[2]]-druga[[2]]=0.290872
Izaberi metod visedimenzionalne optimizacije.
? 2
Pocetna iteracija? {2.18119,2.75072}
Lista koraka? {0.1,0.2}
Velicina smanjenja koraka? 4
Lista minimalnih koraka? {0.01,0.02}
Minimalne vrednosti argumenata? {-10,-10}
Maksimalne vrednosti argumenata? {10,10}
1 za minimum, 2 za maksimum 1
{{2.13631, 2.61431}, -0.130228}
{{2.17604, 2.73243}, -0.354883}
{{2.16808, 2.72679}, -0.262965}
{{2.17149, 2.72724}, -0.318579}
prva[[2]]-druga[[2]]=-0.0389423
Out[1]= {{2.17149, 2.72724}, -0.318579}
```

3.3.2. METODI UNUTRAŠNJIH KAZNENIH FUNKCIJA

Ovi metodi se koriste za rešavanje problema tipa (3.3.2). U metodima spoljašnjih kaznenih funkcija, optimalno rešenje se dostiže pomoću tačaka iz unutrašnjosti skupa dopustivih rešenja. Neki od najpopularnijih metoda unutrašnjih kaznenih funkcija prevode nelinearni problem (3.3.1) u sledeći

problem bezuslovne optimizacije, koji idgovara k-toj iteraciji ([73]):

$$\begin{aligned}\min_{\mathbf{x}} F(\mathbf{x}) &= \min_{\mathbf{x}} \left(Q(\mathbf{x})+\rho^{(k)}P(\mathbf{x})\right) \\ &= \min_{\mathbf{x}} \left(Q(\mathbf{x})+\rho^{(k)}\sum_{i=1}^{p}\frac{1}{[f_i(\mathbf{x})]^2}\right),\end{aligned} \tag{3.3.4}$$

$$\begin{aligned}\min_{\mathbf{x}} F(\mathbf{x}) &= \min_{\mathbf{x}} \left(Q(\mathbf{x})+\rho^{(k)}P(\mathbf{x})\right) \\ &= \min_{\mathbf{x}} \left(Q(\mathbf{x})+\rho^{(k)}\sum_{i=1}^{p}\frac{1}{\min\{0, f_i(\mathbf{x})\}}\right),\end{aligned} \tag{3.3.5}$$

gde je $\rho^{(k)}$ strogo opadajući niz pozitivnih brojeva.

Ovde se razmatra implementacija metoda unutrašnjih kaznenih funkcija koji je definisan pomoću (3.3.4).

Unutrašnja forma odgovarajućeg bezuslovnog minimizacionog problema, u MATHEMATICA može da se generiše na sledeći način:

```
fgoal=q+ro*Sum[1/vf[[i]]^2, {i,Length[vf]}];
```

Unutrašnja forma generisanog bezuslovnog problema je određena elementima $fgoal$ i $prom$.

Ovim je opisana suština prednosti **(5C)** za slučaj metoda unutrašnjih kaznenih funkcija.

Sada sledi implementacija metoda unutrašnjih kaznenih funkcija u MATHEMATICA, za niz $\rho^{(k)}$ određen lokalnom promenljivom.

```
(*METODA UNUTRASNJIH KAZNENIH FUNKCIJA*)
metodaU[q_,prom_List,vf_List,eps_]:=
    Block[{f,fp,i,p,ro=1,prva,druga},
      p=Length[vf]; fp=Sum[1/vf[[i]]^2,{i,p}]; f=q+ro*fp;
      prva=unconstrained[f,prom]; ro=ro/2; f=q+ro*fp;
      druga=unconstrained[f,prom];
      While[Abs[prva[[2]]-druga[[2]]]>eps,
         prva=druga;  ro=ro/2; f=q+ro*fp;
         druga=unconstrained[f,prom];
      ];
      Return[druga] ]
```

Slična je funkcija u MATHEMATICA, i u slučaju kada je niz $\rho^{(k)}$ definisan funkcijom.

```
metodaU1[q_,prom_List,vf_List,eps_,index_]:=
    Block[{p,f,i,nnn,ind=index},
      ro[1]:=1;   ro[nnn_]:=ro[nnn]/2;  p=Length[vf];
      f=q+ro[ind]* Sum[1/vf[[i]]^2,{i,p}];
      prva=unconstrained[f,prom];  ind=ind+1;
      f=q+ro[index] *Sum[1/fi[[i]]^2,{i,p}];
      druga=unconstrained[f,prom];;
      While[Abs[prva[[2]]-druga[[2]]]>eps,
           prva=druga; ind=ind+1;
           f=q+ro[index] Sum[1/fi[[i]]^2,{i,p}];
           druga=unconstrained[f,prom];
      ];
      Return[druga] ]
```

Test primeri.

In[1]:= metodaU[x^2/y^2-1,{x,y},{x-y-3},0.01]

$$f = -1 + (-3 + x - y)^{-2} + \frac{x^2}{y^2}$$

Izaberi metod visedimenzionalne optimizacije.
? 2
Pocetna iteracija? {1.0,1.1}
Lista koraka? {0.1,0.1}
Velicina smanjenja koraka? 4
Lista minimalnih koraka? {0.01,0.02}
Minimalne vrednosti argumenata? {-10,-10}
Maksimalne vrednosti argumenata? {10,10}
1 za minimum, 2 za maksimum 1
{{0.937165, 1.04634}, -0.0943484}
{{0.962551, 1.10348}, -0.137758}
{{0.874865, 1.09285}, -0.262571}
{{0.791528, 1.07527}, -0.36539}
. . .
{{-0.0000418553, 0.87142}, -0.933281}
{{-0.00292587, 0.868865}, -0.933281}
bezuslovna optimizacija daje {{-0.00292587, 0.868865}, -0.933281}
prva = {{-0.00292587, 0.868865}, -0.933281}

$$f = -1 + \frac{1}{2\,(-3 + x - y)^2} + \frac{x^2}{y^2}$$

Izaberi metod visedimenzionalne optimizacije.

```
? 5
Pocetna iteracija? {-0.00292587,0.868865}
Lista koraka? {0.1,0.1}
Minimalne vrednosti argumenata? {-10,-10}
Maksimalne vrednosti argumenata? {10,10}
tacnost? 0.01
Zelite li maximum(2) ili minimum(1)? 1
0.0196869{{-0.00387365, 0.870591}, -0.966672}
0.0185954{{-0.00457083, 0.872314}, -0.966706}
0.017977{{-0.00508559, 0.874037}, -0.966738}
...
0.0153065{{-0.00799299, 1.03144}, -0.969297}
0.015289{{-0.00800755, 1.03297}, -0.96932}
bezuslovna optimizacija daje
{{-0.00800755, 1.03297}, -0.96932}
druga = {{-0.00800755, 1.03297}, -0.96932}
prva[[2]]-druga[[2]]=0.0360395
Izaberi metod visedimenzionalne optimizacije.
? 9
Pocetna iteracija? {-0.00800755,1.03297}
tacnost? 0.001
Zelite li minimum(1) ili maksimum(2)? 1
0.0106969{{-0.00800755, 1.03297}, -0.98463}
0.0106969{{-0.0207617, 2.37327}, -0.991331}
0.00530005{{-0.0311644, 4.15191}, -0.995098}
0.00265171{{-0.0356698, 6.52139}, -0.997233}
0.00124858{{-0.034675, 9.68206}, -0.998441}
bezuslovna optimizacija daje {{-0.034675, 9.68206}, -0.998441}
Izaberi metod visedimenzionalne optimizacije.
? 9
Pocetna iteracija? {-0.034675,9.68206}
tacnost? 0.0001
Zelite li minimum(1) ili maksimum(2)? 1
0.000630583{{-0.034675, 9.68206}, -0.999214}
0.000630583{{-0.0300923, 13.8735}, -0.999558}
0.000266145{{-0.0243167, 19.47}, -0.999751}
0.000108606{{-0.018678, 26.9401}, -0.99986}
bezuslovna optimizacija daje {{-0.018678, 26.9401}, -0.99986}
Out[1]= {{-0.018678, 26.9401}, -0.99986}
```

3.3.3. GENERALISANI LAGRANGEOVI MNOŽITELJI

Ovaj metod se koristi za rešavanje opšteg problema tipa (3.3.1). Da bi se primenila tehnika Lagrangeovih množitelja, ograničenja zadata nejednakostima se moraju prevesti u jednakosti, uvodeći pomoćne (*slack*) promenljive, po jednu za svako ograničenje zadato pomoću nejednakosti. Na taj način, opšti problem (3.3.1) se prevodi u sledeći problem sa ograničenjima tipa

jednakosti:

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\text{P.O.: } f_i(\mathbf{x}) + z_i^2 = 0, \quad i \in \mathcal{P} = \{1, \dots, p\}, \\ &\qquad h_j(\mathbf{x}) = 0, \quad j \in \mathcal{Q} = \{1, \dots, q\}. \end{aligned} \tag{3.3.6}$$

Ovakav optimizacioni problem se transformiše u sledeći niz bezuslovnih minimizacionih problema:

$$\begin{aligned} &\min_{\mathbf{x},\ \mathbf{z}} L(\mathbf{x},\ \mathbf{z}) = \min_{\mathbf{x},\ \mathbf{z}} \left(Q(\mathbf{x}) + P(\mathbf{x},\ \mathbf{z})\right) \\ &= \min_{\mathbf{x},\ \mathbf{z}} \left(Q(\mathbf{x}) + \sum_{i=1}^{p} \mu_i (f_i(\mathbf{x}) + z_i^2) + \sum_{j=1}^{q} \mu_{j+p+1} |h_j(\mathbf{x})| \right), \end{aligned} \tag{3.3.7}$$

gde su μ_i, $i = 1, \dots, p+q+1$, nenegativni težinski faktori nezavisni od $\mathbf{x}$, poznati kao Lagrangeovi množitelji, dok vektor $\mathbf{z}$ sadrži *slack* promenljive z_i, $i = 1, \dots, p$.

Pomoćne promenljive z_i i težinski koeficijenti μ_i u paketu MATHEMATICA mogu se implementirati nizovima $z[i]$ i $vecm[i]$, gde je i brojačka promenljiva koja se koristi u ciklusu. Unutrašnja forma $fgoal$ ciljne funkcije Q_u u MATHEMATICA se formira pomoću sledećih izraza:

```
fgoal=q[[1]]+Sum[vecm[i] (vf[[i]]+z[i]^2), {i,Length[vf]}]
     +Sum[vecm[i+Length[vf]] Abs[vh[[i]]], {i,Length[vh]}]
```

Ova dva ciklusa formiraju analitički izraz ciljne funkcije odgovarajućeg bezuslovnog problema. Očigledno je da su u ovom analitičkom izrazu sadržane pomoćne (*slack*) promenljive $z[i]$ i $vecm[i]$. Time je potvrđena prednost **(4C)**.

Osim toga, očigledna je efektivnost s kojom je izgrađen bezuslovni optimizacioni problem koji proizilazi iz zadatog uslovnog optimizacionog problema.

Takođe, potrebno je da lista parametara bude proširena pomoćnim promenljivama $z[i]$ i $vecm[i]$.

```
Do[prom=Append[prom,z[i]],{i,Length[vf]}];
Do[prom=Append[prom,vecm[i]],{i,Length[vf]+Length[vh]}];
```

Ovim je verifikovana prednost **(3C)** simboličke implementacije uslovnih optimizacionih problema.

Unutrašnja forma generisanog bezuslovnog optimizacionog problema jeste uredeni par $fgoal, prom$.

Izbor metoda kojim se rešava indukovani bezuslovni problem određen je sledećom funkcijom:

```
unconstrained1[f_,lprom_,px_]:=
    Block[metod,rez,lha,hh,hhmin,xxmin,xxmax,eeps,
       Print["Izaberi visedimenzionalnu optimizaciju."];
       Print["<1> slucajni smerovi "];
       Print["<2> slucajno trazenje sa obrnutim korakom "];
       Print["<3> nametnuta slucajnost "];
       Print["<4> Powellov visedimenzioni  "];
       Print["<5> osnovni gradijentni "];
       Print["<6> modifikovani gradijentni "];
       Print["<7> automatska korekcija koraka "];
       Print["<8> steepest descent "];
       Print["<9> Newtonov "];
       Print["<10> modifikovani Newtonov "];
       Print["<11> Markuardov "]; Print["<12> DFP "];
       Print["<13> Conjugate gradients "];
       metod=Input[];
       Which[(metod==1)||(metod==2)||(metod==3),
         hh=Input["Lista koraka?  "];
         lha=Input["Velicina smanjenja koraka?  "];
         hhmin=Input["Lista minimalnih koraka?  "];
         xxmin=Input["Minimalne vrednosti argumenata?  "];
         xxmax=Input["Maksimalne vrednosti argumenata?  "];
         Which[metod==1,
            rez=SluDir[f,lprom,px,hh,hhmin,lha,xxmin,xxmax],
               metod==2,
            rez=SluOb[f,lprom,px,hh,hhmin,lha,xxmin,xxmax],
               metod==3,
            rez=SluCon[f,lprom,px,hh,hhmin,lha,xxmin,xxmax]
         ],
         (metod==5)||(metod==6)||(metod==7),
         hh=Input["Lista koraka?  "];
         xxmin=Input["Minimalne vrednosti argumenata?  "];
         xxmax=Input["Maksimalne vrednosti argumenata?  "];
         eeps=Input["tacnost?  "];
         Which[metod==5,
           rez=OsnGrad[f,lprom,px,hh,xxmin,xxmax,eeps],
               metod==6,
```

```
          rez=ModGrad[f,lprom,px,hh,xxmin,xxmax,eeps],
              metod==7,
          rez=AutGrad[f,lprom,px,hh,xxmin,xxmax,eeps]
        ],
        (metod==8)||(metod==9)||(metod==10)||(metod==11)||
        (metod==12)||(metod==13),
        eeps=Input["tacnost?  "];
        Which[metod==8,   rez=Cauchy[f,lprom,px,eeps],
              metod==9,   rez=newton[f,lprom,px,eeps],
              metod==10,  rez=newtonm[f,lprom,px,eeps],
              metod==11,  rez=mark[f,lprom,px,eeps],
              metod==12,  rez=Dfp[f,lprom,px,eeps],
              metod==13,  rez=Mkg[f,lprom,px,eeps]
        ]
      ];
      Return[rez] ]
```

Sledi sada implementacija metoda generalisanih Lagrangeovih množitelja:

```
(*vecm:Lagrangeovi koeficijenti*)
metodaL[q_,prom1_List,vf_List,vh_List]:=
   Block[{max,prom=prom1,f,i,p,l,lg,xx,hh,hhmin,
            eelha,xxmin,xxmax,izbor},
     p=Length[vf];  l=Length[vh];
     Do[prom=Append[prom,z[i]],{i,p}];
     Do[prom=Append[prom,vecm[i]],{i,p+l}];
     f=q+(Sum[vecm[i]*(vf[[i]]+z[i]^2),{i,p}]+
          Sum[vecm[i+p]*Abs[vh[[i]]],{i,l}]);
     Print["Odgovarajuci bezuslovni problem ",f,prom];
     xx=Print["Pocetna iteracija duzine ",p];
     xx=Input[]; lg=True; i=1;
     While[i<=p && lg,
           lg=vrednost[vf[[i]],prom1,xx]<=0; i++
     ];
     If[lg==False,
         Print["Pocetna iteracija van dozvoljene oblasti"];
         Return[{}]
     ];
     (* Izracunati vrednosti zi koeficijenata *)
     Do[xx=Append[xx,Sqrt[-vrednost[vf[[i]],prom1,xx]]],
              {i,p}];
```

```
        Print["Vrednosti mi koeficijenata duzine ",p+l];
        vecm=Input[];
        Do[xx=Append[xx,vecm[i]],{i,p+l}]; Print[xx];
        Return[unconstrained1[f,prom,xx]] ]

  vrednost[jednacina_,prom1_List,x_List]:=
  Block[{jed=jednacina,i,prom=prom1,x0=x},
        Do[jed=jed/.prom[[i]]->x0[[i]],{i,Length[prom]}];
        Return[jed] ]
```

Numerički rezultati.

```
In[1]:= metodaL[x^2/y^2-1,{x,y},{x-y-3},{x^2-y-2}]
prom={x, y, z[1], vecm$6[1], vecm$6[2]}
Odgovarajuci bezuslovni problem
         2
        x              2                                      2
> -1 + - + Abs[-2 + x  - y] vecm$6[2] + vecm$6[1] (-3 + x - y + z[1] )
         2
        y
> {x, y, z[1], vecm$6[1], vecm$6[2]}
Pocetna iteracija duzine 2
? {0.2,0.2}
2 vrednosti mi koeficijenata
? 2
? 1
prom = {x, y, z[1], vecm$6[1], vecm$6[2]
xx = {0.2, 0.2, 1.73205, 2, 1}
Izaberi metod visedimenzionalne optimizacije.
<1> slucajni smerovi
<2> slucajno trazenje sa obrnutim korakom
<3> nametnuta slucajnost
<4> Powellov visedimenzioni
<5> osnovni gradijentni
<6> modifikovani gradijentni
<7> automatska korekcija koraka
<8> steepest descent
<9> Newtonov
<10> modifikovani Newtonov
<11> Markuardov
<12> DFP
<13> Conjugate gradients
? 1
Lista koraka? {0.1,0.1,0.1,0.1,0.1}
Velicina smanjenja koraka? 4
Lista minimalnih koraka? {0.01,0.02,0.02,0.05,0.01}
Minimalne vrednosti argumenata? {-10,-10,-10,-10,-10}
Maksimalne vrednosti argumenata? {10,10,10,10,10}
```

```
1 za minimum, 2 za maksimum 1
{{0.220916, 0.236886, 1.7881, 2.00856, 1.0152}, 2.45528}
{{0.202197, 0.200778, 1.74538, 2.00761, 1.01514}, 2.30265}
{{0.202263, 0.201818, 1.73245, 2.00281, 1.00184}, 2.17292}
{{0.200956, 0.202132, 1.73534, 2.00281, 1.00237}, 2.17577}
{{0.203041, 0.201623, 1.73521, 2.00288, 1.00161}, 2.20274}
{{0.201277, 0.203514, 1.73417, 2.00286, 1.0011}, 2.15373}
...
{{0.209407, 0.222008, 1.74161, 2.01654, 1.01133}, 2.13407}
{{0.208804, 0.223555, 1.7417, 2.01621, 1.01077}, 2.11368}
{{0.20785, 0.222327, 1.74283, 2.01872, 1.00826}, 2.11753}
{{0.207972, 0.223361, 1.74274, 2.018, 1.01014}, 2.11308}
{{0.207953, 0.22238, 1.74178, 2.01876, 1.01004}, 2.11456}
{{0.209892, 0.221603, 1.74199, 2.01833, 1.01051}, 2.14362}
{{0.207949, 0.220876, 1.74401, 2.0163, 1.0113}, 2.14637}
{{0.207376, 0.223731, 1.74386, 2.01741, 1.00949}, 2.11041}
Out[1]= {{0.207376, 0.223731, 1.74386, 2.01741, 1.00949}, 2.11041}
```

Primer 3.3.2. Dat je nelinearni optimizacioni problem:

$$\begin{aligned} &\text{Minimizirati:} \quad -x_1 - x_2 \\ &\text{P.O.:} \quad f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 \leq 0, \\ &\qquad\quad f_2(x_1, x_2) = -x_1 + x_2^2 \leq 0. \end{aligned}$$

Koristeći metod unutrašnjih kaznenih funkcija, za slučaj $\rho^{(k)} = 1/2^k$ i koristeći DFP metod sa tačnošću 10^{-7} u generisanim bezuslovnim optimizacionim parametriam, dobijaju se sledeći rezultati:

k	$x_1^{(k)}$	$x_2^{(k)}$	$Q(\mathbf{x}^{(k)})$
0	0.5	0.5	$-1.$
1	0.63085851	0.09223888	2.56860712
2	0.70733854	0.67997758	-1.35642624
3	0.70670616	0.70666650	-1.41252669
4	0.70709348	0.70709348	-1.41416037
5	0.70710636	0.70710636	-1.41421188
6	0.70710676	0.70710676	-1.41421350
7	0.70710677	0.70710677	-1.41421352

Tabela 3.3.1

S druge strane, pomoću poznate tradicionalne implementacije, metod unutrašnjih kaznenih funkcija produkuje rezultate prezentovane u Tabeli 3.3.2

([73]). Evidentno je da funkcionalna implementacija daje bolje rezultate. Na primer, tačnost 10^{-6} u Tabeli 3.3.3 je dostignuta u petoj iteraciji, a u Tabeli 3.3.4 tek u 53. iteraciji.

k	$x_1^{(k)}$	$x_2^{(k)}$	$Q(\mathbf{x}^{(k)})$
2	0.6884721	0.3952927	−1.0837648
3	0.7106337	0.3713147	−1.0819484
4	0.7349830	0.4535905	−1.1785735
5	0.7276601	0.5228195	−1.2504796
6	0.7251426	0.5758051	−1.3009477
7	0.7203736	0.6143570	−1.3358093
8	0.7151591	0.6446030	−1.3597621
9	0.7105652	0.6567412	−1.3763306
10	0.7072276	0.6803946	−1.3876222
⋮	⋮	⋮	⋮
53	0.7071067	0.7071067	−1.4142134

Tabela 3.3.2

Može se pokazati da je *simboličkom implementacijom* poboljšan veliki deo kriterijuma koji su preporučljivi za ocenu nekog nelinearnog optimizacionog algoritma [13]:

1. Veličina (dimenzionalnost, broj ograničenja tipa jednakosti i/ili nejednakosti) zadatog problema. Ograničenja tipa jednakosti i nejednakosti smeštena su u listi, tako da njihov broj nije unapred limitiran. S druge strane, u jezicima FORTRAN i C, ako je skup ograničenja smešten u niz $X(1), \ldots, X(50)$, tada je dozvoljena maksimalna dimenzija problema 50.

2. Jednostavnost korišćenja (vreme potrebno za unošenje podataka i funkcije u kompjuterski program). Ciljna funkcija je zadata proizvoljnim MATHEMATICA, odnosno SCHEME aritmetičkim izrazima, koji su inkorporirani u unutrašnju formu problema.

3. Jednostavnost napisanog programa kojim se implementira algoritam.

Koristeći funkcionalne programske jezike obezbeđeno je sledeće:

– Mogućnost da se koriste ciljna funkcija i ograničenja, bez leksičke ili sintaksne analize.

– Jednostavna implementacija parcijalnih izvoda ciljne funkcije.

– Elegantan metod transformacije date unutrašnje forme koja predstavlja uslovni optimizacioni problem u unutrašnju formu odgovarajućeg bezuslovnog optimizacionog problema.

3.3.4. JOŠ JEDAN METOD KAZNENIH FUNKCIJA

Optimizacioni zadatak sa ograničenjima tipa nejednakosti može da se reši istim metodom koji je korišćen za optimizaciju sa funkcionalnim ograničenjima tipa jednakosti. Nova uopštena funkcija se formira pomoću

$$R(\mathbf{x}) = Q(\mathbf{x}) + \beta_0 H(\mathbf{x}),$$

gde je $H(\mathbf{x})$ "kaznena" funkcija, definisana sa

$$H(\mathbf{x}) = \sum_{j=1}^{m_2} [1 + \operatorname{sign}(\Psi_j(\mathbf{x}))]\, \Psi_j(\mathbf{x}),$$

gde je

$$\operatorname{sign}(\Psi_j(\mathbf{x})) = \begin{cases} +1, & \text{ako je ograničenje } \Psi_j(\mathbf{x}) \text{ narušeno ,} \\ -1, & \text{ako ograničenje } \Psi_j(\mathbf{x}) \text{ nije narušeno.} \end{cases}$$

Koeficijent β_0 je pozitivan broj, koji se bira prema kriterijumu

$$\beta_0 \left| \frac{\partial H(\mathbf{x})}{\partial x_i} \right| \gg \left| \frac{\partial Q(\mathbf{x})}{\partial x_i} \right|, \quad i = 1, \dots, n. \tag{3.3.10}$$

4. POSEBNI SLUČAJEVI USLOVNE OPTIMIZACIJE

4.1. Konveksno programiranje

Konveksno programiranje je najbolje obrađeno područje nelinearnog programiranja. Problem u kome treba da se izračuna minimum neke funkcije $Q(\mathbf{x})$ u nekom zadatom skupu F naziva se matematički program. U konveksnom programu skup F je posredno zadat:

$$F = \big\{x \in \mathbb{R}^n : \ f_1(\mathbf{x}) \le 0, \dots, f_p(\mathbf{x}) \le 0\big\},$$

gde su $f_1, \dots, f_p$ proizvoljne konveksne funkcije. U tom slučaju se konveksni program može pretsaviti u obliku

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\text{P.O.: } f_i(\mathbf{x}) \le 0, \quad i \in \mathcal{P} = \{1, \dots, p\} \end{aligned} \tag{4.1.1}$$

gde su f_0, $f_1, \dots, f_p$ neke konveksne funkcije.

Za rešavanje problema (4.1.1) postoji više metoda.

4.1.1. GRADIJENTNI METOD

Metod je osmišljen je kao modifikacija Cauchyevog metoda najstrmijeg pada za matematičke programe sa ograničenjima. Polazi se iz startne tačke, zadate unutar konveksne oblasti određene ograničenjima. Zatim se sledi pravac najbržeg opadanja funkcije, uz poštovanje ograničenja. Zbog jednostavnosti, posmatraćemo konveksni program sa linearnim ograničenjima, tj.

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\text{P.O.: } A\mathbf{x} \leq \mathbf{b}, \qquad x_i \geq 0,\ i = 1, \ldots, n. \end{aligned} \tag{4.1.2}$$

Označimo sa F skup dopustivih rešenja

$$F = \{\mathbf{x} : \ A\mathbf{x} \leq \mathbf{b},\ x_i \geq 0\}.$$

Metod je iterativan, što znači da je optimalno rešenje programa (4.1.2) granična tačka niza njegovih aproksimacija $\mathbf{x}^k$, $k = 0, 1, \ldots$ Svaka aproksimacija $\mathbf{x}^k$ se dobija nakon rešavanja linearnog programa i jednodimenzionalnog pretraživanja. Algoritam se može iskazati u obliku:

Korak 1. Izabrati početnu dopustivu aproksimaciju $\mathbf{x}^0 \in F$. Izračunati $\nabla Q(\mathbf{x}^0)$ i specificirati pravilo zaustavljanja, tj. dovoljno mali broj $\varepsilon > 0$, takav da se algoritam prekida kada je

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \varepsilon.$$

Korak 2. Rešiti linearni program

$$\begin{aligned} &\text{Minimizirati: } \nabla Q(\mathbf{x}^{(k)}) \cdot \mathbf{x}, \\ &\text{P.O.: } A\mathbf{x} \leq \mathbf{b}, \qquad x_i \geq 0,\ i = 1, \ldots, n. \end{aligned} \tag{4.1.3}$$

Neka je $\mathbf{x}_*^{(k)}$ njegovo optimalno rešenje.

Korak 3. Rešiti jednodimenzionalni program

$$\begin{aligned} &\text{Minimizirati: } Q(\mathbf{x}^{(k)} + \lambda(\mathbf{x}_*^{(k)} - \mathbf{x}^{(k)})), \\ &\text{P.O.: } \ 0 \leq \lambda \leq 1. \end{aligned}$$

U ovom slučaju funkcija Q se minimizira na linijskom segmentu koji spaja $\mathbf{x}^{(k)}$ sa $\mathbf{x}_*^{(k)}$. Neka je λ_k njegovo optimalno rešenje.

Korak 4. Izračunati novu aproksimaciju

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda_k(\mathbf{x}_*^{(k)} - \mathbf{x}^{(k)}).$$

Korak 5. Ako je pravilo zaustavljanja ispunjeno, proces se zaustavlja; $\mathbf{x}^{k+1}$ je prihvatljiva aproksimacija optimuma. Inače, vratiti se na *Korak 2.*

4.1.2. METOD DOPUSTIVIH SMEROVA

Postoji više metoda dopustivih smerova, a razlikuju se po uvedenim poboljšanjima kod nalaženja skupa aktivnih ograničenja, kao i načina izračunavanja dopustivih smerova [73]. Za funkciju u kojoj je implementiran ovaj metod bitna su dva uslova: ciljna funkcija, koja može da bude proizvoljna konveksna funkcija i skup ograničenja koji je zadat proizvoljnim konveksnim diferencijabilnim funkcijama. Posmatrajmo problem

$$\begin{aligned} &\text{Minimizirati } Q(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\text{P.O.:} \quad f_i(\mathbf{x}) \leq 0, \quad i \in \mathcal{P} = \{0, \dots, p\}, \end{aligned} \tag{4.1.4}$$

gde su Q i $f_i, i \in \mathcal{P}$ konveksne diferencijabilne funkcije. Neka skup dopustivih rešenja bude označen sa F, tj.

$$F = \{\mathbf{x} : f_i(\mathbf{x}) \leq 0,\ i \in \mathcal{P}\}$$

Za proizvoljno fiksno dopustivo rešenje $\mathbf{x}$, označimo sa $\mathcal{P}(\mathbf{x})$ skup aktivnih ograničenja, tj.

$$\mathcal{P}(\mathbf{x}) = \{i \in \mathcal{P} \ : \ f_i(\mathbf{x}) = 0\}.$$

Kažemo da je vektor $\mathbf{d}$ ima dopustivi smer iz tačke $\mathbf{x} \in F$ ako $\mathbf{d}$ vodi iz tačke $\mathbf{x}$ u skup dopustivih rešenja, tj. ako postoji pozitivan broj $\bar{a} > 0$ takav da je

$$\mathbf{x} + a\mathbf{d} \in F \text{ za svako } \ 0 \leq a \leq \bar{a}.$$

Dakle, dopustivi smerovi su svi oni smerovi koji iz tačke na granici skupa dopustivih rešenja vode u skup dopustivih rešenja. U slučaju izbora tačke iz unutrašnjosti skupa dopustivih rešenja, svi smerovi su dopustivi.

Metodi dopustivih smerova su iterativni. Tipičan metod dopustivih smerova koristi sledeće iteracije, koje su definisane za $k = 0, 1, \dots$ U tački $\mathbf{x}^{(k)}$ prvo se odredi neki dopustiv smer $\mathbf{d}^{(k)}$. Zatim se funkcija cilja pretražuje u smeru $\mathbf{d}^{(k)}$ iz tačke $\mathbf{x}^{(k)}$. Pri tome je važno da se pretraživanje vrši samo u skupu dopustivih rešenja F. Na taj način dolazimo do tačke

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \sigma_k \mathbf{d}^{(k)},$$

gde je σ_k rešenje jednodimenzionalnog programa

$$\begin{aligned} &\text{Minimizirati } Q(\mathbf{x}^{(k)} + \sigma \mathbf{d}^{(k)})) \\ &\text{P.O.:} \quad \mathbf{x}^{(k)} + \sigma \mathbf{d}^{(k)} \in F, \quad \sigma \geq 0. \end{aligned}$$

Optimalno rešenje σ_k zove se optimalna dužina koraka k-te iteracije. Metodi dopustivih smerova međusobno se razlikuju prema tome kako se određuju dopustivi smerovi i kako se određuje optimalna dužina σ_k koraka σ.

Algoritam jednog od metoda dopustivih smerova je ukratko izložen:

Korak 1. Odrediti početno dopustivo rešenje $\mathbf{x}^{(0)} \in F$ i specificirati pravilo zaustavljanja (na primer, dovoljno mali broj $\varepsilon > 0$ takav da se algoritam prekida kada je u nekoj normi $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \varepsilon$).

Korak 2. Odrediti aktivna ograničenja $\mathcal{P}(\mathbf{x}^{(k)})$ i rešiti linearni program

$$\begin{aligned} &\text{Maximizirati} \quad a \\ &\text{P.O.:} \ \nabla Q(\mathbf{x}^{(k)})\mathbf{d} + a \le 0, \\ &\qquad \nabla f_i(\mathbf{x}^{(k)})\mathbf{d} + a \le 0, \quad i \in \mathcal{P}(\mathbf{x}^{(k)}), \\ &\qquad d_i \le 1, \quad i = 1, \ \dots \ , n, \\ &\qquad -d_i \le 1, \quad i = 1, \ \dots \ , n, \end{aligned}$$

gde je vektor d zadat koordinatama $d_1, \dots \ , d_n$. Označimo sa $\mathbf{d}^{(k)}$ i a_k njegovo optimalno rešenje. Ako je $a_k = 0$, proces se prekida; $x^{(k)}$ je optimalno rešenje programa. Ako je $a_k > 0$, vektor $\mathbf{d}^{(k)}$ koristi se kao dopustivi smer; dakle, nastavlja se sa algoritmom.

Korak 3. Rešiti jednodimenzionalni problem u odnosu na promenljivu σ:

$$\begin{aligned} &\text{Minimizirati} \ Q(\mathbf{x}^{(k)} + \sigma\mathbf{d}^{(k)}) \\ &\text{P.O.:} \quad f_i(\mathbf{x}^{(k)} + \sigma\mathbf{d}^{(k)}) \le 0, \quad i \in \mathcal{P}, \ \sigma \ge 0. \end{aligned}$$

Označimo optimalno rešenje (dužinu koraka) sa σ_k.

Korak 4. Izračunajti novu aproksimaciju $x^{(k+1)} = x^{(k)} + \sigma\mathbf{d}^{(k)}$. Ako je pravilo zaustavljanja zadovoljeno, proces se prekida; $x^{(k+1)}$ je prihvatljivo optimalno rešenje. Inače, vratiti se na *Korak 2* i ponoviti iteraciju sa indeksom $k + 1$.

Program u LISPu kojim je realizovana opisana varijanta metoda dopustivih smerova koristi ideje već navedene kod realizacije gradijentnog metoda, ali je isuviše glomazan da bi bio bar u detaljima ovde dat. Može se naći u [41].

IV GLAVA

Optimizacija i linearni sistemi

1. UVOD

Gradijentni metodi optimizacije drugog reda, kao i metodi višekriterijumske optimizacije, mogu biti primenjeni na izračunavanje generalisanih inverza i rešavanje linearnih sistema. Ovakvom primenom dobijeno je nekoliko metoda za izračunavanje *najmanje-kvadratnog rešenja* (LSS rešenja) i *najmanje kvadratnog rešenja minimalne norme* ($NLSS$ rešenja) zadatog sistema linearnih jednačine $Ax = b$, kao i odgovarajućih generalisanih inverza. Prvo je razvijen iterativni metod za izračunavanje LSS rešenja datog linearnog sistema $Ax = b$, koji je baziran na primeni modifikacije Newtonovog optimizacionog metoda u minimizaciji funkcije $\|Ax-b\|^2$. Ovaj metod može da se transformiše u analogni metod za generisanje $\{1,3\}$ generalisanih inverza matrice A. Koristeći Newtonov metod, kao i modifikovani Newtonov metod, u minimizaciji funkcije $\|Ax - b\|^2 + \alpha\|x\|^2$, $\alpha \to 0+$, uvedeno je nekoliko iterativnih metoda za izračunavanje $NLSS$ rešenja i odgovarajućih iterativnih procesa za generisanje Moore-Penroseovog inverza.

Takodje, razmatran je dvoetapni minimizacioni problem za izračunavanje $NLSS$ rešenja, koji je postavljen u [4] i [5], kao višekriterijumski optimizacioni metod sa ciljnim funkcijama $\|Ax - b\|^2$ i $\|x\|^2$. Koristeći metod kaznenih funkcija u rešavanju dobijenog višekriterijumskog optimizacionog problema, razvijeno je više metoda za generisanje $NLSS$ rešenja datog linearnog sistema.

Opisani metodi su implementirani u paketu MATHEMATICA.

1.1. Norme vektora i matrica

U literaturi su česte definicije vektorske norme pomoću skalarnog proizvoda, kao i uopštenje vektorske norme pomoću pozitivno definitnih matrica.

Definicija 1.1.1. Ako x^* označava vektor dobijen primenom konjugacije i transponovanja na vektoru $x \in \mathbb{C}^n$, tada se uobičajeni skalarni proizvod $\langle \cdot, \cdot \rangle$ definiše sa

$$(\forall x, y \in \mathbb{C}^n) \qquad \langle x, y \rangle = x^* y.$$

Euklidova norma vektora $x \in \mathbb{C}^n$ definisana je jednakošću $\|x\| = \langle x, x \rangle^{1/2}$, a vektori x i y su ortogonalni po Euklidovoj normi ako je $\langle x, y \rangle = 0$.

Pored Euklidove, koriste se i sledeće vektorske norme [73]:

(i) p-norme ili Hölderove norme, definisane sa

$$\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}, \qquad p \geq 1;$$

(ii) $$\|x\|_\infty = \max_i |x_i|.$$

Od matričnih normi najčešće se koriste sledeće norme (videti [73]):

$$\|A\|_1 = \max_{1 \leq k \leq n} \sum_{i=1}^{m} |a_{ik}|; \qquad \|A\|_2 = \sigma_{max};$$

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{k=1}^{n} |a_{ik}|; \qquad \|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{k=1}^{n} |a_{ik}|^2} = \sqrt{\sum_{i=1}^{r} \sigma_i(A)};$$

$$\|A\|_M = \max(m, n) \max_{i,k} |a_{ik}|; \quad \|A\|_G = \sqrt{mn} \max_{i,k} |a_{ik}|,$$

gde $\sigma_{\max}$ označava maksimalnu singularnu vrednost za A.

1.2. Moore-Penroseov inverz

Poznat je veći broj ekvivalentnih definicija Moore-Penroseovog inverza. R. Penrose je 1955. godine dokazao sledeću teoremu (videti [32]):

Teorema 1.2.1 (Penrose)**.** *Za datu matricu $A \in \mathbb{C}^{m,n}$ postoji jedinstvena matrica $X \in \mathbb{C}^{n,m}$ koja ispunjava jednačine*

$$\begin{array}{ll} (1) \quad AXA = A & (2) \quad XAX = X \\ (3) \quad (AX)^* = AX & (4) \quad (XA)^* = XA. \end{array} \tag{1.2.1}$$

Penrose je matricu X označio sa $A^\dagger$ i nazvao je *generalisani inverz matrice* A. Za matricu $A^\dagger$ koristi se naziv Moore-Penroseov inverz matrice A.

Teorema 1.2.2 (Moore)**.** *Za datu matricu $A \in \mathbb{C}^{m,n}$ postoji jedinstvena matrica $X \in \mathbb{C}^{n,m}$ tako da za pogodno izabrane matrice Y i Z važi:*

$$AXA = A, \qquad X = YA^* = A^*Z.$$

Osim toga je

$$XAX = X, \qquad (AX)^* = AX, \qquad (XA)^* = XA.$$

Definicija 1.2.1 (Funkcionalna definicija generalisanog inverza)**.** Za datu matricu $A \in \mathbb{C}^{m\times n}$ definišimo linearnu transformaciju $\tilde{A}^\dagger : \mathbb{C}^m \to \mathbb{C}^n$ relacijom $\tilde{A}^\dagger x = 0$ ako $x \in R(A)^\perp$ i $\tilde{A}^\dagger x = \left(\tilde{A}_{|_{R(A^*)}}\right)^{-1} x$ ako $x \in R(A)$. Matrica linearne transformacije $\tilde{A}^\dagger$ označava se sa $A^\dagger$ i naziva se generalisani inverz za A.

Definicija 1.2.2. (Mooreova definicija) Generalisani inverz za $A \in \mathbb{C}^{m\times n}$ je jedinstvena matrica $A^\dagger$ takva da je

$$\text{(i)} \quad AA^\dagger = P_{R(A)}\,, \quad \text{(ii)} \quad A^\dagger A = P_{R(A^\dagger)}.$$

Definicija 1.2.3 (Penroseova definicija [32]). Za $A \in \mathbb{C}^{m\times n}$ generalisani inverz je jedinstvena matrica $A^\dagger \in \mathbb{C}^{n\times m}$ koja zadovoljava jednačine (1), (2), (3) i (4).

Teorema 1.2.3. *Funkcionalna, Mooreova i Penroseova definicija generalisanog inverza su ekvivalentne.*

Teorema 1.2.4 [4]**.** *Neka je $A = PQ$ potpuna rang faktorizacija matrice A, tj. $P \in \mathbb{C}_r^{m\times r}$ i $Q \in \mathbb{C}_r^{r\times n}$. Tada je $A^\dagger = Q^\dagger P^\dagger = Q^*(QQ^*)^{-1}(P^*P)^{-1}P^*$.*

1.3. Aproksimativna svojstva generalisanih inverza

U ovom odeljku se posmatra problem rešavanja linearnog sistema $Ax = b$, pri čemu su $A \in \mathbb{C}^{m\times n}$ i $b \in \mathbb{C}^m$ zadati. Ovaj problem ima rešenje ako i samo ako je $b \in R(A)$; rešenje je jedinstveno ako i samo ako je $N(A) = \{0\}$. Osim toga, posmatra se sledeći problem: ako $b \notin R(A)$, tada odrediti x, tako da Ax bude "najbliže" vektoru b.

Definicija 1.3.1 ([4]**,** [42]**).** Dat je sistem jednačina $Ax = b$, $A \in \mathbb{C}^{m\times n}$. Pseudoinverz X je minimalne norme ako je za svako $b \in R(A)$, $x = Xb$ rešenje jednačine $Ax = b$ i ako je

$$\min_{Ax=b} \|x\| = \|Xb\|.$$

Teorema 1.3.1 ([4], [42])**.** *X je g-inverz matrice A takav da je Xb rešenje sistema $Ax = b$ minimalne norme ako i samo ako X ispunjava uslove*

$$AXA = A \qquad\qquad (XA)^* = XA.$$

Teorema 1.3.2 ([4])**.** *Neka je $A \in \mathbb{C}^{m\times n}$, $b \in \mathbb{C}^m$. Ako je sistem $Ax = b$ saglasan, vektor $x = A^{(1,4)}b$, $A^{(1,4)} \in A\{1,4\}$ je jedinstveno rešenje za koje je norma $\|x\|$ najmanja. Važi i obrnuto, tj. ako je matrica $X \in \mathbb{C}^{n\times m}$ takva da u slučaju saglasnog sistema $Ax = y$, $x = Xb$ predstavlja rešenje sa najmanjom normom, tada $X \in A\{1,4\}$.*

Definicija 1.3.2 ([42])**.** Neka je dat nesaglasni sistem jednačina $Ax = b$. Vektor x_0 je najmanje kvadratno rešenje ako je

$$\|Ax_0 - b\| \leq \|Ax - b\| \quad (x \in \mathbb{C}^n).$$

Teorema 1.3.4 ([42])**.** *X je g-inverz od A, takav da je Xb najmanje srednje-kvadratno rešenje jednačine $Ax = b$, za svako $b \in \mathbb{C}^m$, ako i samo ako X ispunjava uslove*

$$AXA = A, \qquad (AX)^* = AX.$$

Teorema 1.3.5 ([4])**.** *Za $A \in \mathbb{C}^{m\times n}$ i $b \in \mathbb{C}^m$, $\|Ax - b\|$ je najmanje za $x = A^{(1,3)}b$. Obrnuto, ako je $X \in \mathbb{C}^{n\times m}$ takva matrica da je za svaki vektor b norma $\|Ax - b\|$ najmanja, tada $X \in A\{1, 3\}$.*

Veza izmedju Moore-Penroseovog inverza i najmanje-kvadratnog rešenja minimalne norme, koju je prvi dokazao R. Penrose [33] data je sledećom teoremom.

Teorema 1.3.6. *Neka $A \in \mathbb{C}^{m\times n}$ i $b \in \mathbb{C}^m$. Tada, izmedju svih najmanje-kvadratnih rešenja jednačine $Ax = b$, minimalne norme je $x = A^\dagger b$. Obrnuto, ako je $X \in \mathbb{C}^{n\times m}$ takva matrica da je za svaki vektor b norma Xb najmanje-kvadratno rešenje minimalne norme, tada je $X = A^\dagger$.*

Aproksimativna svojstva generalisanih inverza mogu se detaljnije naći u [1], [4], [42]. Aproksimativna svojstva generalisanih inverza omogućavaju da se oni mogu izračunati minimizacijom dve norme, odnosno funkcije. Ova njihova korisna osobina omogućava da se izračunavanje generalisanih inverza kao i LSS, $NLSS$ rešenja dovede u vezu sa različitim metodima optimizacije.

Implementacija različitih metoda za izračunavanje LSS i $NLSS$, koji nisu bazirani na optimizaciji, može se naći u [25–28], [46], [57–58].

2. PRIMENA OPTIMIZACIONIH METODA

2.1. Primena gradijentnih metoda drugog reda

U literaturi je poznata veza linearne optimizacije i Moore-Penroseovog inverza ([7], [38–40]). Takodje, za izračunavanje Moore-Penroseovog inverza korišćena je i Cauchyeva tehnika najstrmijeg pada ([11], [67]), kao i metod konjugovanih gradijenata ([62]). Osnovna namera u ovom poglavlju je da se primene gradijentni metodi optimizacije drugog reda na izračunavanje generalisanih inverza i odgovarajućih rešenja datog linearnog sistema jednačina.

Na početku opisujemo minimizaciju ciljne funkcije $Q(x) = \|Ax - b\|^2$. Imajući u vidu konstantnu vrednost $\nabla^2 Q(x) = A^*A$ Hesseove matrice za

Q, Newtonov optimizacioni metod ne može biti primenjen u minimizaciji funkcije $Q(x)$. Medjutim, ovde se može koristiti modifikovani Newtonov metod.

Teorema 2.1.1. *Neka su H_1 i H_2 konačnodimenzionalni kompleksni Hilbertovi prostori, $n = \dim H_1$, $m = \dim H_2$ i neka je $A : H_1 \mapsto H_2$ linearni operator. Posmatrajmo sledeće iterativne procese, u odnosu na operatorsku jednačinu $Ax = b$*

$$x_{k+1} = x_k - (\lambda_k \mathbf{I} + A^*A)^{-1} A^*(Ax_k - b), \quad k = 0, 1, \ldots \tag{2.1.1}$$

$$X_{k+1} = X_k - (\lambda_k \mathbf{I} + A^*A)^{-1} A^*(AX_k - \mathbf{I}), \quad k = 0, 1, \ldots \tag{2.1.2}$$

u kojoj je $\lambda_k \geq 0$ opadajući niz koji ispunjava uslov $\lambda_k \to 0$ i $x_0 \in H_1$, $X_0 \in B(H_1, H_2)$.

(i) *Ako je x_0 zadato sa $x_0 = Tb$, pri čemu operator T ispunjava uslove $T \in B(H_2, H_1)$, $\operatorname{rang}(T) < \operatorname{rang}(A)$, tada niz x_k konvergira prema LSS rešenju sistema $Ax = b$.*

(ii) *Ako je x_0 izabrano prema kriterijumu $x_0 = Tb$, za $T \in B(H_2, H_1)$, $\operatorname{rang}(T) \geq \operatorname{rang}(A)$, tada x_k konvergira prema NLSS rešenju sistema $Ax = b$.*

(iii) *U slučaju kada je $X_0 \in B(H_2, H_1)$ ograničeni operator koji ispunjava uslov $\operatorname{rang}(X_0) < \operatorname{rang}(A)$, iterativni proces (2.1.2) konvergira ka $X_k \to X \in A\{1, 3\}$.*

(iv) *$X_k \to A^\dagger$ pod pretpostavkom da je $X_0 \in B(H_2, H_1)$ izabrano prema uslovu $\operatorname{rang}(X_0) \geq \operatorname{rang}(A)$.*

Dokaz. Kako je operator A^*A pozitivno definitan i $\alpha_k \geq 0$, tada je i operator $\alpha_k \mathbf{I} + A^*A$ pozitivno definitan, što obezbeđuje egzistenciju iterativnih procesa (2.1.1) i (2.1.2).

(i) Ovaj deo dokaza sledi iz suštine modifikovanog Newtonovog metoda, koji je primenjen na funkciju $Q(x) = \frac{1}{2}\|Ax - b\|^2$. Zaista, u tom slučaju je

$$\nabla Q(x_k) = A^*(Ax_k - b), \qquad \nabla^2 Q(x_k) = A^*A.$$

(iii) Koristeći poznati rezultat da je *LSS* rešenje dato sa $A^{(1,3)}b$, zaključujemo da niz x_k, $k = 0, 1, \ldots$, definisan u (2.1.1) ispunjava $x_k = X_k b$, gde je X_k aproksimacija za $A^{(1,3)}$. Tada se niz x_k, $k = 0, 1, \ldots$, dobija pomoću zamene $b = \mathbf{I}$ u (2.1.2) i on konvergira ka $A^{(1,3)}$.

Tvrđenja (ii) i (iv) sleduju iz $\operatorname{rang}(X_k) = \operatorname{rang}(X_{k+1})$, kao i iz poznatog rezultata: *Proizvoljan $\{1\}$-inverz X za A ispunjava uslov $X \in A\{1, 2\}$ ako i samo ako je* $\operatorname{rang}(X) = \operatorname{rang}(A)$ (videti [4], [42]). □

Primedba 2.1.1. (i) Formula (2.1.1) je slična sa sledećim poznatim rezultatom [3]:

$$A^{\dagger} = \lim_{\alpha \to 0} (\alpha \mathbf{I} + A^* A)^{-1} A^*.$$

Medjutim, u (2.1.2) konvergencija je obezbedjena za svaku fiksiranu vrednost realnih brojeva λ_k. Za veće vrednosti parametra λ_k iteracije (2.1.1) i (2.1.2) ponašaju se slično *steepest descent* metodu. U slučaju $\lambda_k = 0$ ovi iterativni procesi konvergiraju saglasno Newtonovom metodu.

(ii) Uslov $\|X_{k+1} - X_k\| \leq \epsilon$, gde je ϵ mali pozitivan broj, može da se koristi u ulozi kriterijuma završavanja iterativnog procesa (2.1.2). Zaista, koristeći $A^* A A^{\dagger} = A^*$ [4], zaključujemo $A^*(AX_k - \mathbf{I}) \to 0$, što implicira konvergenciju: $\|X_{k+1} - X_k\| \to 0$.

(iii) S obzirom na globalnu konvergenciju *steepest descent* metoda, kao i na brzu konvergenciju Newtonovog metoda, preporučuje se da metodi (2.1.1) i (2.1.2) startuju sa velikim početnim vrednostima za λ_k, kao i da se te vrednosti smanjuju posle uspešnih iteracija, a povećavaju posle neuspešnih.

Posledica 2.1.1. *Uočimo sledeće iterativne procese, vezane za operatorsku jednačinu $Ax = b$:*

$$x_{k+1} = x_k - (\lambda \mathbf{I} + A^* A)^{-1} A^* (A x_k - b), \quad \lambda \geq 0, \ x_0 \in H_1 \tag{2.1.3}$$

$$X_{k+1} = X_k - (\lambda \mathbf{I} + A^* A)^{-1} A^* (A X_k - \mathbf{I}), \quad \lambda \geq 0, \ X_0 \in B(H_1, H_2). \tag{2.1.4}$$

(i) Ako je x_0 dato sa $x_0 = Tb$, gde operator $T \in B(H_2, H_1)$ ispunjava uslov $\operatorname{rang}(T) < \operatorname{rang}(A)$, tada niz (2.1.3) konvergira prema LSS rešenju sistema $Ax = b$.

(ii) U slučaju $x_0 = Tb$, gde $T \in B(H_2, H_1)$ zadovoljava $\operatorname{rang}(T) \geq \operatorname{rang}(A)$, niz (2.1.3) konvergira prema NLSS rešenju sistema $Ax = b$.

(iii) U slučaju $\operatorname{rang}(X_0) < \operatorname{rang}(A)$ dobijamo $X_k \to X \in A\{1, 3\}$.

(iv) $X_k \to A^{\dagger}$ pod pretpostavkom da je matrica X_0 izabrana prema uslovu $\operatorname{rang}(X_0) \geq \operatorname{rang}(A)$.

Dokaz. Iterativni procesi (2.1.1) i (2.1.2) konvergiraju za sve vrednosti λ_k parametra λ. Prema tome, dokaz sleduje iz Teoreme 2.1.1, za svaku fiksiranu vrednost $\lambda = \lambda_k$ $(k = 0, 1, \dots)$. □

Sada izučavamo minimizacijau ciljne funkcije $Q(x) = \|Ax - b\|^2 + \alpha \|x\|^2$, gde je α proizvoljni realni broj.

Teorema 2.1.2. (i) *NLSS rešenje jednačine* $Ax = b$ *dato je sledećim iterativnim procesima:*

$$x_{k+1} = x_k - (\alpha_k \mathbf{I} + A^*A)^{-1} \left[A^*(Ax_k - b) + \alpha_k x_k\right], \tag{2.1.5}$$

$$x_{k+1} = x_k - ((\lambda_k + \alpha_k)\mathbf{I} + A^*A)^{-1} \left[A^*(Ax_k - b) + \alpha_k x_k\right], \tag{2.1.6}$$

$$x_{k+1} = x_k - ((\lambda + \alpha_k)\mathbf{I} + A^*A)^{-1} \left[A^*(Ax_k - b) + \alpha_k x_k\right], \tag{2.1.7}$$

za proizvoljne $x_0, \alpha_0, \lambda_0, \lambda \in H_1$, *pri čemu su* α_k *i* λ_k $(k = 1, \dots)$ *strogo opadajući nizovi koji ispunjavaju uslove* $\alpha_k \to 0$ *i* $\lambda_k \to 0$.

(ii) *Moore-Penroseov inverz* $A^\dagger$ *može biti generisan sledećim iterativnim procesima:*

$$X_{k+1} = X_k - (\alpha_k \mathbf{I} + A^*A)^{-1} \left[A^*(AX_k - \mathbf{I}) + \alpha_k X_k\right], \tag{2.1.8}$$

$$X_{k+1} = X_k - ((\lambda_k + \alpha_k)\mathbf{I} + A^*A)^{-1} \left[A^*(AX_k - \mathbf{I}) + \alpha_k X_k\right], \tag{2.1.9}$$

$$X_{k+1} = X_k - ((\lambda + \alpha_k)\mathbf{I} + A^*A)^{-1} \left[A^*(AX_k - \mathbf{I}) + \alpha_k X_k\right], \tag{2.1.10}$$

gde je $X_0 \in B(H_1, H_2)$ *proizvoljan ograničen operator,* α_0, λ_0 *su proizvoljni realni brojevi, a* α_k, λ_k $(k = 1, \dots)$ *su strogo opadajući nizovi koji ispunjavaju uslove* $\alpha_k \to 0$ *i* $\lambda_k \to 0$.

Dokaz. Egzistencija izraza (2.1.5)–(2.1.10) se lako proverava.

(i) Reprezentacije (2.1.5)–(2.1.7) proizilaze iz klasične regularizacione tehnike [64]: *Ako je* x_α *minimum funkcionala*

$$Q(x) = F_\alpha(x) := \frac{1}{2}\|Ax - b\|^2 + \alpha\frac{1}{2}\|x\|^2,$$

tada $\lim_{\alpha \to 0} x_\alpha = A^\dagger b$.

Lako se proveravaju i sledeća tvrdjenja:

$$\nabla Q(x_k) = A^*(Ax_k - b) + \alpha x_k, \quad \nabla^2 Q(x_k) = A^*A + \alpha \mathbf{I}.$$

Primenom Newtonovog metoda u toku minimizacije funkcionala $F_\alpha(x)$, dobijmo sledeću aproksimaciju za x_α:

$$x'_{k+1} = x'_k - (\alpha\mathbf{I} + A^*A)^{-1}\left[A^*(Ax'_k - b) + \alpha x_k\right]. \tag{2.1.11}$$

Slično, prema modifikovanom Newtonovom metodu sa opadajućim nizom $\lambda_k \to 0$, dobijamo iterativni metod

$$x'_{k+1} = x'_k - ((\lambda_k+\alpha)\mathbf{I} + A^*A)^{-1}\left[A^*(Ax'_k - b) + \alpha x'_k\right]. \tag{2.1.12}$$

koji ispunjava uslov $\lim\limits_{k\to\infty} x'_k = x_\alpha$.

Primenom modifikovanog Newtonovog metoda, sa fiksiranim vrednostima $\lambda_k = \lambda$ $(k = 0, 1, \dots)$ u svim iteracijama, dobijaju se sledeće aproksimacije za x_α:

$$x'_{k+1} = x'_k - ((\lambda+\alpha)\mathbf{I} + A^*A)^{-1}\left[A^*(Ax'_k - b) + \alpha x'_k\right]. \tag{2.1.13}$$

Koristeći strogo opadajući niz $\alpha_k \to 0$ u iterativnim procesima (2.1.11)–(2.1.13), dobijajamo iterativne procese (2.1.5)–(2.1.7).

(ii) Koristeći $b = \mathbf{I}$, saglasno sa (i), dobijamo $X_k \to A^\dagger\mathbf{I} = A^\dagger$. □

Parametar α u Teoremi 2.1.1 može se automatski smanjivati na način opisan u sledećoj teoremi:

Teorema 2.1.3. *Neka su ispunjeni uslovi Teoreme* 2.1.1. *Posmatrajmo proširene matrice* $A_N = \begin{bmatrix} A \\ \vdots \\ A \end{bmatrix}$, $\mathbf{I}_N = \begin{bmatrix} \mathbf{I} \\ \vdots \\ \mathbf{I} \end{bmatrix}$ *i prošireni vektor* $b_N = \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}$, $N \geq 1$. *Ako se iteracije* (2.1.1), (2.1.2), (2.1.5) *i* (2.1.8) *primene na* A_N, b_N *and* $\mathbf{I}_N$, *dobija se*

$$x_{k+1} = x_k - \left(\frac{\alpha_k}{N}\mathbf{I} + A^*A\right)^{-1} A^*(Ax_k - b), \tag{2.1.1′}$$

$$X_{k+1} = X_k - \left(\frac{\alpha_k}{N}\mathbf{I} + A^*A\right)^{-1} A^*(AX_k - \mathbf{I}), \tag{2.1.2′}$$

$$x_{k+1} = x_k - \left(\frac{\alpha_k}{N}\mathbf{I} + A^*A\right)^{-1}\left[A^*(Ax_k - b) + \frac{\alpha_k}{N}x_k\right], \tag{2.1.5′}$$

$$X_{k+1} = X_k - \left(\frac{\alpha_k}{N}\mathbf{I} + A^*A\right)^{-1}\left[A^*(AX_k - \mathbf{I}) + \frac{\alpha_k}{N}X_k\right]. \tag{2.1.8$'$}$$

Dokaz. Primena (2.1.1) na matricu A_N i vektor b_N daje

$$x_{k+1} = x_k - (\alpha_k\mathbf{I} + A_N^*A_N)^{-1}A_N^*(A_Nx_k - b_N).$$

Koristeći $A_N^*A_N = NA^*A$ i $A_N^*b_N = NA^*b$, dobijamo

$$\begin{aligned} x_{k+1} &= x_k - (\alpha_k\mathbf{I} + NA^*A)^{-1}NA^*(Ax_k - b) \\ &= x_k - \left(\frac{\alpha_k}{N}\mathbf{I} + A^*A\right)^{-1}A^*(Ax_k - b), \quad k = 0, 1, \ldots \end{aligned}$$

Tvrdjenje (2.1.2$'$) sleduje iz

$$X_{k+1} = X_k - (\alpha_k\mathbf{I} + A_N^*A_N)^{-1}A_N^*(A_NX_k - I_N), \quad k = 0, 1, \ldots$$

kao i $A_N^*A_N = NA^*A$, $A_N^*I_N = NA^*\mathbf{I} = NA^*$. □

2.2. Implementacija

Metod uveden u Teoremi 2.1.1 implementiran je sledećom funkcijom:

```
NewIter[a_, start_, alfa0_, brit_]:=
Block[{t=a,t0=start,a0=alfa0,iter=brit,m,n,tz,it=0,
        p1,p2,im,in},
    {m,n}=Dimensions[t];
    im=IdentityMatrix[m]; in=IdentityMatrix[n];
    tz=Conjugate[Transpose[t]];
    While[it<=iter,
           p1=Inverse[a0 in +tz.t];    p2=t.t0-im;
           t1=t0-p1.tz.p2;    t0=t1; it+=1; a0 =a0/2;
    ];
    t1
]
```

Metod iz Posledice 2.1.1 je implementiran kako sledi:

```
NewIter2[a_, start_, alfa0_, brit_]:=
Block[{t=a,t0=start,a0=alfa0,iter=brit,m,n,tz,it=0,
        p1,p2,im,in},
    {m,n}=Dimensions[t];
    im=IdentityMatrix[m]; in=IdentityMatrix[n];
```

```
    tz=Conjugate[Transpose[t]];
    While[it<=iter, (promeni uslov)
           p1=Inverse[a0 in +tz.t]; p2=t.t0-im;
           t1=t0-p1.tz.p2;        t0=t1; it+=1;
    ];
    t1
]
```

Sledi implementacija metoda koji je uveden u Teoremi 2.1.2.

```
NewIter1[a_, start_, alfa0_, brit_]:=
Block[{t=a,t0=start,a0=alfa0,iter=brit,m,n,tz,it=0,
        p1,p2,im,in},
    {m,n}=Dimensions[t];
    im=IdentityMatrix[m]; in=IdentityMatrix[n];
    tz=Conjugate[Transpose[t]];
    While[it<=iter,
           p1=Inverse[a0 in +tz.t];  p2=t.t0-im;
           t1=t0-p1.(tz.p2 + a0 t0); t0=t1; it+=1; a0=a0/2
    ];
    t1
]
```

Rezultat Teoreme 2.1.3 može se implementirati sledećim programom:

```
NewIterN[a_, start_, alfa0_, brit_]:=
Block[{t=a,t0=start,a0=alfa0,iter=brit,m,n,tz,it=0,
       p1,p2,im,in},
    {m,n}=Dimensions[t];
    im=IdentityMatrix[m]; in=IdentityMatrix[n];
    While[it<=iter,
           tz=Conjugate[Transpose[t]];
           p1=Inverse[a0 in +tz.t];    p2=t.t0-im;
           t1=t0-p1.tz.p2;    t0=t1; it+=1;
           im=Join[im,im];  t=Join[t,t];
    ];
    t1
]
```

Literatura

[1] V. Aleksić, V. Rakočević, *Approximate properties of the Moore-Penrose inverse*, VIII Conference on Applied Mathematics, Tivat (1993), 1–14.

[2] М. Базаара, К. Шетти, Нелинейное программирование, Теория и алгоритмы, Мир, Москва, 1982.

[3] A. Ben-Israel, *On matrices of index zero or one*, SIAM J. Appl. Math. **17** (1969), 1118–1121.

[4] A. Ben-Israel and T.N.E. Grevile, *Generalized Inverses: Theory and Applications*, Wiley-Interscience, New York, 1974.

[5] A. Ben-Israel, *A volume associated with $m \times n$ matrices*, Linear Algebra Appl. **167** (1992), 87–111.

[6] C.H. Bischof, A. Bouaricha, P.M. Khademi, J.J. Moré, *Computing Gradients in Large-scale Optimization Using Automatic Differentiation*, Technical Report, 1995.

[7] S.L. Campbell, C.D. Meyer, *Generalized Inverses of Linear Transformations*, Pitman, New York, 1979.

[8] D. Cvetković, M. Čangalović, Dj. Dugošija, V. Vujčić-Kovačević, S. Simić, J. Vuleta, *Kombinatorna optimizacija*, Društvo operacionih istraživača Jugoslavije – DOPIS, Beograd, 1996.

[9] J.E. Dennis, J.J. Moré, *Quasi-Newton methods, motivation and theory*, SIAM Rev. **19** (1977), 46–89.

[10] G.C. Godwin, P.V. Kabaila, T.S. Ng, *On the Optimization of Vector-Valued Performance Criteria*, IEEE Trans. Automatic Control **20** (1975).

[11] C.W. Groetch, *Generalized Inverses of Linear Operators*, Marcel Dekker, Inc. New York and Basel, 1977.

[12] L.W. Henessey, *Common* LISP, McGraw-Hill Book Company, 1989.

[13] D.M. Himmellblau, *Applied Nonlinear Programming*, McGraw-Hill Book Company, 1972.

[14] E. Hyvönen, J. Seppänen, Мир Лиспа, Mir, Москва, 1990.

[15] S.L.S. Jacoby, J.S. Kowalik, J.T. Pizzo, *Iterative Methods for Nonlinear Optimization Problems*, Prentice-Hall, Inc, Englewood, New Jersey, 1977.

[16] R. Kalaba, A. Tishler, *A computer program to minimize a function with many variables using computer evaluated exact higher-order derivates*, Appl. Math. Comput. **13** (1983), 143–172.

[17] W.J. Kammerer, M.Z. Nashed, *On the convergence of the conjugate gradient method for singular linear operator equations*, SIAM Rev. **9** (1972), 165–181.

[18] W.J. Kammerer, M.Z. Nashed, *Steepest descent for singular linear operators with nonclosed range*, Applicable Analysis **1** (1971), 143–159.

[19] E.H. Kaufman, D.J. Leeming, *CONMAX, Version 1.7*, Technical Report, 1996.

[20] V.V. Kovačević-Vujčić, *A view of interior point methods for linear programming*, YUJOR **5** (1995), 173–193.

[21] N. Krejić, Dj. Herceg, *Matematika i* MATHEMATICA, Računari u univerzitetskoj praksi, Novi Sad, 1993.

[22] S. Krčevinac, M. Ćupić, J. Petrić, I. Nikolić, *Algoritmi i programi iz operacionih istraživanja*, Naučna knjiga, Beograd, 1983.

[23] Ю.Н. Кузнецов, V.I. Kuzubov, А.Б. Болошенко, Математическое программирование, Высшая школа, Moskva, 1968.

[24] C. Lawrence, J.L. Zhou, A.L. Tits, *User's Guide for CFSQP Version 2.5: A C Code for Solving Constrained Nonlinear Optimization Problems, Generating Iterates Satisfying All Inequality Constrains*, Technical Report, 1997.

[25] G.V. Milovanović, *Numerička analiza I*, Naučna knjiga, Beograd, 1991.

[26] G.V. Milovanović, *Numerička analiza II*, Naučna knjiga, Beograd, 1991.

[27] G.V. Milovanović, D.S. Mitrinović, Th.M. Rassias, *Topics in Polynomials: Extremal Problems, Inequalities, Zeros*, World Scientific, Singapore - New Jersey - London - Hong Kong, 1994.

[28] G.V. Milovanović, S. Wrigge, *Least square approximation with constraints*, Math. Comp. **46** (1986), 551–565.

[29] S.K. Mitra, R.C. Rao, *Extensions of a duality theorem concerning g-inverses of matrices*, The Indian Journal of Statistics **37** (1975), 439–445.

[30] S. Opricović, *Optimizacija sistema*, Nauka, Beograd, 1992.

[31] T.S. Parker, L.O. Chua, *INSITE – a software toolkit for the analysis of nonlinear dynamic systems*, Proceedings of the IEEE **75** (1987), 1081–1089.

[32] R. Penrose, *A generalized inverse for matrices*, Proc. Cambridge Phil. Soc. **51** (1955), 406–413.

[33] R. Penrose, *On best approximate solution of linear matrix equations*, Proc. Camb.Phil. Soc. **52** (1956), 17–19.

[34] J. Petrić, *Operaciona istraživanja, Naučna knjiga, Beograd, 1989.*

[35] M.J.D. Powel, *A survey of numerical methods for unconstrained optimization*, SIAM Rev. **12** (1970), 79–97.

[36] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, New York-Melbourne-Sydney, 1990.

[37] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes*, Cambridge University Press, New York, Melbourne, Sydney, 1986.

[38] L.D. Pyle, *The weighted generalized inverse in nonlinear programming-active set selection using a variable-metric generalization of the simplex algorithm*, International symposium on extremal methods and systems analysis, Lecture Notes in Economics and Mathematical Systems **174** (1977), 197–231.

[39] L.D. Pyle and R.E. Cline, *The generalized inverse linear programming-interior gradient projection methods*, SIAM J. Appl. Math. **24** (1973), 511–534.

[40] L.D. Pyle, *The generalized inverse linear programming. Basic structure*, SIAM J. Appl. Math. **22 No 3** (1972), 335–355.

[41] S. Rančić, *Implementacija nekih metoda matematičkog programiranja u LISPu*, Magistarska teza, Univerzitet u Nišu, Filozofski fakultet, 1997.

[42] C.R. Rao, S.K. Mitra, *Generalized Inverse of Matrices and its Applications*, John Wiley & Sons, Inc, New York, London, Sydney, Toronto, 1971.

[43] J.K. Reid, *The use of conjugate gradients for system of linear equations possesing "property A"*, SIAM Rev. **9** (1972), 325–332.

[44] W.S. Richard, *LISP, Lore and Logic*, Springer-Verlag, 1990.

[45] S. Robert, *Algorithms in* C, Addison-Wesley, New York, 1990.

[46] J.B. Rosen, *Minimum and basic solutions to singular linear systems*, SIAM **12** (1964), 156–162.

[47] J.D. Smith, *An Introduction to Scheme*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[48] P.S. Stanimirović, S. Rančić, *Unidimensional search optimization in LISP*, Proceedings of the II Mathematical Conference, 1996, Priština, 253–262.

[49] P.S. Stanimirović, S. Rančić, *Unconstrained optimization in LISP*, Proceedings of the XI Conference on Applied Mathematics, 1996, Budva, 355–362.

[50] P.S. Stanimirović, S. Rančić, *First-order gradient optimization methods in LISP*, Korean J. Comput. Appl. Math. **5** (1998), 611–626.

[51] P.S. Stanimirović, S. Rančić, *Second order optimization methods in LISP*, YUJOR **9** (1999), 113–127.

[52] P.S. Stanimirović, S. Rančić, *Implementation of penalty function methods in LISP*, Acta Math. Inform. Univ. Ostraviensis **7** (1999), 119–141.

[53] P.S. Stanimirović, S. Rančić, *Symbolic implementation of lexicographic multicriteria program*, FILOMAT **12** (1998), 1–8.

[54] P.S. Stanimirović, I.B. Stanković, *Symbolic implementation of simplex method*, XIII Conference on Applied Mathematics, Budva (1998).

[55] P.S. Stanimirović, M.B. Tasić, M. Ristić, *Symbolic implementation of the Hooke-Jeeves method*, YUJOR **9** (1999), 285–301.

[56] P.S. Stanimirović, S. Rančić, M. Tasić, *Repetitive applications of functions as arguments in programming languages*, Proceedings of VIII Conference on Logic and Computer Science, LIRA '97, 1997 Novi Sad, 231–238.

[57] P.S. Stanimirović, *Computing minimum and basic solutions of linear systems using the Hyper-power method*, Studia Sci. Math. Hungar. **35** (1999), 175–184.

[58] N. Stojković, P.S. Stanimirović, *Two direct methods in linear programming*, European J. Operational Research **131** (2001), 417–439.

[59] N. Stojković and P.S. Stanimirović, *Initial point in primal-dual interior point method*, Facta Univ. Ser. Mech. Rob. **3** (2001), 219–222.

[60] С. Стоянов , Методы и Алгоритми за Оптимизация, Държавно издателство, Техника, София, 1990.

[61] G.J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Massachusetts, 1985.

[62] K. Tanabe, *Conjugate-gradient method for computing the Moore-Penrose inverse and rank of a matrix*, J. Optim. Theory Appl. **22** (1977), 1–23.

[63] M.B. Tasić, *Implementacija nekih metoda za izračunavanje ekstremnih vrednosti i generalisanih inverza*, Magistarska teza, Univerzitet u Nišu, Niš, 2000.

[64] O. Todorović, *Operaciona istraživanja, Prosveta, Niš, 1992.*

[65] M.R. Trummer, *A method for solving ill-possed linear operator equation*, SIAM J. Numer. Anal. **21** (1984), 729–737.

[66] S. Vukadinović, S. Cvejić, *Matematičko programiranje, Univerzitet u Prištini, 1996.*

[67] T.M. Whitney, R.K. Meany, *Two algorithms related to the method of steepest descent*, SIAM Rev. **4** (1967), 109–118.

[68] R. Wilensky, *Common LISPcraft*, Norton, New York, 1986.

[69] S. Wolfram, *Mathematica: a System for Doing Mathematics by Computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[70] S. Wolfram, *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[71] J.L. Zhou, A.L. Tits, C.T. Lawrence, *User's Guide for CFSQP Version 3.7: A FORTRAN Code for Solving Constrained Nonlinear Optimization Problems, Generating Iterates Satisfying All Inequality and Linear Constrains*, Technical Report, 1997.

[72] J.L. Zhou, A.L. Tits, C.T. Lawrence, *User's Guide for CFSQP Version 3.7: A C Code for Solving Constrained Nonlinear Optimization Problems, Generating Iterates Satisfying All Inequality and Linear Constrains*, Technical Report, 1997.

[73] G. Zielke, *Some remarks on matrix norms, condition numbers, and error estimates for linear equations*, Linear Algebra Appl. **110** (1988), 29–41.

[74] S. Zlobec, *Nelinearno programiranje*, Naučna knjiga, Beograd, 1989.

Indeks