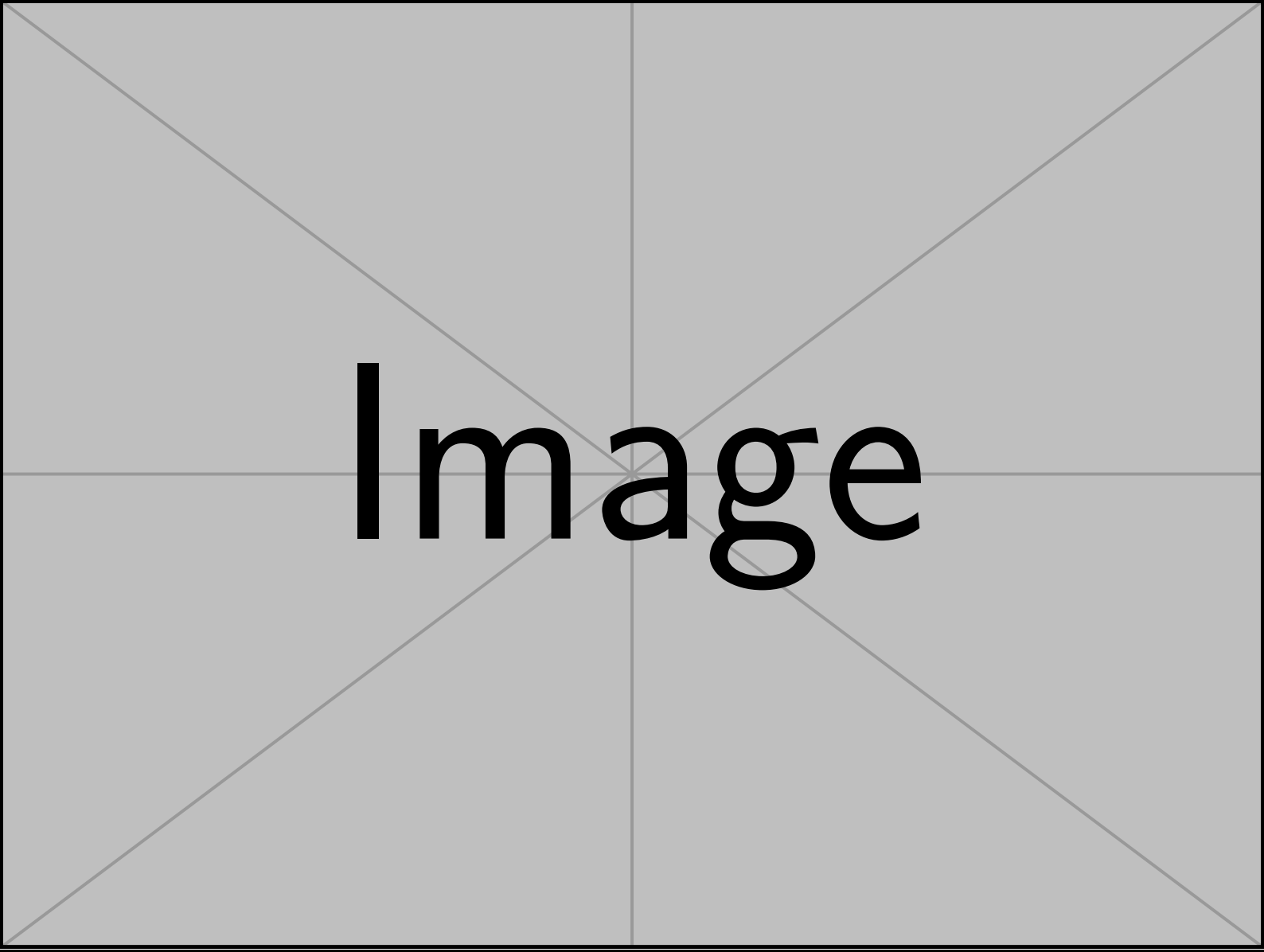


Postupci numeričke optimizacije

Autor: Marko Đurasević

Domagoj Jakobović

Sadržaj

1 Uvod **2**

1.1 Određivanje minimuma funkcije 5

1.2 Svojstva kvadratne funkcije 9

2 Optimizacija funkcija jedne varijable **10**

2.1 Postupak pronalaženja početnog unimodalnog intervala 10

2.2 Redukcija unimodalnog intervala 12

2.3 Postupak zlatnog reza 14

2.4 Fibonaccijeva metoda 19

2.5 Kvadratna interpolacija 22

2.6 Newtonova metoda 28

3 Optimizacija funkcija više varijabli bez uporabe derivacija **29**

3.1 Koordinatno pretraživanje 31

3.2 Powellova metoda konjugiranih smjerova 34

3.3 Rosenbrockova metoda 39

3.4 Postupak Hooke-Jeeves 42

3.5 Simpleks postupak po Nelderu i Meadu 46

4 Optimiranje uz uporabu gradijenta **53**

4.1 Gradijentni spust 54

4.2 Konjugirani gradijenti 58

4.3 Newtonova metoda 61

4.4 Kvazi-Newtonove metode 65

4.5 Metoda područja povjerenja 70

5 Metode za probleme najmanjih kvadrata **72**

5.1 Gauss-Newtonova metoda 72

5.2 Metoda po Levenberg–Marquardtu 74

6 Optimiranje uz ograničenja **76**

6.1 Traženje unutarnje točke 80

6.2 Metode kazne i barijere 83

6.3 Boxov postupak 87

Napomena

Inačica teksta: siječanj 2025.

Ovaj tekst je u izradi. Pojedini dijelovi teksta još uvijek nedostaju te tekst nije gramatički niti pravopisno ispravljen. Osim toga, izvodi, formule i algoritmi nisu provjereni te su moguće greške u njima. Stoga se preporučuje ovu knjigu trenutno koristiti samo kao dodatak ostalim materijalima. Ako postoji određena nekonzistentnost u materijalima, onda se preporučuje držati se onoga što piše u postojećim materijalima, ili se oko nejasnoća posavjetovati s nastavnicima. Knjiga će se inkrementalno unapređivati dodavanjem novih primjera, ali i ispravljanjem pogrešaka. Eventualne prijedloge i kritike možete slobodno javiti na marko.durasevic@fer.hr odnosno domagoj.jakobovic@fer.hr

POGLAVLJE 1

Uvod

U gotovo svakoj ljudskoj djelatnosti javlja se implicitna težnja za optimizacijom. Od svakodnevnih aktivnosti kao što su planiranje putovanja do složenih poslovnih strategija, ljudi su neprestano u potrazi za najučinkovitijim pristupom ili najboljim rješenjem. Ekonomski sustavi teže optimirati raspodjelu sredstava kako bi maksimizirali dobit, dok pojedinci teže optimirati svoje vrijeme kako bi uz što manji uloženi trud postigli željene rezultate. U mnogim znanstvenim disciplinama, od inženjerstva do biologije, optimizacija igra ključnu ulogu, bilo da se radi o minimizaciji troškova, maksimizaciji učinkovitosti ili pronalaženju optimalnih putanja. Stoga se može tvrditi da je sposobnost optimizacije temeljni element ljudske prirode kojega nalazimo u svim postupcima istraživanja.

Optimizacijski problemi javljaju se u različitim oblicima i veličinama, od pronalaženja najučinkovitijeg rasporeda sredstava u proizvodnji do minimizacije troškova pri transportu ili maksimizacije dobiti u financijskim portfeljima. Ovi problemi često uključuju složene kriterije, netrivijalna ograničenja i veliki broj varijabli, čineći njihovo rješavanje izazovnim zadatkom. Proces optimiranja predstavlja traženje što boljeg rješenja zadanog problema s obzirom na definirane kriterije, uz uvjet zadovoljavanja zadanih ograničenja.

Kako bi optimizacija bila moguća, potrebno je definirati *kriterij optimizacije*, tj. numeričku mjeru kvalitete procesa kojega optimiramo. Iznos kriterija ovisit će o odabranim svojstvima procesa, koje nazivamo *varijablama* ili *nepoznanicama*. Odabirom vrijednosti varijabli, cilj optimizacije je uz što manje utrošenih sredstava - npr. vremena procesora - pronaći skup vrijednosti varijabli koji daje najbolji mogući iznos kriterija optimizacije. Ukoliko se optimizacija provodi simuliranjem promatranog procesa na računalu kako bismo odredili iznos optimiranog kriterija, govorimo o numeričkoj optimizaciji.

Algoritmi numeričke optimizacije predstavljaju moćan alat koji omogućuje pronalaženje najboljih - ili dovoljno dobrih - rješenja u prostoru svih mogućih rješenja nekog optimizacijskog problema. Nažalost, ne postoji univerzalni optimizacijski algoritam. Ovisno o konkretnom problemu, samo će neki optimizacijski postupci biti primjenjivi na dani problem, a neki od postupaka mogu biti značajno učinkovitiji od drugih. Odabir odgovarajuće metode često ovisi o prirodi funkcije koja se optimira i karakteristikama problema. U ovom poglavlju bit će stoga govora o različitim algoritmima numeričke optimizacije, kao i o vrstama problema na koje se postojeći algoritmi mogu učinkovito primijeniti.

1.0.1 Problem optimiranja

U svim primjerima koristit ćemo sljedeće pretpostavljene oznake:

- $\boldsymbol{x}$ predstavlja vektor varijabli, odnosno nepoznanica ili parametara;
- f predstavlja *funkciju cilja*, kojom definiramo željeni kriterij optimitacije.

Funkcija cilja je skalarna funkcija jednog ili više parametara x_i, ovisno o tome koliko varijabli se promatra tijekom postupka optimiranja. Postupak optimiranja matematički se izražava kao traženje

minimuma ili maksimuma funkcije cilja. Kako se problem traženja maksimuma uvijek može svesti na traženje minimuma negativne funkcije, u nastavku će se pretpostavljati postupak traženja minimuma funkcije cilja.

Optimizacijski problem tako može biti definiran kao

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{1.1}$$

gdje je $\boldsymbol{x} \in \mathbb{R}^n$ n-dimenzijski vektor, uz $n \geq 1$:

$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix},$$

a $f : \mathbb{R}^n \to \mathbb{R}$ je skalarna funkcija cilja.

U idealnom slučaju, rezultat optimizacije je pronalazak najboljeg mogućeg rješenja, odnosno takvog $\boldsymbol{x}^*$ za koji vrijedi $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^n$. Rješenje koje daje najmanju moguću vrijednost funkcije na cijelom području definicije naziva se *globalni optimum*. Nažalost, za funkciju cilja općenitog oblika nije moguće sa sigurnošću odrediti položaj i vrijednost globalnog optimuma, budući da algoritmi optimizacije mogu u konačnom vremenu pretražiti samo maleni dio prostora rješenja. Algoritmi optimizacije su stoga većinom osmišljeni tako da pronalaze točku koja predstavlja *lokalni optimum*. Za neku točku kažemo da je lokalni optimum ako u njenoj okolici ne postoji točka koja daje manju (tj. bolju) vrijednost funkcije cilja. Drugim riječima, točka $\boldsymbol{x}^*$ je lokalni minumum ako postoji susjedstvo S od $\boldsymbol{x}^*$ takvo da je $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ za svaki $\boldsymbol{x} \in S$.

Funkcije cilja koje imaju samo jedan minimum u području definicije nazivaju se *unimodalne funkcije*; unimodalna funkcija ima samo jedan lokalni minumum, koji je ujedno i globalni minimum. Višemodalne funkcije, s druge strane, imaju više od jednog lokalnog minimuma, od kojih je jedan ujedno i globalni.

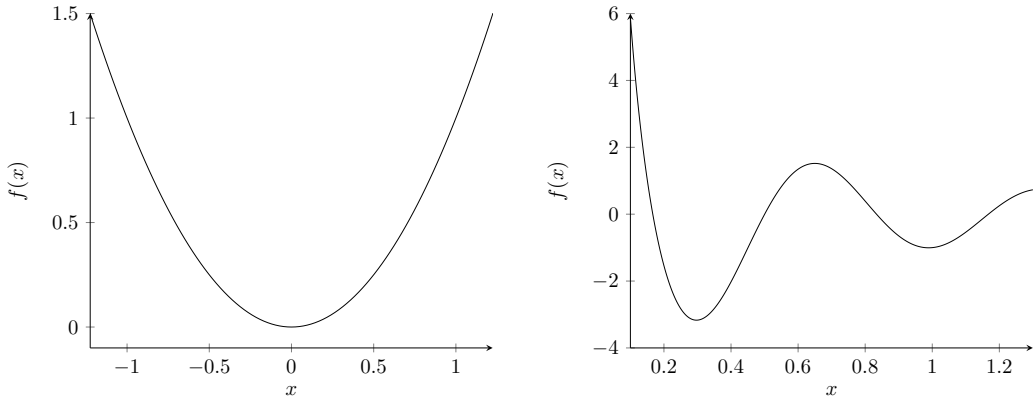
Slika 1.1 prikazuje jednu unimodalnu i jednu višemodalnu funkciju. Na primjeru unimodalne funkcije s minimumom u $x = 0$ može se vidjeti kako vrijednost funkcije u svim točkama oko minumuma monotono raste. S druge strane, kod višemodalne funkcije kretanje vrijednosti funkcije cilja nije monotono pa dobivamo više lokalnih minimuma (i posljedično, lokalnih maksimuma). Na slici 1.2 dan je primjer funkcije s označenim minimumima i maksimumima koja u promatranom području ima jedan globalni i dva lokalna minimuma. Osim toga, funkcija ima i dva globalna maksimuma, jer obje točke imaju jednaku funkcijsku vrijednost, kao i dva lokalna maksimuma.

Postojanje višestrukih minimuma unosi dodatne teškoće u problem pronalaženja minimuma. Naime, ne postoje postupci koji pouzdano vode do globalnog minimuma, već je se za određivanje globalnog minimuma najčešće koriste postupci za pronalaženje lokalnog minimuma koji se višestruko ponavljaju s različitim početnim uvjetima. U daljem tekstu ovoga poglavlja će se, ako nije drugačije rečeno, razmatrati samo postupci koji vode do lokalnog minimuma, odnosno pretpostavit ćemo određivanje minimuma unimodalnih funkcija.

Osim postojanja samo jednog ili više optimuma, još jedno važno svojstvo funkcije cilja je i *konveksnost*. Funkcija je konveksna ako vrijedi

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2), \quad \forall x_1, x_2 \in X, \forall t \in [0,1], \tag{1.2}$$

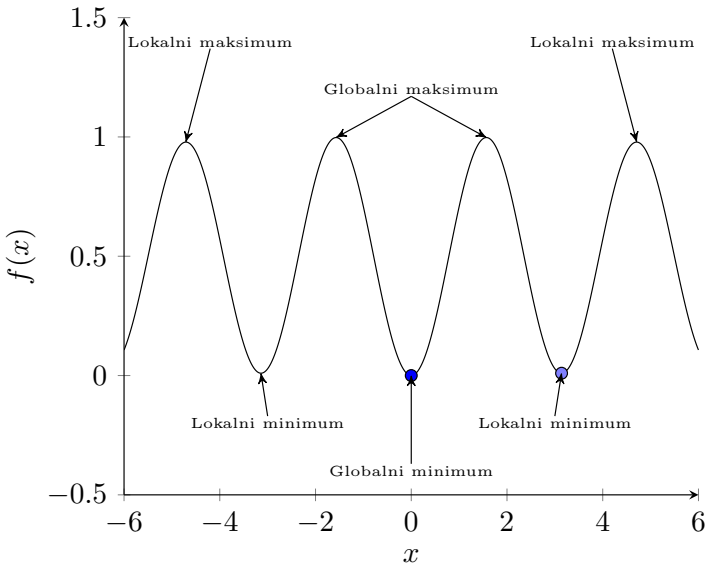
gdje X predstavlja domenu funkcije f. Prethodni izraz može se ilustrirati na sljedeći način: ako je funkcija konveksna, možemo odabrati bilo koje dvije točke x_1 i x_2 iz domene, te prikazati dužinu koja spaja te dvije točke u kodomeni funkcije. Za bilo koje dvije krajnje točke, u svim točkama dužine vrijednost funkcije



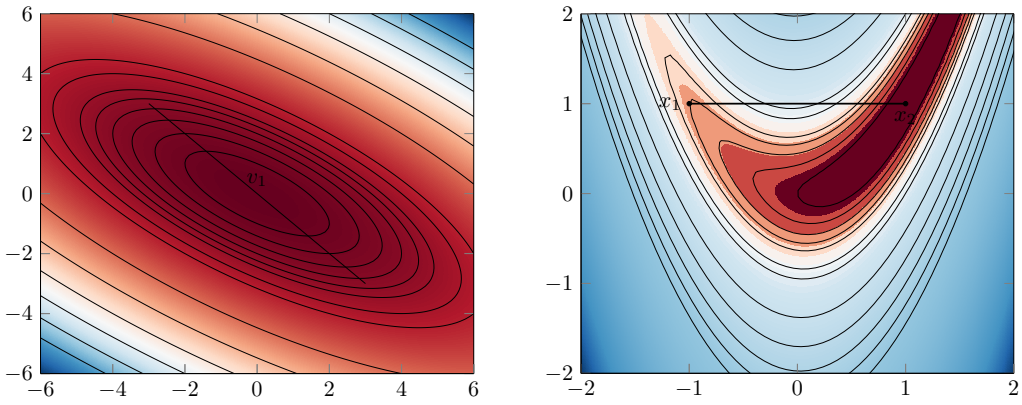
(a) Unimodalna funkcija

(b) Višemodalna funkcija

Slika 1.1: Primjer unimodalne i višemodalne funkcije



Slika 1.2: Primjer funkcije s lokalnim i globalnim minimumima



(a) Konveksna funkcija dvije varijable

(b) Nekonveksna funkcija dvije varijable

Slika 1.3: Primjer konveksne i nekonveksne funkcije

cilja će biti manja ili jednaka interpoliranim vrijednostima na toj dužini.

Slika 1.3 daje primjer konveksne i nekonveksne funkcije dvije varijable. Kod konveksne funkcije na slici 1.3a možemo vidjeti kako za bilo koju dužinu koji bi provukli između proizvoljno odabranih točaka, funkcijska vrijednost točaka na tom pravcu nikada nije veća od vrijednosti dobivene interpolacijom između početne i završne točke dužine. S druge strane, kod nekonveksne funkcije na slici 1.3b moguće je konstruirati dužinu za koju će vrijednost funkcije cilja u točkama koje se nalaze na njoj imati veću vrijednost nego vrijednost točke na dužini. Konveksne funkcije se često razmatraju u optimizaciji iz razloga što su one ujedno i unimodalne. Osim toga, za bilo koji pravac koji konstruiramo u području domene konveksne funkcije, vrijednosti funkcije cilja na tom pravcu uvijek čine unimodalni skup, što može jamčiti pronalaženje lokalnog minimuma.

1.1 Određivanje minimuma funkcije

Tražimo li minimum funkcije, kako možemo znati predstavlja li neka točka minimum zadane funkcije cilja? Kao prvo, bez dodatnih informacija o obliku funkcije - primjerice bez pretpostavke unimodalnosti - nije moguće jednoznačno odrediti radi li se zaista o *globalnom* minimumu. Stoga će u nastavku biti govora isključivo o lokalnom minimumu i njegovim svojstvima. Za odgovor na pitanje predstavlja li neka točka *lokalni* minimum funkcije, intuitivni pristup uključivao bi promatranje vrijednosti funkcije cilja u svim točkama u okolici promatrane točke, te provjeru postiže li se uvijek vrijednost veća nego u pretpostavljenom minimumu. Naravno, ovakav pristup u praksi nije izvediv; no uz pretpostavku neprekidivosti funkcije cilja te time i postojanje njene derivacije, moguće je identificirati nužna i dovoljna svojstva za lokalni minimum.

Derivacija funkcije pokazuje kako se mijenja vrijednost funkcije u ovisnosti o promjeni argumenta; za funkcije više varijabli, ova promjena se može predstaviti za svaku promatranu varijablu u obliku parcijalne derivacije funkcije po toj varijabli. Kombiniranjem svih parcijalnih derivacija neke funkcije po

svim varijablama, dobivamo *gradijent funkcije* koji je definiran kao sljedeći vektor:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_{n-1}} \end{bmatrix}, \tag{1.3}$$

gdje x_i predstavlja i-tu varijablu. Gradijent funkcije se može interpretirati kao n-dimenzijski vektor koji pokazuje smjer u kojemu iznos funkcije najbrže raste. Nastavimo li sve prve parcijalne derivacije još jednom derivirati po svakoj varijabli, možemo dobiti i druge parcijalne derivacije. Budući da ovdje postoji $n \times n$ kombinacija, druge derivacije prikazuju se u obliku *Hesseove matrice* H koja je definirana kao

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_0^2} & \frac{\partial^2 f}{\partial x_0 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_{n-1}} \\ \frac{\partial^2 f}{\partial x_1 \partial x_0} & \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_{n-1} \partial x_0} & \frac{\partial^2 f}{\partial x_{n-1} \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_{n-1}^2} \end{bmatrix}. \tag{1.4}$$

Vrijednosti derivacija zadane funkcije nije uvijek moguće analitički odrediti, pa se u tom slučaju koristimo pomoćnim aproksimacijama, no o njima će biti govora kasnije. Jednom kada imamo uvid u derivacije funkcije cilja, vrijednost funkcije jedne varijable u bilo kojoj točki x možemo odrediti uz pomoć razvoja u Taylorov red:

$$f(x) = f(a) + \frac{f^{'}(a)(x-a)}{1!} + \frac{f^{''}(a)(x-a)^2}{2!} + \frac{f^{'''}(a)(x-a)^3}{3!} + \ldots, \tag{1.5}$$

gdje a predstavlja točku oko razvijamo red, a x predstavlja točku u kojoj se računa vrijednost funkcije. U slučaju funkcije više varijabli, aproksimacija Taylorovim redom poprima sljedeći oblik:

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + (\boldsymbol{x}-\boldsymbol{a})^T \nabla f(\boldsymbol{a}) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{a})^T H(\boldsymbol{x})(\boldsymbol{x}-\boldsymbol{a}) + \cdots. \tag{1.6}$$

Uz korištenje samo prva tri člana razvoja u red, ovakav oblik naziva se i kvadratna aproksimacija.

Jedan način kako za neku točku odrediti predstavlja li lokalni minimum funkcije moguće je dobiti razvojem funkcije u red oko točke za koju pretpostavljamo da je minumum, $\boldsymbol{x}_{min}$. Točka za koju pri tome izražavamo vrijednost funkcije je $\boldsymbol{x}$, a vektor razlike između tih točaka možemo označiti s $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_{min}$. Tada razvoj u red oko točke minimuma možemo prikazati kao

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_{min}) + \nabla f(\boldsymbol{x}_{min})\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T \boldsymbol{H} \Delta\boldsymbol{x}. \tag{1.7}$$

Ako je $\boldsymbol{x}_{min}$ lokalni minimum, tada vrijednost funkcije u svim točkama $\boldsymbol{x}$ u okolici minimuma mora biti veća, odnosno vrijedit će $f(\boldsymbol{x}) \geq f(\boldsymbol{x}_{min})$ za svaki $\boldsymbol{x}$ u okolici. To je moguće jedino ako je zbroj dva desna člana u gornjem izrazu uvijek pozitivan, neovisno o pomaku od minimuma $\Delta\boldsymbol{x}$.

Prvi od ta dva člana predstavlja skalarni umnožak vektora gradijenta i pomaka od minumuma. Elementi vektora gradijenta su konstantni jer je gradijent određen u točki minimuma $\boldsymbol{x}_{min}$, dok elementi vektora pomaka mogu poprimiti vrijednosti bilo kojeg predznaka, ovisno o smjeru pomaka. Budući da vrijednost funkcije mora rasti u svim smjerovima pomaka, jedina mogućnost po kojoj ovaj član neće narušiti taj uvjet je da je vektor gradijenta u točki minimuma jednak nul-vektoru:

$$\nabla f(\boldsymbol{x}_{min}) = \boldsymbol{0}. \tag{1.8}$$

Na taj način, vrijednost drugog člana razvoja u red ostaje nula bez obzira na odmak od točke minimuma. Prethodni uvjet smatra se *nužnim* uvjetom za postojanje lokalnog minimuma u točki $\boldsymbol{x}_{min}$, no to nije i dovoljan uvjet za minimum.

Preostali član razvoja u red trebao bi, uz pretpostavku o lokalnom minimumu, poprimiti samo pozitivne iznose za bilo koji $\Delta\boldsymbol{x}$. Može se pokazati da predznak izraza $\Delta\boldsymbol{x}^T H \Delta\boldsymbol{x}$ neće ovisiti o vektoru $\Delta\boldsymbol{x}$ ako matrica H zadovoljava neka dodatna svojstva. Konkretno, ako je matrica H *pozitivno definitna*, tada će za bilo koji pomak predznak tog člana uvijek biti pozitivan, odnosno vrijedi

$$\Delta\boldsymbol{x}^T H \Delta\boldsymbol{x} > 0, \forall \Delta\boldsymbol{x}. \tag{1.9}$$

Prema tome, ako je u nekoj točki gradijent nul-vektor, a Hesseova matrica je pozitivno definitna, tada ta točka predstavlja lokalni minimum funkcije. No što ako H nije pozitivno definitna? Osim ovog svojstva, neka matrica može biti i *negativno definitna*, u kojem slučaju vrijedi

$$\Delta\boldsymbol{x}^T H \Delta\boldsymbol{x} < 0, \forall \Delta\boldsymbol{x}. \tag{1.10}$$

Ako je Hesseova matrica negativno definitna, svaki pomak od početne točke izazvat će *smanjenje* funkcije cilja, što zapravo pokazuje da je promatrana točka lokalni *maksimum*. Konačno, matrica ne mora biti niti pozitivno niti negativno definitna; u tom slučaju matrica je *indefinitna*, što znači da vrijednost izraza $\Delta\boldsymbol{x}^T H \Delta\boldsymbol{x}$ može biti i pozitivna i negativna, ovisno o pomaku $\Delta\boldsymbol{x}$. U tom slučaju, promatrana točka (za koju i dalje pretpostavljamo da vrijedi nužni uvjet 1.8) nije niti maksimum, niti minimum, već *sedlo*, jer će pomak u nekim smjerovima značiti povećanje, a pomak u drugim smjerovima smanjenje iznosa funkcije cilja.

Sve tri navedene vrste točaka, lokalni minimum, maksimum ili sedlo, nazivaju se i *stacionarne točke* funkcije. Želimo li provjeriti je li neka točka lokalni minimum funkcije, nužni uvjet za to (kao i za druge dvije vrste stacionarnih točaka) je da je gradijent funkcije u toj točki jednak nul-vektoru. Za dovoljan uvjet potrebno je provjeriti i Hesseovu matricu u toj točki; ako je ona pozitivno ili negativno definitna, riječ je o ekstremu: minimumu ili maksimumu. U slučaju indefinitne matrice, promatrana točka je sedlo funkcije.

Naposljetku, ako je matrica drugih derivacija $\boldsymbol{H}$ pozitivno definitna na cijelom području definicije funkcije, tada postoji samo jedan minimum, a osim što je unimodalna, funkcija je ujedno i konveksna.

Primjer 1.1

Zadana je funkcija $f(x) = 2x_1^3 + 6x_1x_2^2 - 3x_2^3 - 150x_1$. Potrebno je odrediti sve stacionarne točke zadane funkcije.

Prvo je potrebno izračunati parcijalne derivacije funkcije po varijablama x_1 i x_2 i izjednačiti ih s 0.

$$\frac{\partial f(x)}{\partial x_1} = 6x_1^2 + 6y^2 - 150 = 0 \tag{1.11}$$

$$\frac{\partial f(x)}{\partial x_2} = 12xy - 9y^2 = 0 \tag{1.12}$$

Nakon što sredimo izraze dobivamo

$$x_1^2 + x_2^2 = 25 \tag{1.13}$$

$$x_2(4x_1 - 3x_2) = 0. \tag{1.14}$$

Iz drugog izraza možemo vidjeti da je prvo rješenje ovog nelinearnog sustava $x_2 = 0$, što kada uvrstimo u prvi izraz dobivamo

$$x_1^2 = 25 \tag{1.15}$$

$$x_1 = \pm 5 \tag{1.16}$$

Prva dva rješenja su stoga jednaka $T_1 = (5, 0)$ i $T_2 = (-5, 0)$. S druge strane ako $x_2 \neq 0$ onda izraz u zagradi mora biti jednak nuli što daje

$$x_1 = \frac{3}{4}x_2 \tag{1.17}$$

Kada se taj izraz uvrsti u prvu jednadžbu dobiva se

$$\frac{9}{16}x_2 + x_2 = 25 \tag{1.18}$$

što znači da je $x_2 = \pm 4$. Kada se te vrijednosti uvrste u izraz 1.17 dobiju se točke $T_3 = (3, 4)$ i $T_4 = (-3, -4)$. Za prethodno izračunate točke potrebno je odrediti predstavljaju li one minimum, maksimum ili točku sedla. To možemo napraviti na način da izračunamo Hesseovu matricu za funkciju f

$$H = \begin{bmatrix} 12x_1 & 12x_2 \\ 12x_2 & 12x_1 - 18x_2 \end{bmatrix} \tag{1.19}$$

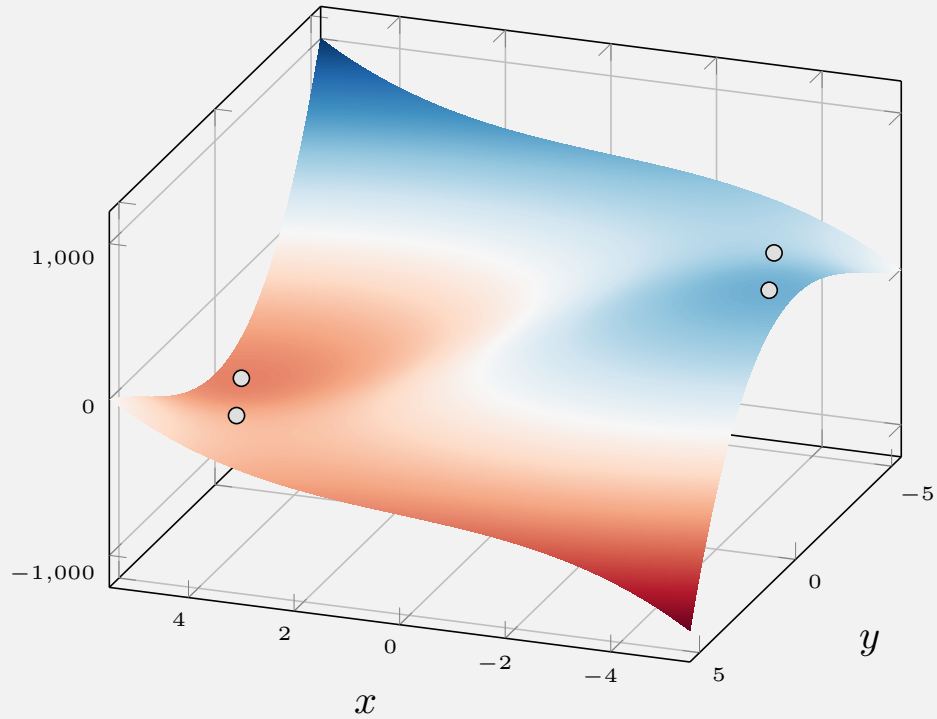
Kada se u Hesseovu matricu uvrste četiri prethodno dobivene točke dobiju se iduće četiri matrice

$$H_1 = \begin{bmatrix} 60 & 0 \\ 0 & 60 \end{bmatrix}, \quad H_2 = \begin{bmatrix} -60 & 0 \\ 0 & -60 \end{bmatrix}, \quad H_3 = \begin{bmatrix} 36 & 48 \\ 48 & -36 \end{bmatrix}, \quad H_4 = \begin{bmatrix} -36 & -48 \\ -48 & 36 \end{bmatrix} \tag{1.20}$$

Sada je potrebno za svaku od prethodno navedenih matrica provjeriti njenu definitnost. Definitnost matrice može se provjeriti na različite načine. Jedna mogućnost jest odrediti njene svojstvene vrijednosti i onda na temelju njih donijeti zaključak. Naime ako su sve svojstvene vrijednosti pozitivne onda je matrica pozitivno definitna, ako su negativne onda je negativno definitna, a inače je indefinitna. No postoji još jedan jednostavniji način kako provjeriti definitnost matrice, a to je svođenje matrice na gornju trokutastu matricu te se provjera svih elemenata na dijagonali. Ako su svi dijagonalni elementi pozitivni, tada je i sama matrica pozitivno defininta; ako su negativni, onda je negativno definitna, inače je indefinitna. Matrice H_1 i H_2 već i jesu u dijagonalnom obliku. Kod matrice H_1 možemo vidjeti da su svi elementi na dijagonali pozitivni, stoga je ona i pozitivno definitna. Zbog toga, točka T_1 predstavlja lokalni minimum funkcije f. S druge strane, matrica H_2 ima negativne vrijednosti na dijagonali što znači da je matrica negativno definitna. Dakle, točka T_2 predstavlja lokalni maksimum funkcije f. Preostale dvije matrice nisu u tom obliku, stoga ih je potrebno transformacijama nad retcima dovesti u taj oblik te se dobiva

$$H_3 = \begin{bmatrix} 36 & 48 \\ 0 & -100 \end{bmatrix}, \quad H_4 = \begin{bmatrix} -36 & -48 \\ 0 & 100 \end{bmatrix} \tag{1.21}$$

Iz prethodnih dviju matrica vidi se da su elementi na dijagonalama suprotnih predznaka. Zbog toga možemo zaključiti kako se radi o indefinitnim matricama te kako točke T_3 i T_4 predstavljaju sedla funkcije f. Slika 1.4 prikazuje primjer ove funkcije s označenim stacionarnim točkama.



Slika 1.4: Prikaz funkcije $f(x) = 2x_1^3 + 6x_1x_2^2 - 3x_2^3 - 150x_1$ ♣

1.2 Svojstva kvadratne funkcije

Tijekom razmatranja optimizacijskih problema korisno je pobliže promotriti *kvadratne funkcije*. Kvadratne su funkcije konveksne, pa se njihovi ekstremi mogu jednoznačno odrediti, a mnoge funkcije cilja ponašaju se *približno* kvadratno u okolici lokalnog minimuma. Upravo zbog ove pretpostavke o sličnosti, mnogi postupci optimizacije temelje se na različitim svojstvima kvadratih funkcija te su bez obzira na to primjenjivi i na funkcije koje nisu kvadratne. Kvadratne funkcije općenito zapisujemo kao

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c, \tag{1.22}$$

pri čemu je $\boldsymbol{A}$ simetrična matrica dimenzija $n \times n$, $\boldsymbol{b}$ vektor dimenzija $n \times 1$, a c je skalar. Iz ovog zapisa iznimno je lako odrediti gradijent kvadratne funkcije

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}. \tag{1.23}$$

S druge strane, Hesseova matrica jednaka je

$$\boldsymbol{H} = \boldsymbol{A}. \tag{1.24}$$

Iz nužnog uvjeta o postojanju minimuma $\nabla f(\boldsymbol{x}_{min}) = \boldsymbol{0}$ za kvadratne funkcije slijedi

$$\boldsymbol{x}_m = -\boldsymbol{A}^{-1}\boldsymbol{b}, \tag{1.25}$$

odnosno ako je $\boldsymbol{A}$ pozitivno definitna, tada točka $\boldsymbol{x}_m$ predstavlja minimum kvadratne funkcije, a maksimum ako je matrica negativno definitna. Prema tome, određivanje ekstrema kvadratne funkcije svodi se na rješavanje linearnog sustava

$$\boldsymbol{A}\boldsymbol{x}_m = -\boldsymbol{b}. \tag{1.26}$$

POGLAVLJE 2

Optimizacija funkcija jedne varijable

Iako u stvarnosti rijetko nailazimo na optimizacijske probleme koji ovise samo o jednoj varijabli, u nastavku ćemo prve primjere postupaka optimizacije pokazati upravo na takvim problemima. Jedan razlog je jednostavnost ovih problema te ilustracija uvjeta koji se mogu odnositi i na funkcije cilja više varijabli. Drugi razlog je taj što se postupci optimiranja funkcija jedne varijable često koriste kao osnovni dio složenijih postupaka za optimiranje više varijabli, o čemu će biti riječi u sljedećem poglavlju.

Kako se služimo numeričkim metodama, rješenje koje dobivamo njihovom primjenom nikada nije analitički točno, već uvijek nosi i određeni iznos tolerancije. Iz tog se razloga rješenje optimiranja funkcije jedne varijable najčešće ne predstavlja kao jedna vrijednost, već u obliku *intervala*. Pri tome se, uz određene pretpostavke (kao što je unimodalnost funkcije) jamči da se točno rješenje nalazi unutar rezultatnog intervala, a njegova se veličina može proizvoljno smanjiti (npr. izvođenjem dodatnih iteracija postupka) za veću preciznost.

Većina postupaka određivanja minimuma funkcije jedne varijable svodi se prema tome na dva koraka:

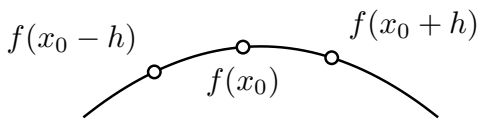
- određivanje početnog unimodalnog intervala u obliku $x \in [a, b]$;
- smanjivanje (redukcija) intervala do proizvoljne preciznosti ε.

Oba koraka mogu se provesti potpuno neovisno, tj. moguće je koristiti različite algoritme i u jednom i u drugom koraku. U nastavku ćemo prvo prikazati jedan predloženi postupak pronalaženja početnog unimodalnog intervala, a potom nekoliko metoda redukcije intervala.

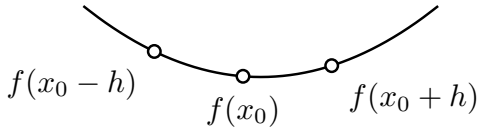
2.1 Postupak pronalaženja početnog unimodalnog intervala

Pretpostavit ćemo optimizaciju unimodalne funkcije jedne varijable $f(x)$ te postojanje neke zadane početne točke x_0 (koja se može odabrati i proizvoljno). Unimodalni interval, unutar kojega se po pretpostavci nalazi minimum promatrane funkcije, moguće je odrediti promatranjem vrijednosti funkcije u barem tri točke: dvije točke koje nazivamo vanjskim točkama, te jednoj točki između njih koju nazivamo unutarnjom. Ako je vrijednost funkcije u unutarnjoj točki manja od vrijednosti u obje vanjske točke, tada se minimum nalazi između vanjskih točaka, koje čine unimodalni interval. Uz samo jednu zadani početnu točku x_0, potrebno je odabrati barem još dvije dodatne točke kako bismo mogli ispitati položaj unimodalnog intervala.

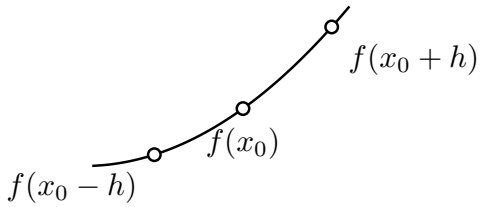
Predloženi postupak pronalaženja intervala koristi jedan dodatni parametar h kojega nazivamo početnim korakom. Postupak započinje određivanjem vrijednosti funkcije cilja u tri točke: x_0, $x_0 + h$ te $x_0 - h$, čime dobivamo vrijednosti $f(x_0 - h)$, $f(x_0)$, i $f(x_0 + h)$. Promatranjem odnosa ove tri vrijednosti, moguće je prepoznati četiri ishoda prikazana na slici 2.1. Prvi slučaj, gdje su vrijednosti u obje vanjske točke manje nego u unutarnjoj točki, moguć je samo ako funkcija nije unimodalna, što



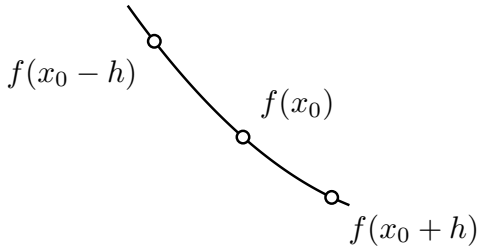
(a) Prvi slučaj



(b) Drugi slučaj



(c) Treći slučaj



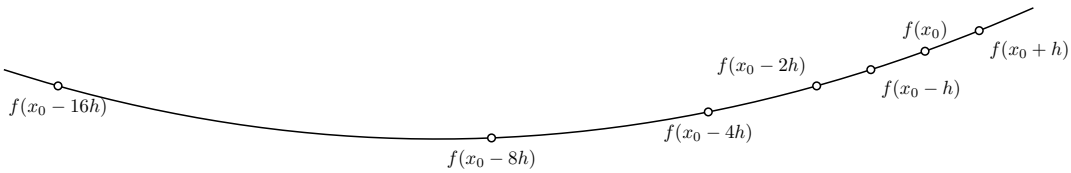
(d) Četvrti slučaj

Slika 2.1: Mogući slučajevi početnog koraka traženja unimodalnog intervala

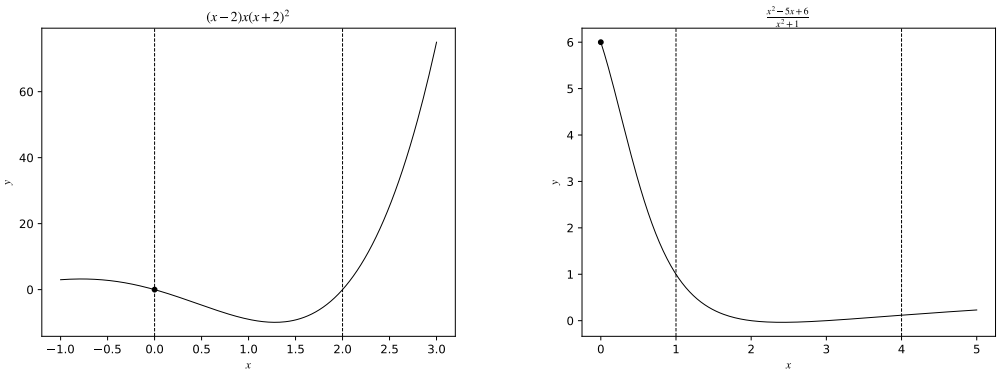
narušava našu početnu pretpostavku, te se taj ishod neće dalje razmatrati. U praksi, daljnja pretraga se može nastaviti u proizvoljnom smjeru, budući da je očito postojanje više od jednog minimuma. Ukoliko su vrijednosti funkcije u vanjskim točkama veće nego u unutarnjoj točki (drugi slučaj), ispunjen je uvjet za lokalizaciju minimuma koji se tada sigurno nalazi između vanjskih točaka; u tom slučaju, postupak se zaustavlja i kao rezultat vraća interval $[x_0 - h, x_0 + h]$.

Ako je pak najmanja funkcijska vrijednost pronađena u jednoj od vanjskih točaka, npr. u točki $x_0 - h$ kao na trećoj slici, očito je da se minimum nalazi "lijevo" od početne točke. U tom slučaju postupak nastavlja ispitivati vrijednosti funkcije cilja u novim točkama u tom smjeru, kako bismo ogradili minimum. Predloženi postupak tada ispituje vrijednost funkcije u točkama čiji položaj iznosi $x_0 - 2^i * h$, gdje i predstavlja trenutnu iteraciju postupka, čime je svaka nova točka na dvostruko većoj udaljenosti od početne točke. Razlog za ovakav pristup je činjenica da ne znamo koliko je minimum udaljen, te da uz što manje evaluacija funkcije cilja želimo doći do unimodalnog intervala.

Iteracije ispitivanja funkcije cilja nastavljaju se sve dok vrijednost u promatranom smjeru ne počne *rasti*, odnosno do uvjeta da je $f(x_0 - 2^k * h) > (x_0 - 2^{k-1} * h)$. U tom trenutku vidimo da predzadnja posjećena točka, $x_0 - 2^{k-1} * h$, ima manju funkcijsku vrijednost od zadnje točke $x_0 - 2^k * h$, ali i od one



Slika 2.2: Primjer postupka određivanja unimodalnog intervala



Slika 2.3: Primjeri pronalaženja unimodalnog intervala

u točki $x_0 - 2^{k-2} * h$ jer je do tog trenutka vrijednost funkcije uvijek padala. Time smo zapravo došli do identične situacije kao u drugom slučaju, jer je u obje vanjske točke vrijednost funkcije veća nego u unutarnjoj točki; time je položaj minimuma ograničen na interval $[x_0 - 2^k * h, x_0 - 2^{k-2} * h]$.

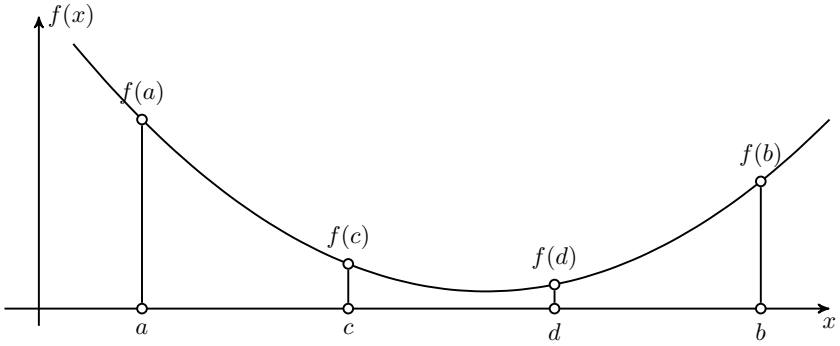
Konačno, situacija simetrična prethodnoj prikazana je u četvrtom slučaju, gdje vrijednost funkcije cilja monotono raste s povećanjem iznosa varijable x. Postupak tada na identičan način ispituje vrijednost funkcije u točkama $x_0 + 2^i * h$, sve dok ne bude ispunjen uvjet $f(x_0 + 2^k * h) > (x_0 + 2^{k-1} * h)$. U tom trenutku postupak se zaustavlja i kao rezultat daje interval $[x_0 + 2^{i-2} * h, x_0 + 2^i * h]$.

Na slici 2.2 prikazan je primjer određivanja unimodalnog intervala. U početnom koraku ispitane su vrijednosti u točkama $x_0 - h$ i $x_0 + h$ i uspoređene s vrijednosti u početnoj točki. Postupak nastavlja ispitivati točke "lijevo", odnosno za padajuće vrijednosti argumenta, u sve većim razmacima. U posljednjoj iteraciji ispitana je točka $x_0 - 16h$ te se, budući da je vrijednost funkcije u toj točki veća nego u prethodnoj $x_0 - 8h$, postupak zaustavlja. Budući da vrijedi da je $f(x_0 - 16h) > f(x_0 - 8h)$ te istovremeno $f(x_0 - 8h) < f(x_0 - 4h)$, može se zaključiti kako se minimum nalazi unutar intervala $[x_0 - 16h, x_0 - 4h]$.

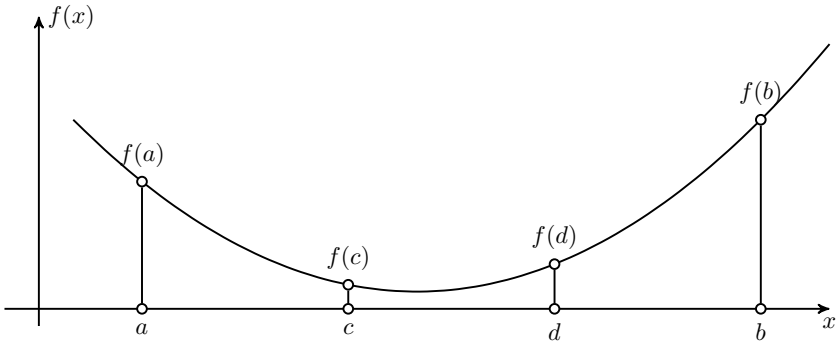
Ovaj postupak ne predstavlja jedini mogući pristup, no mi ćemo u nastavku pretpostaviti da je početni unimodalni interval, ukoliko nije zadan, uvijek određen upravo prethodno opisanim algoritmom.

2.2 Redukcija unimodalnog intervala

Nakon pronalaženja početnog unimodalnog intervala, isti je moguće reducirati do proizvoljno male veličine ε, tj. do željene razine preciznosti. Smanjivanje intravala svodit će se na uzastopno odbacivanje dijelova intervala za koje znamo da ne sadrže točku minimuma. To je učinkovito moguće provesti jedino odbacivanjem rubnih dijelova intervala, pa se redukcija zapravo obavlja odbacivanjem jedne od rubnih točaka trenutnog intervala. Kako bismo mogli odbaciti jednu rubnu točku, potrebna nam je informacija o kretanju funkcije cilja unutar intervala. Sa samo jednom unutarnjom točkom to nije moguće izvesti, jer nemamo nikakvu informaciju o odnosu vrijednosti funkcije s lijeve i desne strane jedne unutarnje točke;



(a) Prvi slučaj: odbacuje se podinterval $[a, c]$



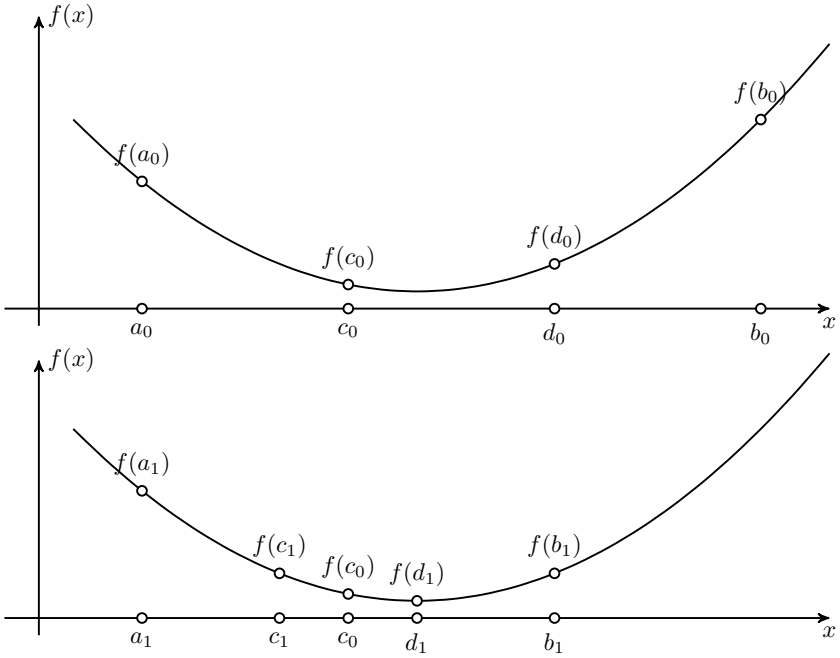
(b) Drugi slučaj: odbacuje se podinterval $[d, b]$

Slika 2.4: Korak redukcije unimodalnog intervala

istovremeno, interval želimo reducirati što učinkovitije, odnosno uz što manje evaluacija funkcije cilja. Stoga svi postupci redukcije intervala koriste dvije unutarnje točke, temeljem čega se može odrediti koji od tri dobivena podintervala može biti odbačen.

Primjer takve podjele intervala $[a, b]$ prikazan je na slici 2.4, uz pomoć dvaju dodatnih točaka c i d. Korištenjem vrijednosti funkcije cilja u te dvije točke može se odrediti u kojem podintervalu se nalazi minimum. Ako je $f(c) > f(d)$, onda se minimum sigurno nalazi u podintervalu $[c, b]$, a interval $[a, c]$ se odbacuje. Budući da se funkcijska vrijednost smanjuje od točke c do točke d, to znači da se lijevo od točke c sigurno ne nalazi točka s manjom funkcijskom vrijednošću. S druge strane, ako vrijedi da je $f(c) < f(d)$, onda znamo da se minimum nalazi u podintervalu $[a, d]$. Nakon što je trenutni interval reduciran na neki od podintervala, postupak se ponavlja željene veličine intervala manje ili jednake ε.

Postavlja se pitanje kako rasporediti unutarnje točke intervala u svakom koraku redukcije? Naivni pristup mogao bi interval jednostavno podijeliti na jednake dijelove, odnosno na trećine. Na slici 2.5 prikazana je takva podjela gdje su unutarnje točke određene kao $c_0 = \frac{b_0 - a_0}{3}$ i $c_0 = \frac{2*(b_0 - a_0)}{3}$. Na temelju odnosa vrijednosti funkcije u točkama c i d, odbacujemo podinterval $[d_0, b_0]$. Točke a_0 i d_0 predstavljaju granice novog intervala te su označene s a_1 i b_1 na donjem dijelu slike 2.5. Želimo li i novodobiveni interval ponovo podijeliti na trećine, potrebne bi nam bile dvije nove točke c_1 i d_1 u intervalu $[a_1, b_1]$, no to nosi sljedeće nedostatke: prvo, ne koristimo informaciju o vrijednosti funkcije iz postojeće točke c_0, koja se već nalazi u tom intervalu. Kao drugo, ovaj pristup zahtijevao bi dodatne dvije evaluacije funkcije cilja u svakom koraku redukcije, iako znamo da nam iz prethodnog koraka uvijek ostaje jedna točka koju možemo iskorisiti u sljedećem koraku.



Slika 2.5: Postupak podjele intervala na trećine

Drugi pristup bio bi da se točka c_0 iskoristi i izračuna jedna dodatna točka, primjerice c_1, pa se pomoću te dvije točke dalje provodi redukcija. Problem ovakvog pristupa je što se točka c_0 nalazi točno na polovici intervala $[a_1, b_1]$, dok bi se točka c_1 nalazila na jednoj trećini. To bi značilo da podintervali $[a_1, c_0]$ i $[c_1, b_1]$ ne bi bili jednaki, što bi dovelo do toga da bi broj koraka postupka uvelike ovisio o izgledu funkcije te bi u nekim slučajevima smanjivanje intervala moglo biti sporije i neučinkovito.

Iz prethodno opisanih razloga, učinkovita metoda redukcije intervala trebala bi imati ova svojstva:

- u svakom koraku obavlja se samo jedna evaluacija funkcije (jer koristimo jednu unutarnju točku iz prethodnog koraka);
- u svakom koraku želimo smanjiti interval za jednaki faktor; drugim riječima, omjer veličina intervala u uzastopnim koracima bi trebao biti stalan.

Metoda zlatnog reza, koja je opisana u sljedećem odjeljku, ispunjava oba ova ciljana svojstva.

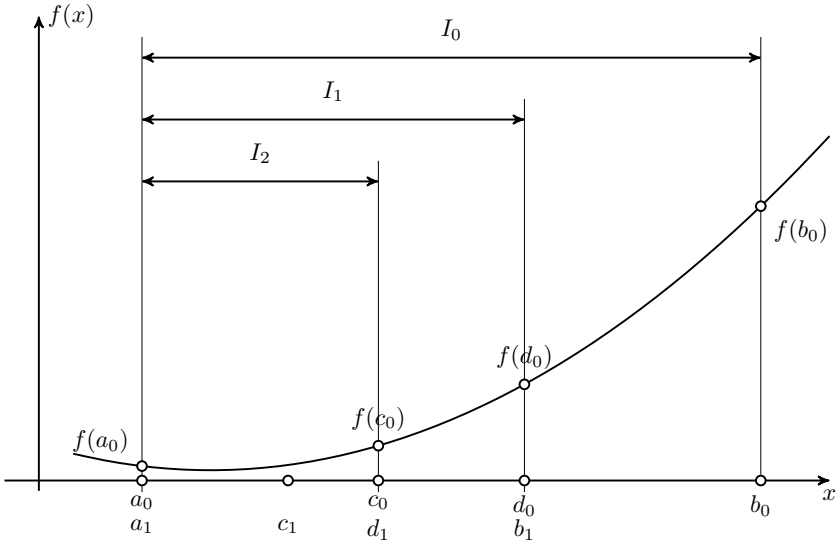
2.3 Postupak zlatnog reza

Oblikovanje postupka zlatnog reza, kao i drugih postupaka redukcije intervala, svodi se zapravo na definiranje položaja *unutarnjih* točaka intervala. U svakoj iteraciji postupka, unutarnje točke bit će postavljene na jednakim položajima s obzirom na granice intervala (kako bi ispunili drugi od prethodno navedena dva uvjeta), odnosno dijelit će trenutni interval u stalnom omjeru. Pretpostavimo da je taj omjer poznat i označen s k te da su zadane granice početnog intervala $[a_0, b_0]$ i njegova veličina I_0; u tom slučaju, unutarnje točke c_0 i d_0 možemo odrediti na sljedeći način:

$$c_0 = b_0 - k(b_0 - a_0) \tag{2.1}$$

$$d_0 = a_0 + k(b_0 - a_0). \tag{2.2}$$

Treba naglasiti kako će se unutarnje točke računati na ovaj način u svakoj iteraciji, samo uzimajući u obzir trenutne granice intervala $[a_i, b_i]$, a uz stalan omjer podjele k.



Slika 2.6: Izvedba postupka zlatnog reza

Kako bi odredili nepoznati omjer k, ilustrirajmo dva uzastopna koraka redukcije intervala, što je prikazano na slici 2.6. Uz kretanje funkcije cilja na slici, u ovom koraku odbacujemo desnu granicu početnog intervala, b_0, jer je $f(c) < f(d)$, pa su granice intervala u idućem koraku $a_1 = a_0$ i $b_1 = d_0$. Duljinu novog intervala $[a_1, b_1]$ označit ćemo s I_1. Kako želimo iskoristiti unutarnju točku c_0, ona sada postaje nova *desna* unutarnja točka d_1. Drugu potrebnu unutarnju točku c_1 možemo odrediti na jednak način kao i u prethodnom koraku uz nove granice intervala, tj.

$$c_1 = b_1 - k(b_1 - a_1). \tag{2.3}$$

U sljedećem koraku opet vrijedi $f(c) < f(d)$, pa bi novi interval, nakon odbacivanja desne granice, bio $[a_1, d_1]$, čija je veličina I_2.

Promotrimo odnose veličina intervala u uzastopnim koracima: kako se u svakom koraku trenutni interval smanjuje za jednaki faktor k, vrijedi $\frac{I_0}{I_1} = \frac{I_1}{I_2} = k$. Nadalje, budući da se unutarnje točke postavljaju simetrično s obzirom na granice intervala (tj. odmaknute za jednaku udaljenost od njihovih najbližih granica), tada vidimo i da vrijedi

$$I_0 = I_1 + I_2, \tag{2.4}$$

odnosno da je veličina prvog intervala jednaka zbroju veličina podintervala u iduće dvije iteracije. To možemo iskoristiti i podijeliti čitav izraz s I_0

$$1 = \frac{I_1}{I_0} + \frac{I_2}{I_0}. \tag{2.5}$$

Iz omjera intervala I_0, I_1 i I_2, možemo $\frac{I_1}{I_0}$ zamijeniti s k, a $\frac{I_2}{I_0}$ s k^2 pa se dobiva kvadratna jednadžba

$$1 = k + k^2, \tag{2.6}$$

odnosno

$$k^2 + k - 1 = 0. \tag{2.7}$$

Rješavanjem kvadratne jednadžbe dobivaju se moguća rješenja

$$k_{1,2} = \frac{-1 \pm \sqrt{5}}{2}, \tag{2.8}$$

od kojih jedino pozitivno rješenje ima smisla:

$$k = \frac{-1+\sqrt{5}}{2} \approx 0.618. \tag{2.9}$$

Na ovaj način određen je faktor za koji se smanjuje interval u postupku zlatnog reza, koji se naziva i *koeficijentom zlatnog reza*. Prednost korištenja ovog omjera podjele jest u činjenici da se obje unutarnje točke mogu iskoristiti u idućem koraku, jedna kao nova granica intervala, a druga kao jedna od unutarnjih točaka. Nakon prve iteracije, u kojoj je potrebno izračunati funkcijske vrijednosti u dvije unutarnje točke, u svakoj je idućoj iteraciji potrebno izračunati vrijednost funkcije samo u novoj unutarnjoj točki. Osim toga, budući da se koristi konstantan faktor smanjenja intervala, moguće je unaprijed odrediti broj koraka koje je potrebno napraviti kako bi početni interval smanjili do zadane preciznosti. Do tog izraza možemo doći na način da postavimo nejednadžbu

$$(b-a) * k^n \leq \varepsilon, \tag{2.10}$$

gdje n predstavlja broj koraka. Ako se n izluči, dobiva se izraz

$$n = \left\lceil \frac{\log \frac{\varepsilon}{b-a}}{\log k} \right\rceil. \tag{2.11}$$

Drugim riječima, broj iteracija će biti prvi veći cijeli broj od zadanog izraza.

U izvedbi postupka zlatnog reza, za početni interval $[a_0, b_0]$ položaji unutarnjih točaka računaju se kao

$$c_0 = b_0 - k(b_0 - a_0), \tag{2.12}$$

$$d_0 = a_0 + k(b_0 - a_0). \tag{2.13}$$

Nakon toga, dok god je širina intervala $[a_i, b_i]$ u i-toj iteraciji manja od željene preciznosti ε, ispituju se odnosi vrijednosti funkcije cilja u točkama c_i i d_i. Ako vrijedi $f(c_i) < f(d_i)$, točke za iduću iteraciju postavljaju se kao

$$a_{i+1} = a_i \tag{2.14}$$

$$b_{i+1} = d_i \tag{2.15}$$

$$d_{i+1} = c_i \tag{2.16}$$

$$c_{i+1} = b_i - k(b_i - a_i). \tag{2.17}$$

U slučaju da vrijedi $f(c_i) > f(d_i)$, tada se točke postavljaju uz pomoć izraza

$$a_{i+1} = c_i \tag{2.18}$$

$$b_{i+1} = b_i \tag{2.19}$$

$$c_{i+1} = d_i \tag{2.20}$$

$$d_{i+1} = b_i - k(b_i - a_i). \tag{2.21}$$

Algoritam 2.1: Algoritam zlatnog reza

```
Function ZlatniRez(f, a, b, ε):
    Podatci: f: funkcija koja se minimizira
             a: početna granica unimodalng intervala
             a: završna granica unimodalnog intervala
             ε: željena preciznost unimodalnog intervala
    k = (1+√5)/2;
    c = b − k ∗ (b − a);
    d = a + k ∗ (b − a);
    fc = f(c);
    fd = f(d);
    dok |a − b| < ε čini
        ako fc < fd onda
            b = d;
            d = c;
            c = b − k ∗ (b − a);
            fd=fc;
            fc = f(c);
        inače
            a = c;
            c = d;
            d = a + k ∗ (b − a);
            fc=fd;
            fd = f(d);
        kraj
    kraj
    vrati (a+b)/2;
```

Primjer 2.1

Zadana je funkcija $f(x) = (x-5)^2$ i početni unimodalni interval $[-3, 7]$. Potrebno je provesti postupak zlatnog reza dok se unimodalni interval ne smanji do preciznosti $\varepsilon < 1$.

U prvoj iteraciji zlatnog reza početni interval jednak je $a_1 = -3$ i $b_1 = 7$. Kao što se može vidjeti, širina intervala veća je od željene preciznosti, stoga se računaju unutarnje točke c_1 i d_1

$$c_1 = b_1 - k(b_1 - a_1) = 7 - 0.618 \cdot (7 - (-3)) = 0.82 \tag{2.22}$$

$$d_1 = a_1 + k(b_1 - a_1) = 3 + 0.618 \cdot (7 - (-3))) = 3.18 \tag{2.23}$$

Nakon toga računaju se vrijednosti funkcije u ovim točkama:

$$f(c_1) = (0.82 - 5)^2 = 17.4724 \tag{2.24}$$

$$f(d_1) = (3.18 - 5)^2 = 3.3124 \tag{2.25}$$

Možemo vidjeti kako je funkcijska vrijednost manja u točki d_1 nego u točki c_1. To znači da se minimum sigurno nalazi između točaka c_1 i b_1, te se u idućoj iteraciji razmatra interval $[c_1, b_1]$. U idućoj iteraciji točka d se ne mijenja, stoga vrijedi $d_2 = d_1 = 7$. Točka a_1 se odbacuje i umjesto

nje uzimamo točku c_1 kao novu granicu unimodalnog intervala $a_2 = c_1 = 0.82$. Konačno, druga unutarnja točka d_1 u novom intervalu ne predstavlja lijevu već desnu unutarnju točku, odnosno vrijedi $c_2 = d_1 = 3.18$.

U drugoj iteraciji imamo položaje svih točaka osim unutarnje točke d_2. Za razliku od prve iteracije gjde je bilo potrebno računati vrijednosti obaju unutarnjih točaka c_1 i d_1, u ovoj iteraciji i nadalje je potrebno odrediti položaj samo jedne točke, što je u ovom slučaju točka d_2:

$$d_2 = a_2 + k(b_2 - a_2) = 0.82 + 0.618 \cdot (7 - 0.82)) = 4.64. \tag{2.26}$$

Funkcijska vrijednost točke d_2 je $f(d_2) = 0.1296$. Usporedbom funkcijskih vrijednosti točaka c_2 i d_2 vidimo da točka d_2 ima manju funkcijsku vrijednost od točke c_2, koja ima vrijednost 3.3124. To znači da se minimum nalazi u intervalu $[c_2, b_2]$ te se desna granica opet ne mijenja, odnosno $b_3 = b_2$, a lijeva granica se postavlja na lijevu unutarnju točku, odnosno $a_3 = c_3$. Konačno, desna unutarnja točka d_2 u novom intervalu nalazi se na mjestu lijeve unutarnje točke pa vrijedi $c_3 = d_3$.

U trećoj iteraciji potrebno je opet odrediti točku d kao

$$d_3 = a_3 + k(b_3 - a_3) = 3.18 + 0.618 \cdot (7 - 3.18)) = 5.54, \tag{2.27}$$

čija je funkcijska vrijednost jednaka $f(d_3) = 0.2916$. Možemo vidjeti kako točka c_3 ima manju funkcijsku vrijednost od točke d_3 te se zbog toga minimum nalazi u intervalu $[a_3, d_3]$. Lijeva granica intervala u idućem koraku ostaje $a_4 = a_3 = 3.18$, dok se desna granica ažurira kao $b_4 = d_3$. Točka b_3 se odbacuje, a točka c_3 postaje nova unutarnja desna točka, odnosno $d_4 = c_3$.

U četvrtoj iteraciji određujemo unutarnju točku c_3

$$c_4 = b_4 - k(b_4 - a_4) = 5.54 - 0.618 \cdot (5.54 - (-3.18)) = 4.08, \tag{2.28}$$

čija je funkcijska vrijednost jednaka $f(c_4) = 0.8464$. Ovaj put točka d_4 ima manju funkcijsku vrijednost, te se minimum nalazi u intervalu $[c_i, b_i]$. Lijeva granica intervala je tada $a_5 = c_4$, dok desna ostaje $b_5 = b_4$. Također, postavlja se lijeva unutarnja točka u za novu iteraciju $c_5 = d_5$.

U petoj iteraciji uvjet zaustavljanja još nije zadovoljen jer $|b_5 - a_5| = 1.46 \not< \varepsilon$, pa se postupak nastavlja. Računa se nova unutarnja točka

$$d_5 = a_5 + k(b_5 - a_5) = 4.08 + 0.618 \cdot (5.54 - 4.08)) = 4.98, \tag{2.29}$$

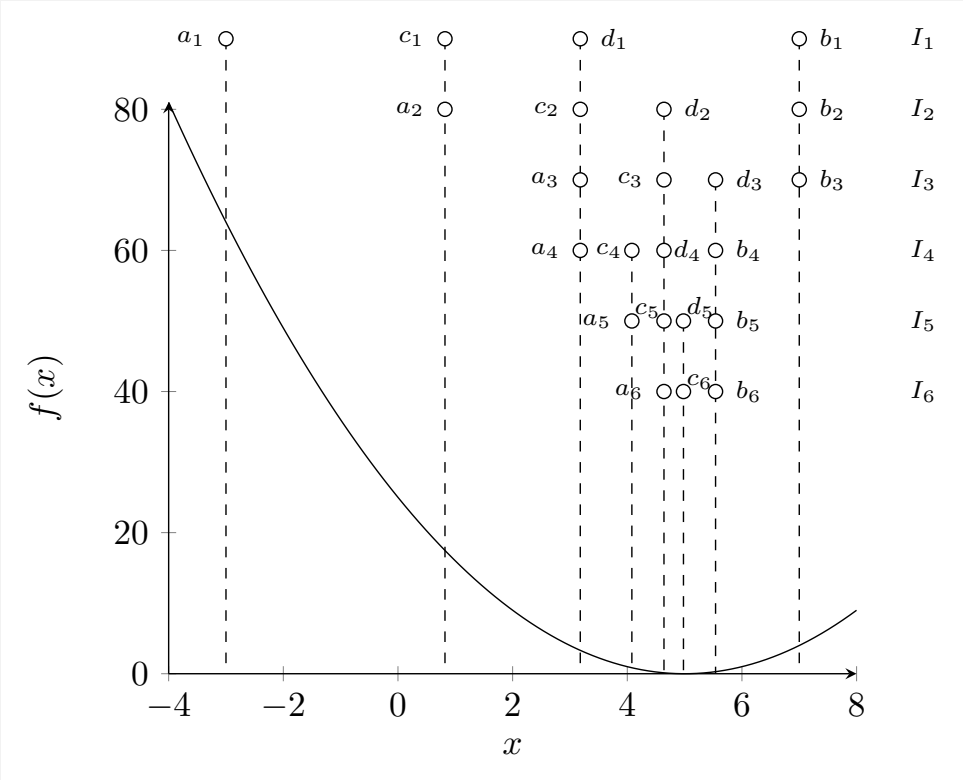
čija je funkcijska vrijednost $f(d_5) = 0.04$. Točka d_5 ima manju vrijednost od točke c_5, što znači da se lijeva granica postavlja na točku $a_6 = c_5$. Desna granica ostaje nepromijenjena $b_6 = b_5$, a lijeva se točka u idućoj iteraciji postavlja na d_5, odnosno $c_6 = d_5$.

Provjerom kriterija zaustavljanja prije početka šeste iteracije vidimo da je uvjet zaustavljanja sada zadovoljen uz $|b-a| = 0.9 < 1$. Stoga se postupak prekida te se kao rezultat vreća unimodalni interval $[a_6, b_6] = [4.64, 5.54]$. Umjesto intervala, moguće je vratiti i točku koja se nalazi u tom intervalu, pri čemu se najčešće uzima sredina intervala $\frac{5.54+4.64}{2} = 5.09$. Može se vidjeti kako je postupak reducirao interval do željene preciznosti u pet koraka (jer je postupak zaustavljen prije nego što je provedena šesta iteracija).

Koraci postupka mogu se pregledno prikazati i tablično, što je prikazano u tablici 2.1. U tablici je označeno kako se točke iz iteracije u iteraciju ažuriraju na temelju prethodnih vrijednosti te koje se točke nanovo računaju u svakoj iteraciji. Slika 2.7 prikazuje grafički kako postupak iterativno smanjuje interval oko minimuma funkcije.

Tablica 2.1: Prikaz iteracija postupka zlatnog reza

i	a_i	c_i	d_i	b_i
1	-3	0.82	3.18	7
2	0.82	3.18	4.64	7
3	3.18	4.64	5.54	7
4	3.18	4.08	4.64	5.54
5	4.08	4.64	4.98	5.54
6	4.64	4.98		5.54



Slika 2.7: Primjer provedbe postupka zlatnog reza

♣

2.4 Fibonaccijeva metoda

Fibonaccijeva metoda minimizacije jednodimenzionalne funkcije slična je metodi zlatnog ideja. Glavna ideja ostaje ista, i u ovoj metodi unutarnje točke iz iteracije u iteraciju žele se posložiti na način da je u svakoj novoj iteraciji potrebno računati jednu dodatnu točku. Ako I_0 predstavlja početni interval, prisjetimo se iz zlatnog reza da je moralo vrijediti

$$I_0 = I_1 + I_2 \tag{2.30}$$

$$I_1 = I_2 + I_3 \vdots$$

$$I_n = I_{n+1} + I_{n+2} \tag{2.31}$$

Uz pretpostavku da je $I_{n+2} = 0$, odnosno da u zadnjem koraku interval više ne trebamo podijeliti na manje podintervale. Na temelju zadanih izraza možemo izračunati kako bi se odnosile duljine pojedinih

intervala. Ako krenemo od najmanjeg intervala

$$I_{n+1} = I_n - I_{n+2} = I_n \tag{2.32}$$

$$I_n = I_{n+1} + I_{n+2} = I_n \tag{2.33}$$

$$I_{n-1} = In + I_{n+1} = I_n + I_n = 2I_n \tag{2.34}$$

$$I_{n-2} = I_{n-1} + I_n = 2I_n + I_n = 3I_n \tag{2.35}$$

$$I_{n-3} = I_{n-2} + I_{n-1} = 3I_n + 2I_n = 5I_n \tag{2.36}$$

$$I_{n-4} = I_{n-3} + I_{n-2} = 5I_n + 3I_n = 8I_n \tag{2.37}$$

$$I_{n-5} = I_{n-4} + I_{n-3} = 8I_n + 5I_n = 13I_n \tag{2.38}$$

$$\vdots$$

$$I_0 = I_1 + I_2 = F_n I_n \tag{2.39}$$

Možemo vidjeti kako ovaj niz jednakosti podsjeća upravo na Fibonaccijev niz brojeva. Prisjetimo se da u je Fibonaccijev niz definiran kao

$$F_n = F_{n-1} + F_{n-2}, \text{za } n > 1 \tag{2.40}$$

$$F_1 = 1 \tag{2.41}$$

$$F_0 = 1. \tag{2.42}$$

Posljedično, prethodne jednakosti možemo kraće zapisati kao

$$I_{n+1} = F_1 I_n \tag{2.43}$$

$$I_n = F_2 I_n \tag{2.44}$$

$$I_{n-1} = F_3 I_n \tag{2.45}$$

$$\vdots$$

$$I_1 = F_{n-1} I_n \tag{2.46}$$

$$I_0 = F_n I_n \tag{2.47}$$

Na temelju prethodnih izraza možemo također odrediti i omjer između pojedinih intervala. Primjerice, ako uzmemo intervale I_0 i I_1, njihov omjer je jednak

$$\frac{I_1}{I_0} = \frac{F_{n-1}}{F_n}. \tag{2.48}$$

Dakle, omjer dva uzastopna intervala jednak je omjeru pridruženih im Fibonaccijevih brojeva. Tu spoznaju koristimo kod određivanja faktora redukcije u pojedinom koraku postupka. Dakle umjesto konstantnog faktora redukcije od 0.618 koji se koristi kod zlatnog reza, ovdje će faktor redukcije ovisiti o konkretnom

koraku

$$\frac{F_0}{F_1} = 1 \tag{2.49}$$

$$\frac{F_1}{F_2} = 0.5 \tag{2.50}$$

$$\frac{F_2}{F_3} = 0.666 \tag{2.51}$$

$$\frac{F_3}{F_4} = 0.6 \tag{2.52}$$

$$\frac{F_5}{F_5} = 0.625 \tag{2.53}$$

$$\vdots$$

$$\frac{F_{11}}{F_{12}} = 0.618 \tag{2.54}$$

Može se dakle vidjeti da kvocijent trenutnog i idućeg broja u Fibonaccijevom nizu teži prema faktoru redukcije koji se koristi i kod zlatnog reza. Jedino za manje vrijednosti su odstupanja od tog koeficijenta značajnije.

Nakon što je definirano kako računamo koeficijent redukcije u pojedinim iteracijama, onda možemo odrediti i kako se unutarnje točke određuju kod Fibonaccijevog postupka. Ako je zadan interval $[a, b]$, onda se unutranje točke preko Fibonaccijevog postupka određuju kao

$$c = b - \frac{F_{n-1}}{F_n}(b - a), \tag{2.55}$$

$$d = a + \frac{F_{n-1}}{F_n}(b - a). \tag{2.56}$$

Odabir podintervala obavlja se na potpuno isti način kao kod zlatnog reza usporedbom funkcijskih vrijednosti u unutarnjim točkama c i d. No pitanje koje se i dalje postavlja jest s kojim F_n započeti postupak. Iz prethodno navedenih jednakosti vidjeli smo da će vrijediti $I_0 = F_n I_n$, dakle intervalu kojeg želimo dobiti pomnoženim s odgovarajućim Fibonaccijevim brojem. Ako radimo s preciznošću ϵ želimo da za krajnji interval koji dobijemo vrijedi

$$I_n < \epsilon. \tag{2.57}$$

Izraz možemo preoblikovati na način da krajnji interval zamijenimo s početnim korištenjem prethodno spomenute jednakosti

$$\frac{I_0}{F_n} < \epsilon \tag{2.58}$$

odnosno

$$F_n > \frac{I_0}{\epsilon} = \frac{b - a}{\epsilon}, \tag{2.59}$$

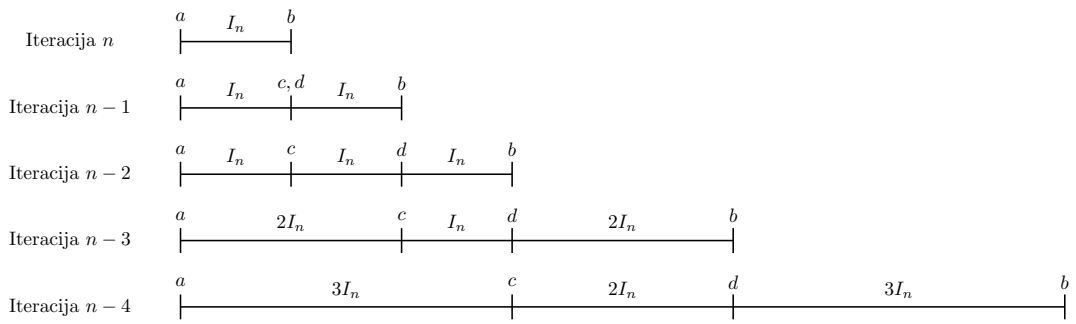
dakle uzimamo prvi Fibonaccijev broj za koji vrijedi prethodni izraz.

Iz prethodnih razmatranja možemo uočiti jednu zanimljivu situaciju koja će se dogoditi kod Fibonaccijevog postupka. Naime, možemo vidjeti da će u predzadnjem koraku postupka vrijednost koeficijenta redukcije biti jednaka $\frac{F_1}{F_2} = 0.5$, što zapravo znaći da bi se za vrijednosti unuarnjih točaka dobilo

$$c = b - 0.5(b - a) = 0.5(b - a) \tag{2.60}$$

$$d = a + 0.5(b - a) = 0.5(b - a), \tag{2.61}$$

što znači da će vrijednosti unutarnjih točaka biti jednake. No kao što je ranije bilo napomenuto, kako bi otkrili u kojem podintervalu se nalazi minimum potrebne su nam 2 točke. No kako se u idućem koraku interval više ne smanjuje, izrazima za računanje unutarnjih točaka nije moguće dobiti niti jednu



Slika 2.8: Podjela intervala Fibonnacijevim postupkom

novu točku unutar intervala. Zbog toga, kada se dođe do ovog koraka, kada se unutarnje točke poklope, vrijednost točke c oduzet će se iznimno malom vrijednošću i dobit će se nova točka $c^{'} = c - \epsilon_0$, gdje ϵ_0 predstavlja neku malu vrijednost (različitu od ϵ vrijednosti koja određuje željenu širinu reduciranog intervala). Nakon toga se uspoređuju funkcijske vrijednosti u točkama $c^{'}$ i d se vraća interval $[a, d]$ ako je vrijednost u točki $c^{'}$ manja, odnosno interval $[c^{'}, b]$ ako je funkcijska vrijednost u točki d manja.

Podjela intervala na podintervale kod Fibonnacijevog postupka može se detaljnije vidjeti na slici 2.8. U posljednoj n-toj iteraciji duljina intervala jednaka je I_n i ona predstavlja konačni dobiveni interval. U prethodnoj iteraciji možemo vidjeti kako je taj interval bio dvostruko dulji, zbog toga što su obje unutarnje točke c i d imale istu vrijednost. U $n - 2$ iteraciji.

2.5 Kvadratna interpolacija

Postupak zlatnog reza jamči stabilnu konvergenciju i omogućava točnu procjenu broja iteracija, a time i broja evaluacija funkcije cilja, koji su potrebni za postizanje željene preciznosti. Međutim, uz određene pretpostavke moguće je postići i bržu konvergenciju. Metoda kvadratne interpolacije temelji se na pretpostavci da se funkcija cilja, barem u okolici minimuma, ponaša približno kao kvadratna funkcija, $f(x) = ax^2 + bx + c$, koja ima oblik parabole.

Pretpostavimo li da su zadane tri točke x_1, x_2 i x_3, možemo izračunati njihove funkcijske vrijednosti; uz pomoć tih informacija moguće je kvadratnom interpolacijom odrediti položaj *tjemena parabole* kao

$$x_T = -\frac{x_2^2 f(x_1) - x_3^2 f(x_1) - x_1^2 f(x_2) + x_3^2 f(x_2) + x_1^2 f(x_3) - x_2^2 d(x_3)}{2(x_2 f(x_1) - x_3 f(x_2) - x_1 f(x_2) + x_3 f(x_2) + x_1 f(x_3) - x_2 f(x_3))}. \tag{2.62}$$

Za kvadratnu funkciju cilja, ovo bi rezultiralo izravnim pronalaskom minimuma u točki x_T. U slučaju da funkcija nije kvadratna, ovom aproksimacijom dobiva se točka koja bi se približno mogla poklapati s minimumom funkcije, to točnije što je funkcija više nalik kvadratnoj. Na taj način je definiran iterativan postupak redukcije intervala: uz pomoć tri postojeće točke, nova točka se određuje kao tjeme parabole konstruirane uz početne tri točke. Nakon što imamo dvije unutarnje točke, korak redukcije se provodi kao i u postupku zlatnog reza, odbacivanjem jedne vanjske točke na temelju odnosa vrijednosti funkcije cilja u unutarnjim točkama.

Postupak kvadratne interpolacije izvodi se tako da je na početku je potrebno zadati početni unimodalni interval $[a, b]$. Kako su za kvadratnu interpolaciju potrebno 3 točke, treću točku c potrebno je odabrati iz unimodalnog intervala, primjerice odabirom točke koja se nalazi na polovici početnog intervala. Nakon toga računa se druga unutarnja točka d na temelju izraza 2.62. Budući da unaprijed ne znamo položaj tjemena u odnosu na početne tri točke, mogu se pojaviti četiri slučaja koja su prikazana na slici 2.9. Prva

dva slučaja pokazuju situaciju kada se točka d nalazi lijevo od točke c. Ovisno o izgledu funkcije ona ne mora nužno imati manju funkcijsku vrijednost od točke c, stoga način na koji će se točke ažurirati ovisi o odnosu funkcijskih vrijednosti točaka c i d. Ako vrijedi $f(c) > f(d)$, onda se točke u sljedećoj iteraciji $k+1$ ažuriraju kao

$$a_{k+1} = a_k \tag{2.63}$$

$$b_{k+1} = d_k \tag{2.64}$$

$$c_{k+1} = b_k. \tag{2.65}$$

Ako s druge strane vrijedi da je $f(c) < f(d)$, točke se ažuriraju izrazima

$$a_{k+1} = d_k \tag{2.66}$$

$$b_{k+1} = b_k \tag{2.67}$$

$$c_{k+1} = c_k. \tag{2.68}$$

Ako se točka d nalazi s desne strane točke c i ako vrijedi $f(c) > f(d)$, onda se točke ažuriraju izrazima

$$a_{k+1} = b_k \tag{2.69}$$

$$b_{k+1} = d_k \tag{2.70}$$

$$c_{k+1} = c_k, \tag{2.71}$$

dok se u suprotnom uz $f(c) < f(d)$ ažuriranje provodi kao

$$a_{k+1} = a_k \tag{2.72}$$

$$b_{k+1} = b_k \tag{2.73}$$

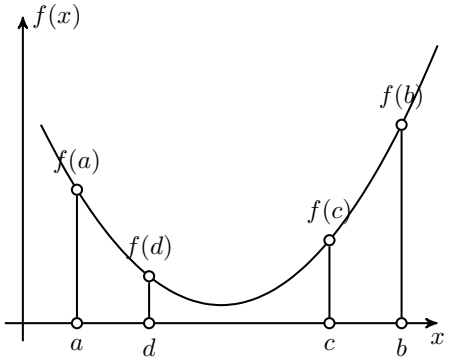
$$c_{k+1} = d_k. \tag{2.74}$$

Na taj način osigurano je da točke a i b uvijek predstavljaju granice unimodalnog intervala unutar kojeg se nalazi točka c. Postupak na ovaj način iterativno ažurira točke sve dok nije zadovoljen neki kriterij zaustavljanja. Osim širine intervala, može se koristiti kriterij da je udaljenost između trenutne točke d_k i prethodne točke d_{k-1} manja od željene preciznosti:

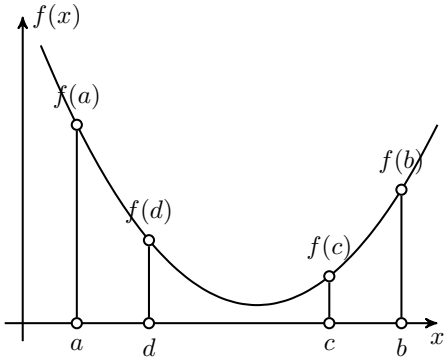
$$|d_k - d_{k-1}| < \varepsilon. \tag{2.75}$$

Primjer rada kvadratne interpolacije prikazan je na slici 2.10. Crnom bojom označene su funkcija cilja i točke na temelju kojih se računa interpolacija. Crvenom bojom označena je kvadratna funkcija koja predstavlja aproksimaciju, kao i izračunata točka d koja predstavlja minimum kvadratne funkcije. U prvom koraku vidimo kako se, iako postupak loše aproksimira traženu funkciju, dobivena točka d nalazi relativno blizu minimuma. Već u sljedećoj iteraciji možemo vidjeti kako je područje oko minimuma puno bolje aproksimirano te da je novodobivena točka d još bliža minimumu.

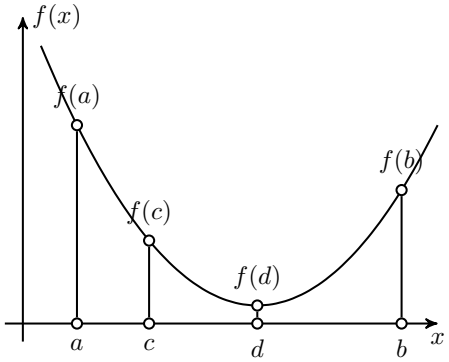
Algoritam 2.2 prikazuje postupak kvadratne interpolacije. U svakoj se iteraciji određuje točka d te se sve točke ažuriraju ovisno o relativnom položaju i vrijednosti funkcije cilja.



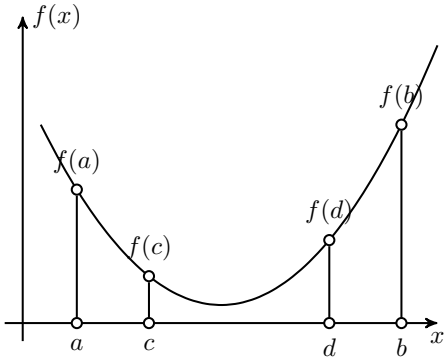
(a) Prvi slučaj



(b) Drugi slučaj

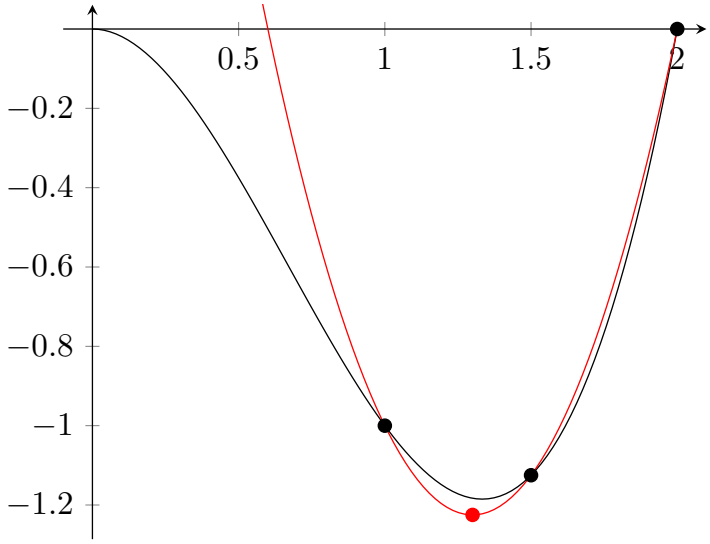


(c) Treći slučaj

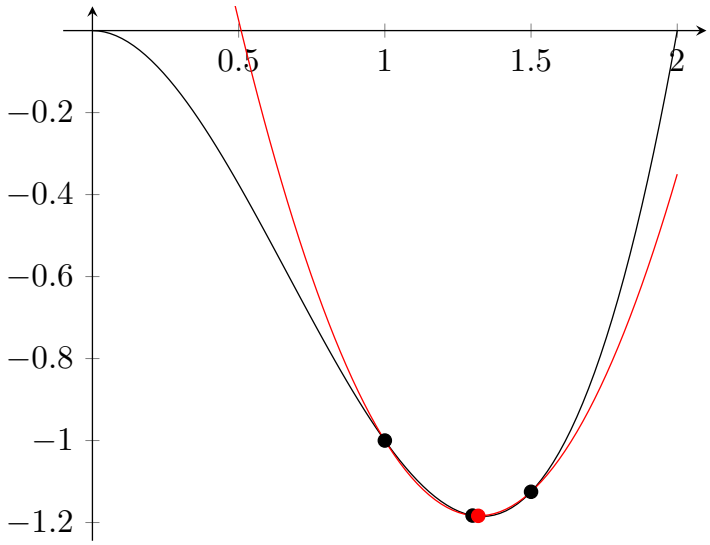


(d) Četvrti slučaj

Slika 2.9: Mogući slučajevi položaja tjemena prilikom kvadratne interpolacije



(a) Prvi korak



(b) Drugi korak

Slika 2.10: Primjer provedbe postupka kvadratne interpolacije

Algoritam 2.2: Algoritam kvadratne interpolacije

Function `KvadratnaInterpolacija`(f, a, b, ε)**:**

Podatci: f: funkcija koja se minimizira

a: početna granica unimodalng intervala

a: završna granica unimodalnog intervala

ε: željena preciznost unimodalnog intervala

$c = \frac{a+b}{2}$;

$d = c$;

ponavljaj

$d_s = d$;

$d = -\frac{x_2^2 f(x_1) - x_3^2 f(x_1) - x_1^2 f(x_2) + x_3^2 f(x_2) + x_1^2 f(x_3) - x_2^2 d(x_3)}{2(x_2 f(x_1) - x_3 f(x_2) - x_1 f(x_2) + x_3 f(x_2) + x_1 f(x_3) - x_2 f(x_3))}$;

ako $a < d < c$ *i* $f(d) < f(c))$ **onda**

$c = b$;

$b = d$;

inače ako $a < d < c$ *i* $f(d) < f(c))$ **onda**

$c = b$;

$b = d$;

inače ako $a < d < c$ *i* $f(d) < f(c))$ **onda**

$c = b$;

$b = d$;

inače ako $a < d < c$ *i* $f(d) < f(c))$ **onda**

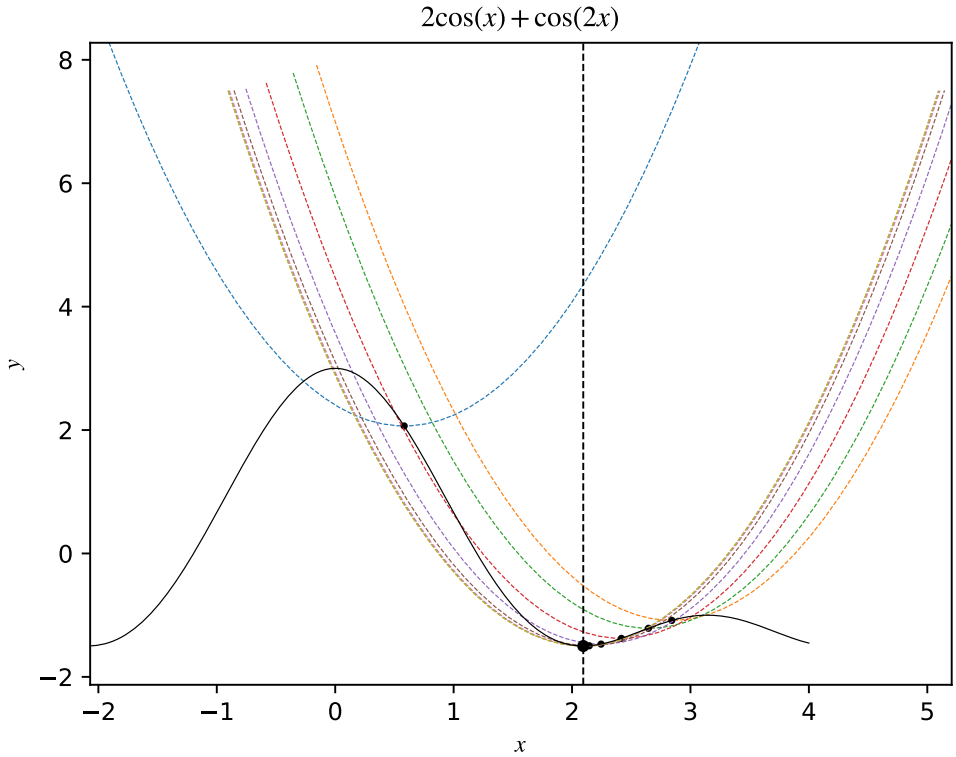
$c = b$;

$b = d$;

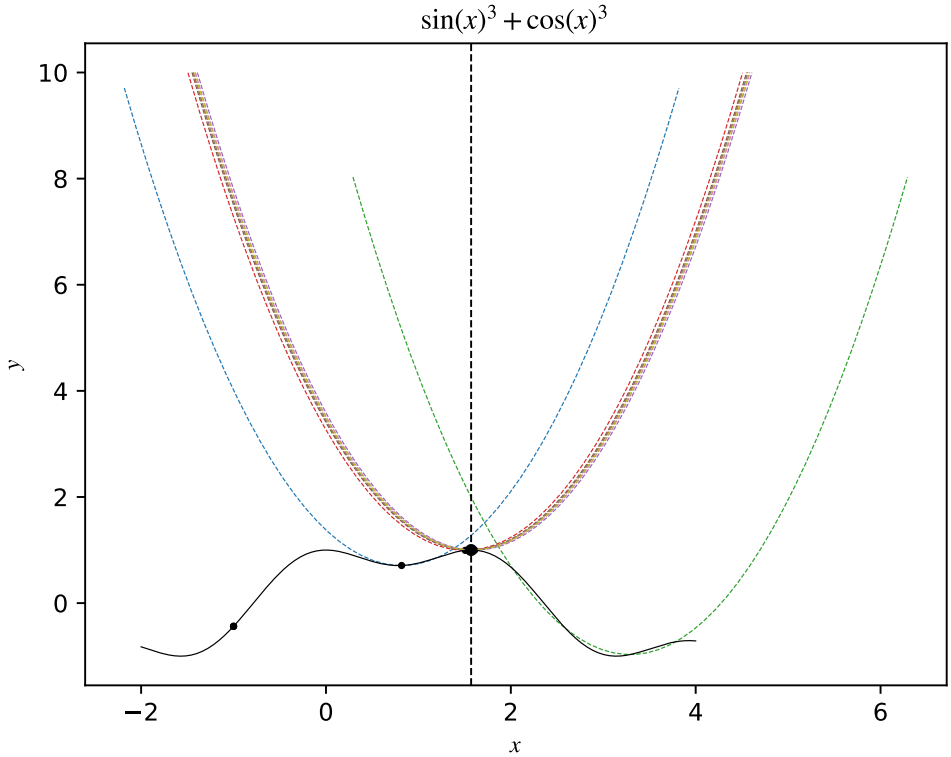
dok ne bude $|d_s - d| < \varepsilon$;

vrati d;

Iako može postići veću brzinu konvergencije nego algoritam zlatnog reza, ovaj postupak ipak ima nedostataka. Prvi nedostatak je osjetljivost na točnost određivanja tjemena parabole po izrazu 2.62; naime, ukoliko su tri početne točke kolinearne, tjeme parabole teži beskonačnosti, pa će nazivnik toga izraza težiti vrijednosti nula. Čak i kada su točke samo približno kolinearne, mali iznos u nazivniku može, zbog ograničene preciznosti prikaza vrijednosti na računalu, polučiti velike pogreške u interpolaciji, pa ovaj postupak može biti numerički nestabilan. Osim toga, izraz koji se koristi za određivanje tjemena parabole ne daje nužno položaj minimuma, već jednako tako i maksimuma kvadratne funkcije, ovisno o odnosu funkcijskih vrijednosti u polaznim točkama. Primjerice, pokrenemo li program **??** za funkciju $\sin(x)^3 + \cos(x)^3$ uz granice intervala $[-1, 3]$, postupak će konvergirati k lokalnom maksimumu funkcije kao što je prikazano na slici 2.12. Također, za razliku od metode zlatnog reza, kvadratna interpolacija može konvergirati u točku izvan granica početnog intervala, što može predstavljati problem ako je potrebno zadovoljiti zadana ograničenja. Iz navedenih razloga, postupak kvadratne interpolacije se nikada ne koristi samostalno! Umjesto toga, moguće je oblikovati postupak koji će koristiti kvadratnu interpolaciju kada je to moguće (tj. samo ako time ostvarujemo napredak u redukciji intervala), ali se i poslužiti sigurnim postupkom zlatnog reza kada pretpostavke kvadratne interpolacije nisu ispunjene. Na toj ideji temelji se algoritam opisan u sljedećem odjeljku.



Slika 2.11: Pronalaženje minimuma funkcije kvadratnom interpolacijom



Slika 2.12: Konvergencija kvadratne interpolacije prema lokalnom maksimumu

2.6 Newtonova metoda

Kao što je pokazano kod metode bisekcije, ako je moguće izračunati derivacije funkcije, to nam može dati dodatne informacije koje mogu ubrzati konvergenciju postupka. Postavlja se pitanje, može li korištenje derivacija višeg reda dati dodatne informacije koje se mogu iskoristiti kako bi se definirao postupak s još boljom konvergencijom. Uprvao jedna takva metoda je i Newtonova metoda koja koristi prve i druge derivacije neke funkcije. Naime, ideja Newtonove metode jest da se jednodimenzijska funkcija aproksimira kvadratnom funkcijom, jer onda analitički možemo izračunati minimum takve funkcije i pomaknuti se u njen minimum. Ako je funkcija slična kvadratnoj funkciji, aproksimacija će biti dobra i špomaknut ćemo se jako blizu minimuma.

$$q(x) = ax^2 + bx + c \tag{2.76}$$

$$2a = f''(x) \tag{2.77}$$

$$a = \frac{f''(x)}{2} \tag{2.78}$$

$$2ax + b = f'(x) \tag{2.79}$$

$$f''(x)x + b = f'(x) \tag{2.80}$$

$$b = f'(x) - f''(x)x \tag{2.81}$$

$$\frac{f''(x)}{2}x^2 + (f'(x) - f''(x)x) + c = f(x) \tag{2.82}$$

Minimum kvadratne funkcije možemo dobiti deriviranjem i izjednačavanjem s 0:

$$q'(x) = 0 \tag{2.83}$$

$$2ax + b = 0 \tag{2.84}$$

$$x = -\frac{b}{2a} \tag{2.85}$$

Dakle, minimum kvadratne funkcije se može izraćunati preko izraza xxx. Ako sada uvrstimo u ovaj izraz vrijednosti za a i b koje smo dobili za našu aproksimiranu funkciju, onda dobivamo

$$x_{k+1} = -\frac{b}{2a} = -\frac{f'(x) - f''(x)x}{f''(x)} = x_k - \frac{f'(x)}{f''(x)} \tag{2.86}$$

POGLAVLJE 3

Optimizacija funkcija više varijabli bez uporabe derivacija

U prethodnom poglavlju opisani su osnovni postupci za ograničavanje i redukciju unimodalnog intervala koji su izravno primjenjivi samo u optimizaciji funkcije jedne varijable. Iako podrazumijevaju samo jednu varijablu, prethodno opisani postupci mogu se iskoristiti i u optimizaciji funkcija više varijabli. Ideja takvog pristupa je da algoritam optimizacije sadrži dvije komponente: jedna komponenta određuje *smjer* pretraživanja u višedimenzijskom prostoru, dok druga pronalazi optimum funkcije u zadanom smjeru.

Pretpostavimo da je zadana početna točka pretraživanja u n-dimenzijskom prostoru (domeni funkcije n varijabli) $\boldsymbol{x_0}$. Neki postupak optimizacije mogao bi temeljem svoje interne logike odabrati smjer $\boldsymbol{v}$ kao prikladan smjer za kretanje prema optimumu iz početne točke $\boldsymbol{x_0}$. Kretanje u zadanom smjeru iz fiksne početne točke zapravo je kretanje na pravcu u višedimenzijskom prostoru, gdje bilo koja točka na pravcu poprima oblik

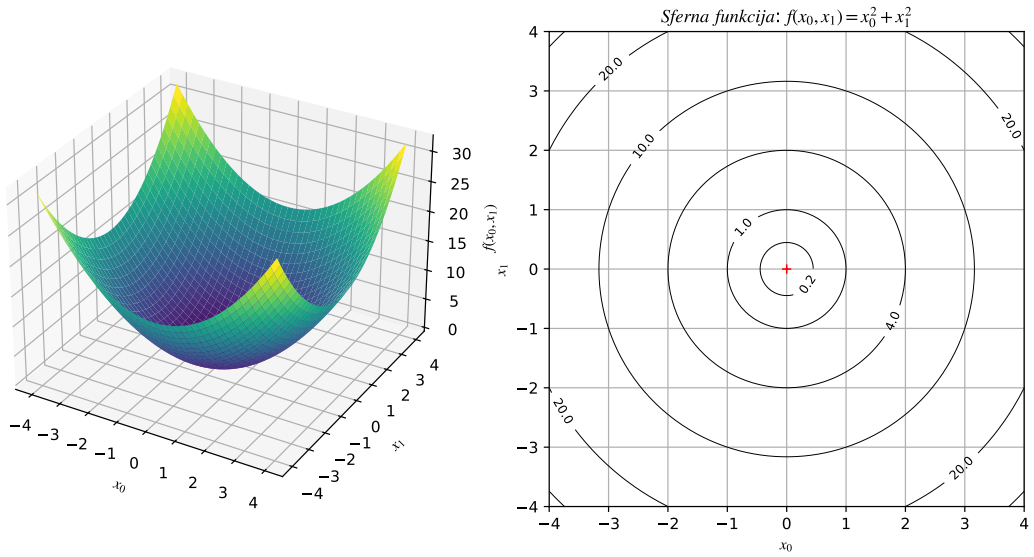
$$\boldsymbol{x} = \boldsymbol{x}_0 + \lambda \cdot \boldsymbol{v}, \tag{3.1}$$

gdje λ određuje udaljenost na pravcu od početne točke $\boldsymbol{x_0}$. Pronalaženje optimuma funkcije više varijabli $f(\boldsymbol{x})$ na zadanom pravcu svodi se na optimzaciju funkcije oblika $f(\boldsymbol{x_0} + \lambda \cdot \boldsymbol{v})$. Primijetimo da je u ovom obliku jedina promjenjiva vrijednost λ, pa se pronalaženje minimuma funkcije na pravcu svodi na minimizaciju *jedne* varijable:

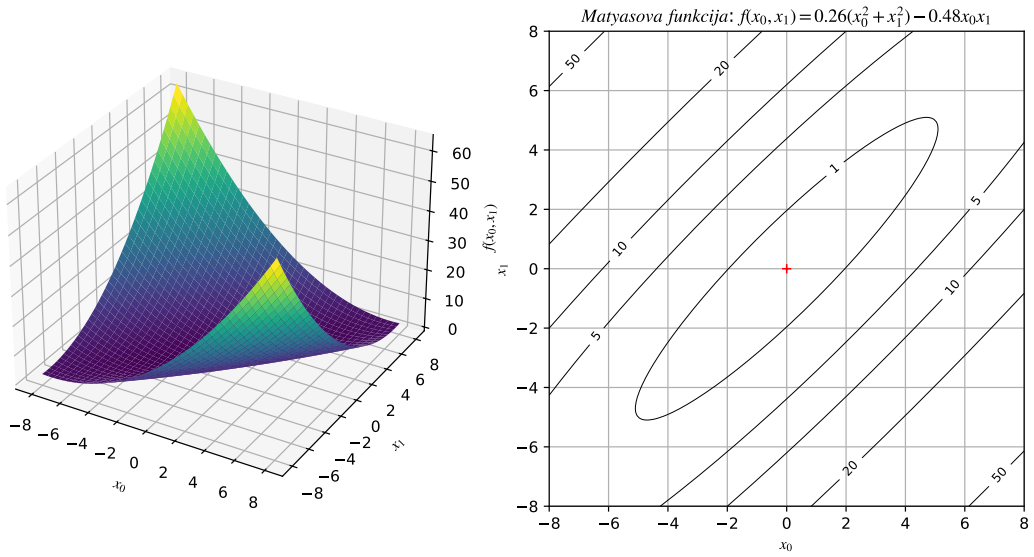
$$\lambda = \arg\min_{\lambda \in R} f(\boldsymbol{x}_0 + \lambda \cdot \boldsymbol{v}) \tag{3.2}$$

Budući da se optimira samo jedna nadomjesna varijabla - udaljenost λ - za pronalaženje optimuma na pravcu može se primijeniti bilo koji postupak optimizacije funkcije jedne varijable; naravno, uz prethodno pronalaženje početnog unimodalnog intervala nadomjesne varijable λ. U ovom se kontekstu postupak optimiranja pomaka λ naziva još i linijsko pretraživanje. Na prethodno opisanoj ideji temelje se mnogi postupci optimizacije više varijabli: iz trenutne točke na neki prikladan način odabire se pogodan smjer pretraživanja (primjerice, uz pomoć gradijenta funkcije), pronalazi se točka koja predstavlja optimum na pravcu određenom tim smjerom, a potom odabir novog smjera započinje iz pronađene točke.

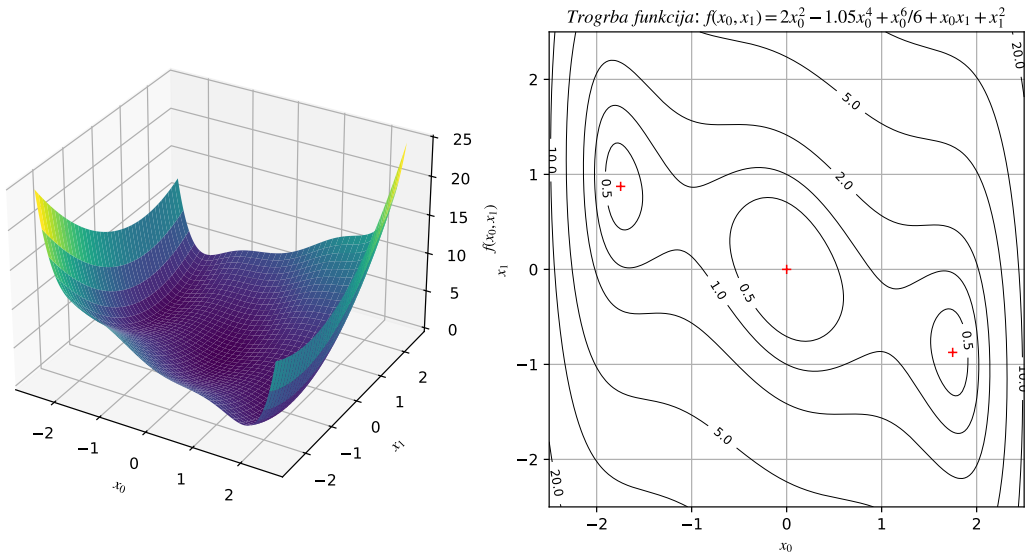
Rosenbrockova funkcija, prikazana na slici 3.2, jedna je od najpoznatijih ispitnih funkcija za procjenu učinkovitosti optimizacijskih postupaka. Funkcija je unimodalna (no nije konveksna), a do minimuma se dolazi kretanjem po uskoj zakrivljenoj "dolini" koja je vrlo blagog nagiba u odnosu na iznos funkcije izvan toga područja. Zbog specifičnog zakrivljenog oblika, ova se funkcija popularno naziva i "banana funkcija" te je često korištena u ilustracijama rada algoritama optimiranja.



(a) Sferna funkcija

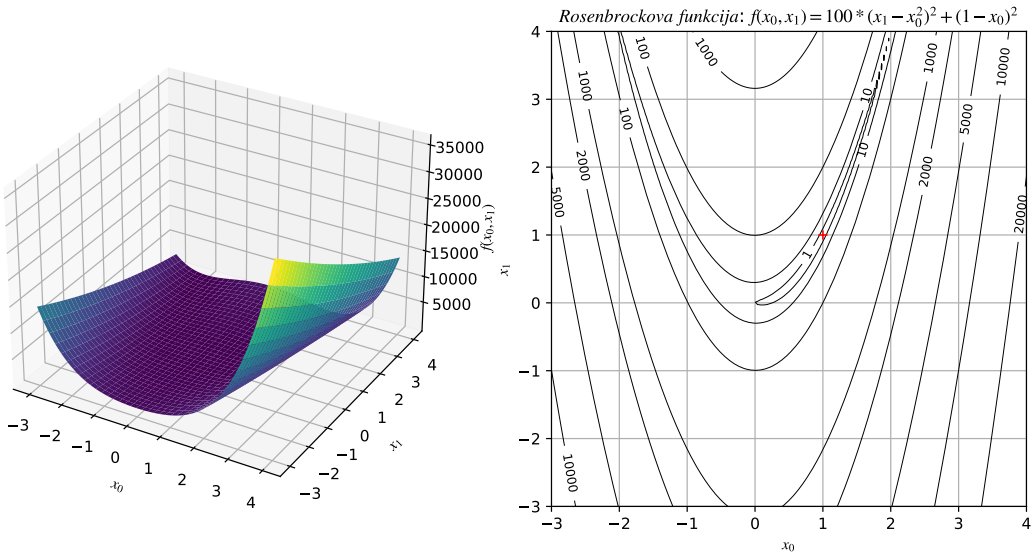


(b) Matyasova funkcija



(c) Trogrba funkcija

Slika 3.1: Primjeri ploha i kontura funkcija dvije varijable



Slika 3.2: Rosenbrockova funkcija

3.1 Koordinatno pretraživanje

Metoda koordinatnog pretraživanja je najjednostavniji primjer primjene prethodno opisanog pristupa optimiranja po nekom pogodnom smjeru u kojemu očekujemo smanjenje funkcije cilja. No umjesto specifično odabranog smjera, u ovoj metodi pretraživanje se provodi uzastopno po smjerovima svih koordinathih osi. Pretraživanje po koordinatnoj osi i zapravo se svodi na optimizaciju i-te komponente točke $\boldsymbol{x}$, odnosno varijable x_i. U svakoj iteraciji postupak iterira sve koordinatne osi i pomiče se nezavisno po svakoj osi za optimalni iznos određen minimizacijom izraza 3.2.

U svakoj iteraciji, postupak optimira funkciju u smjeru svake koordinatne osi te traži optimalni pomak za koji će se pomaknuti u tom smjeru. Prilikom optimiranja funkcije po jednoj od koordinathih osi, optimimum po toj osi nalazi se korištenjem nekog od ranije opisanih postupaka linijskog pretraživanja. Nakon što je pronađen optimalni iznos pomaka, trenutna točka se ažurira na sljedeći način:

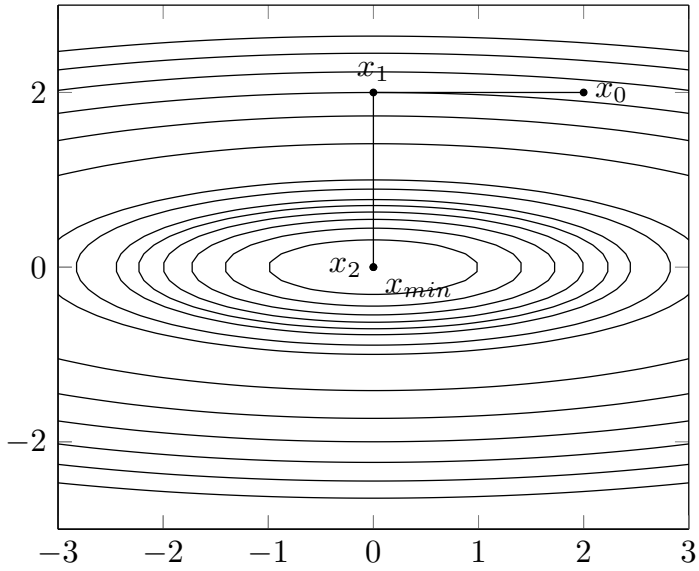
$$\lambda_i = \underset{\lambda_i \in R}{\arg\min}\, f(x_0^{k+1}, x_1^{k+1}, \ldots, x_i^k, \ldots, x_{n-1}^k) \tag{3.3}$$

$$x_i^{k+1} = x_i^k + \lambda_i \tag{3.4}$$

gdje $\boldsymbol{x}$ predstavlja točku u n-dimenzijskom prostoru, i predstavlja indeks varijable koja se optimira, k predstavlja trenutnu iteraciju postupka, a λ_i pomak koji je potrebno napraviti po osi i. Prilikom optimizacije varijable x_i, za sve prethodne varijable $x_j, j < i$ koriste se prethodno određene vrijednosti, koje su u izrazu označene indeksom $k+1$. Nakon varijable x_i pojavljuju se ostale varijable čije vrijednosti još nisu bile optimirane u ovoj iteraciji, te se za njih koriste trenutne vrijednosti.

Na kraju svake iteracije, nakon što je postupak prošao sve koordinatne osi, provjerava se je li postignut napredak s obzirom na početnu točku u trenutnoj iteraciji, najčešće provjeravanjem udaljenosti početne i krajnje točke u jednoj iteraciji. Ako je postignut dovoljno veliki napredak, optimizacija se nastavlja iz točke dobivene na kraju iteracije.

Primjer rada koordinatnog pretraživanja prikazan je na slici 3.3. Postupak kreće iz točke $\boldsymbol{x_0}$ te pretaživanjem po osi apscisa pomiče u točku $\boldsymbol{x_1}$ u kojoj se nalazi minimum u na tom pravcu. Nakon toga, postupak iz točke x_1 provodi pretraživanje po osi ordinate te dolazi do točke x_2. Kako se radi o dvodimenzijskoj funkciji, točka x_2 predstavlja rezultat prve iteracije algoritma (jer je po svakoj osi



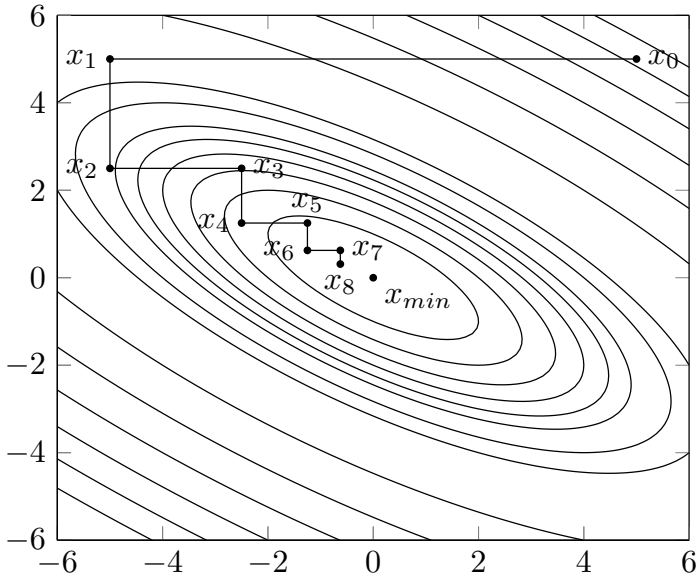
Slika 3.3: Pretraživanje po koordinatnim osima nad funkcijom $f(x_0, x_1) = x_0^2 + 10 \cdot x_1^2$ iz početne točke $(2, 2)$

obavljeno jedno pretraživanje). Za ovu funkciju može se vidjeti kako je postupak u samo jednoj iteraciji došao do minimuma. No takvo ponašanje posljedica je izgleda funkcije cilja, odnosno činjenice da su osi elipsa, koje predstavljaju konture funkcije, poravnate s koordinatnim osima. Ova situacija svojstvo je funkcija cilja izgrađenih od dvije odvojive (separabilne) komponente $(x_1^2, 10 \cdot x_2^2)$ koje ovise samo o pojedinoj varijabli.

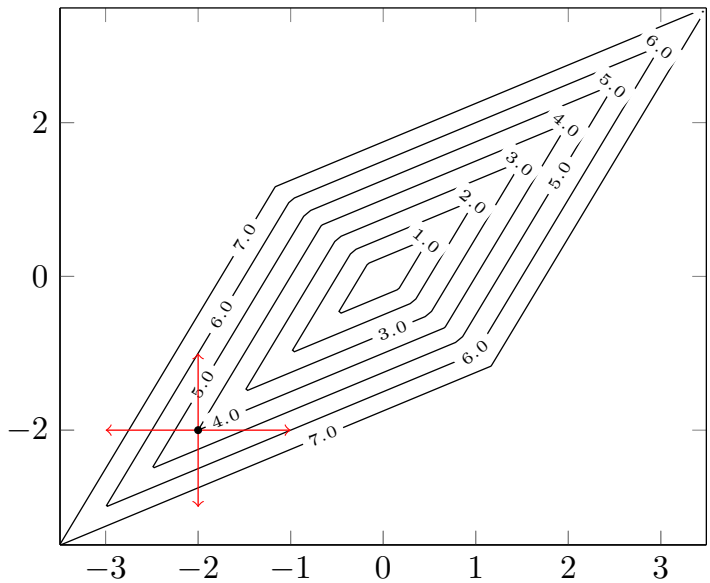
No ako postoji međuovisnost među varijablama, kao kod funkcije cilja na slici 3.4, tada je vidljiva neučinkovitost postupka koordinatnog pretraživanja. U ovom slučaju problem nastaje zbog člana $x_1 \cdot x_2$ koji ovisi o obje varijable, pa optimiranjem svake varijable zasebno nije moguće jednostavno doći do minimuma funkcije. U svakoj iteraciji postupak se približava minimumu funkcije, no može se vidjeti kako su u svakoj iteraciji pomaci sve manji i manji. Zbog nagnutosti kontura funkcije cilja (koje sada nisu poravnate s koordinatnim osima, pa time i smjerovima pretraživanja), postupak radi sve manje korake što se više približava minimumu. Iako za ovu funkciju postupak relativno brzo dolazi blizu minimuma, što je funkcija izduženija po nekoj osi, postupku će biti potreban sve veći broj iteracija kako bi se približio točki minimuma.

Osim relativno spore konvergencije, moguće su i situacije u kojima postupak neće moći niti započeti pretragu, odnosno ostat će zaglavljen u početnoj točki. Takav primjer prikazan je na slici 3.5; u ovom slučaju radi se o funkciji koja nije glatka i čije su konture oštre zbog apsolutne vrijednosti. Ako postupak započne pretragu iz točke $\boldsymbol{x_0} = (-2, -2)$, neće biti moguće pomaknuti se iz te točke. Naime, ako pogledamo smjerove pretraživanja ovog postupka (označeni crvenim strelicama), možemo vidjeti da pomak u bilo kojem od tih smjerova za bilo koji iznos uzrokuje *porast* funkcije cilja. Zbog toga postupak ostaje u početnoj točki te, zbog nemogućnosti pomaka, završava u istoj iteraciji.

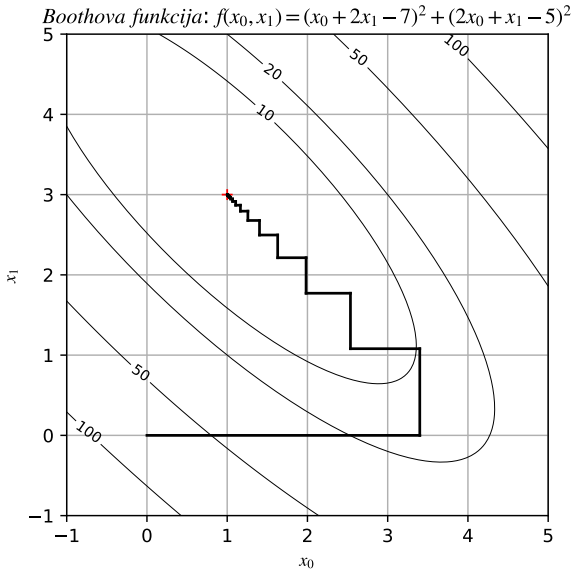
Algoritam 3.1 prikazuje korake metode pretraživanja po koordinatnim osima. U svakoj iteraciji algoritma provodi se linijsko pretraživanje po svakoj od osi koordinatnog sustava. Za svaku os se proizvoljnim postupkom linijskog pretraživanja određuje iznos pomaka λ za koji se dolazi do minimuma na toj osi. Nakon toga se trenutna točka ažurira translacijom za navedeni pomak u smjeru osi. Nakon obavljenog pretraživanja po svim osima, ispituje se kriterij zaustavljanja. Ako je udaljenost između početne



Slika 3.4: Pretraživanje po koordinatnim osima nad funkcijom $f(x_0, x_1) = 0.5x_0^2 + x_1^2 + x_0x_1$



Slika 3.5: Pretraživanje po koordinatnim osima nad funkcijom $f(x_0, x_1) = max(|x_0|, |x_1|)$



Slika 3.6: Primjena postupka koordinatnog traženja na Boothovu funkciju cilja

točke i točke dobivene nakon pretraživanja po svim osima manja od zadane preciznosti ε, postupak se prekida; u suprotnom, ponavlja se pretraživanje po svim osima dok uvjet ne postane zadovoljen.

Algoritam 3.1: Metoda koordinatnog pretraživanja

Function `KoordinatnoPretrazivanje`(f, x_0, ε)**:**
  Podatci: f: funkcija koja se minimizira
    x_0: početna točka
    ε: željena preciznost unimodalnog intervala
  $x = x_0$;
  ponavljaj
    $x_s = x$;
    za $i = 1, 2, \ldots, n$ **čini**
      $argmin_\lambda(F(x + \lambda e_i))$;
      $x = x + \lambda * e_i$;
  dok ne bude $|x - x_s| > \varepsilon$;
  vrati x;

3.2 Powellova metoda konjugiranih smjerova

Neki od navedenih nedostataka prethodnog postupka mogu se ispraviti pametnijim izborom smjerova u kojima se provodi minimizacija na pravcu. U Powellovoj metodi, smjerovi pretraživanja na pravcu ažuriraju se u svakoj iteraciji, na temelju napretka u prethodnim iteracijama, kako bi se pretraga što bolje prilagodila zadanoj funkciji cilja. Kao i u postupki kvardatne interpolacije, gdje se pretpostavlja da je oblik funkcije cilja nalik kvadratnoj funkciji, i Powellova metoda konjugiranih smjerova temelji se na pretpostavci da se funkcija cilja ponaša približno kvadratnoj te na dva svojstva kvadratne funkcije opisana u nastavku.

Opći oblik kvadratne funkcije u n varijabli opisan je izrazom

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^t \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^t \boldsymbol{x} + c, \tag{3.5}$$

gdje matrica $\boldsymbol{A}$, vektor $\boldsymbol{b}$ i skalarna vrijednost c definiraju određenu kvadratnu funkciju. Prvo svojstvo kvadratne funkcije, koje se koristi u Powellovom postupku, tvrdi kako se pretraživanjem u n *konjugiranih* smjerova, može od bilo koje početne točke doći do minimuma funkcije. Dva vektora su međusobno konjugirana s obzirom na matricu $\boldsymbol{A}$ zadane kvadratne funkcije ako vrijedi

$$\boldsymbol{v}_i^t \boldsymbol{A} \boldsymbol{v}_j = 0. \tag{3.6}$$

Drugim riječima, za neki skup n vektora kažemo da su međusobno konjugirani ako za svaki par vektora v_i i v_j, $i = 0, 1, \ldots, n-1$, $j = 0, 1, \ldots, n$ i $i \neq j$ vrijedi prethodni izraz.

Prethodno spomenuto svojstvo govori nam kako je pronalaženje minimuma kvadratne funkcije moguće određivanjem n konjugiranih vektora i uzastopnim pretraživanjem minimuma na pravcu definiranom po svakom od njih. Ovo svojstvo možemo ilustrirati na sljedeći način. Pretpostavimo da krećemo iz točke $\boldsymbol{x}_0$ i pomičemo se u iduću točku $\boldsymbol{x}_1$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \lambda_1 \boldsymbol{v}_1, \tag{3.7}$$

gdje v_1 predstavlja prvi smjer pretraživanja a λ_1 iznos kojim je potrebno pomnožiti taj smjer kako bi došli do minimuma na pravcu. Svaku sljedeću točku određujemo tako da se iz trenutne točke $\boldsymbol{x}_i$ pomaknemo u smjeru $\boldsymbol{v}_{i+1}$ za optimalan iznos λ_{i+1},

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \lambda_{i+1} \boldsymbol{v}_{i+1}. \tag{3.8}$$

Konačno, posljednju točku pretraživanja dobivamo kao

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \lambda_n \boldsymbol{v}_n. \tag{3.9}$$

Ovo možemo i drugačije zapisati na način da na početnu točku nadodamo sve pomake

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i. \tag{3.10}$$

Prvo spomenuto svojstvo kvadratne funkcije tvrdi da je točka $\boldsymbol{x}_n$ minimum ako za sve smjerove $\boldsymbol{v}_i$ vrijedi da su međusobno konjugirani. Kako bismo to pokazali, kvadratnu funkciju možemo razviti u Taylorov red:

$$f(\boldsymbol{x} + \Delta \boldsymbol{x}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \Delta \boldsymbol{x} + \frac{1}{2} \Delta \boldsymbol{x}^T \boldsymbol{H} \Delta \boldsymbol{x} \tag{3.11}$$

Budući da je funkcija kvadratna, vrijedi da je $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ i $\boldsymbol{H} = \boldsymbol{A}$. Također znamo da je u koraku i pomak $\Delta \boldsymbol{x}$ definiran kao $\Delta \boldsymbol{x}_i = \lambda_i \boldsymbol{v}_i$. Prema tome, prethodni izraz možemo zapisati kao

$$f(\boldsymbol{x}_i + \lambda_{i+1} \boldsymbol{v}_{i+1}) = f(\boldsymbol{x}_i) + \lambda_{i+1} (\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b})^T \boldsymbol{v}_{i+1} + \lambda_{i+1}^2 \frac{1}{2} \boldsymbol{v}_{i+1}^T \boldsymbol{A} \boldsymbol{v}_{i+1} \tag{3.12}$$

Ako prethodni izraz deriviramo po λ_{i+1} i izjednačimo s 0, dobivamo

$$(\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b})^T \boldsymbol{v}_{i+1} + \lambda_{i+1} \boldsymbol{v}_{i+1}^T \boldsymbol{A} \boldsymbol{v}_{i+1} = 0 \tag{3.13}$$

Sada možemo izraziti λ_{i+1} i dobiti analitički izraz za određivanje optimalnog iznosa pomaka

$$\lambda_{i+1} = -\frac{(\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b})^T \boldsymbol{v}_{i+1}}{\boldsymbol{v}_{i+1}^T \boldsymbol{A} \boldsymbol{v}_{i+1}} \tag{3.14}$$

Budući da općenito vrijedi $\boldsymbol{y}^T \boldsymbol{z} = \boldsymbol{z}^T \boldsymbol{y}$, brojnik gornjeg izraza može se napisati kao

$$(\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b})^T \boldsymbol{v}_{i+1} = \boldsymbol{v}_{i+1}^T (\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b}). \tag{3.15}$$

Izraz 3.14 može se pojednostaviti na sljedeći način: do točke $\boldsymbol{x_i}$ došli smo tako da smo se od početne točke $\boldsymbol{x_0}$ pomaknuli po svim smjerovima $\boldsymbol{v}_j$ za koje vrijedi da je $j \leq i$,

$$\boldsymbol{x}_i = \boldsymbol{x}_0 + \sum_{j=1}^{i} \lambda_j \boldsymbol{v}_j. \tag{3.16}$$

Sada izraz 3.16 možemo uvrstiti u brojnik izraza 3.14 te dobivamo

$$\boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b}) = \boldsymbol{v}_{i+1}^T \left(\boldsymbol{A}(\boldsymbol{x}_0 + \sum_{j=1}^{i} \lambda_j \boldsymbol{v}_j) + \boldsymbol{b} \right). \tag{3.17}$$

Za smjer $\boldsymbol{v}_{i+1}$ znamo da je konjugiran sa svim prethodnim smjerovima $\boldsymbol{v}_j$. To znači da će umnožak $\boldsymbol{v}_{i+1}\boldsymbol{A}\boldsymbol{v}_j$ biti jednak 0 za sve j te u konačnici dobivamo da je brojnik izraza 3.14 jednak

$$\boldsymbol{v}_{i+1}^T\boldsymbol{A}\boldsymbol{x}_0 + \sum_{j=1}^{i} \lambda_j \boldsymbol{v}_{i+1}^T\boldsymbol{A}\boldsymbol{v}_j + \boldsymbol{v}_{i+1}^T\boldsymbol{b} = \boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b}). \tag{3.18}$$

Prethodni izraz nam govori kako za početnu točku $\boldsymbol{x}_0$ nije bitan redoslijed kojim upotrebljavamo konjugirane smjerove, već da će iznos za koji se moramo pomaknuti u nekom konjugiranom smjeru biti jednak bez obzira koristi li se taj konjugirani smjer kao prvi ili zadnji smjer pretrage. Nakon što je iz početne točke $\boldsymbol{x}_0$ obavljeno pretraživanje po svih n konjugiranih smjerova, postupak dolazi do točke $\boldsymbol{x}_n$:

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i = \boldsymbol{x}_0 - \sum_{i=1}^{n} \frac{\boldsymbol{v}_i^T(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b})}{\boldsymbol{v}_i^T\boldsymbol{A}\boldsymbol{v}_i}\boldsymbol{v}_i. \tag{3.19}$$

Spomenuto svojstvo tvrdi da pretraživanjem po konjugiranim smjerovima dolazimo do točke koja predstavlja minimum kvadratne funkcije. Moguće je pokazati kako upravo točka $\boldsymbol{x}_n$ predstavlja minimum funkcije $f(\boldsymbol{x})$. Prvo izraz 3.19 preoblikujemo na način da $\boldsymbol{x}_0$ prebacimo na lijevu stranu:

$$(\boldsymbol{x}_n - \boldsymbol{x}_0) = -\sum_{i=1}^{n} \frac{\boldsymbol{v}_i^T(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b})}{\boldsymbol{v}_i^T\boldsymbol{A}\boldsymbol{v}_i}\boldsymbol{v}_i. \tag{3.20}$$

Ako se obje strane transponiraju i pomnože s desna s $\boldsymbol{A}\boldsymbol{v}_j$, za bilo koji $j \leq n$, dobiva se:

$$(\boldsymbol{x}_n - \boldsymbol{x}_0)^T\boldsymbol{A}\boldsymbol{v}_j = -\left(\sum_{i=1}^{n} \frac{\boldsymbol{v}_i^T(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b})}{\boldsymbol{v}_i^T\boldsymbol{A}\boldsymbol{v}_i}\boldsymbol{v}_i^T \right) \boldsymbol{A}\boldsymbol{v}_j. \tag{3.21}$$

Kako znamo da su svi smjerovi konjugirani, odnosno da vrijedi $\boldsymbol{v}_i^T\boldsymbol{A}\boldsymbol{v}_j = 0$, eliminiraju se svi članovi sume osim onoga za vektor $\boldsymbol{v}_j$

$$(\boldsymbol{x}_n - \boldsymbol{x}_0)^T\boldsymbol{A}\boldsymbol{v}_j = -\frac{\boldsymbol{v}_j^T(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b})}{\boldsymbol{v}_j^T\boldsymbol{A}\boldsymbol{v}_j}\boldsymbol{v}_j^T\boldsymbol{A}\boldsymbol{v}_j. \tag{3.22}$$

Nakon što izraz pokratimo, dobiva se

$$(\boldsymbol{x}_n - \boldsymbol{x}_0)^T\boldsymbol{A}\boldsymbol{v}_j = -\boldsymbol{v}_j^T(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b}), \tag{3.23}$$

odnosno, budući da je matrica $\boldsymbol{A}$ kvadratne funkcije simetrična,

$$\boldsymbol{A}(\boldsymbol{x}_n - \boldsymbol{x}_0) = -(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b}). \tag{3.24}$$

Nakon množenja s $\boldsymbol{A}^{-1}$ s lijeva dobiva se

$$\boldsymbol{x}_n - \boldsymbol{x}_0 = -\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{A}^{-1}\boldsymbol{b}. \tag{3.25}$$

Nakon pokrate, dobivamo poznati izraz za minimum kvadratne funkcije koji glasi

$$\boldsymbol{x}_n = \boldsymbol{A}^{-1}\boldsymbol{b}. \tag{3.26}$$

Iz posljednjeg izraza možemo vidjeti kako točka $\boldsymbol{x}_n$, do koje se dolazi pretraživanjem minimuma u n konjugiranih smjerova, uistinu predstavlja minimum zadane kvadratne funkcije, što je dokaz prvog spomenutog svojstva.

U praktičnoj primjeni određivanje n konjugiranih smjerova nije jednostavno, pogotovo što za proizvoljne funkcije cilja, koje nisu kvadratne, ne znamo elemente matrice $\boldsymbol{A}$ uz pomoć koje bi mogli odrediti skup konjugiranih vektora. Ovdje u igru ulazi drugo svojstvo kvadratne funkcije; ono tvrdi da se smjer konjugiran zadanom smjeru $\boldsymbol{v}_i$ može konstruirati kao spojnicu minimuma koji se nalaze na dva paralelna

pravca u smjeru $\boldsymbol{v}_i$. Ovo se svojstvo može dokazati krenemo li od izraza 3.14 za minimum funkcije na pravcu,

$$\boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b}) + \boldsymbol{v}_{i+1}^T\boldsymbol{A}\lambda_{i+1}\boldsymbol{v}_{i+1} = 0 \tag{3.27}$$

Budući da se pomak do sljedeće točke može izraziti kao

$$\boldsymbol{x}_{i+1} - \boldsymbol{x}_i = \lambda_{i+1}\boldsymbol{v}_{i+1}, \tag{3.28}$$

prethodni izraz postaje

$$\boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b}) + \boldsymbol{v}_{i+1}^T\boldsymbol{A}(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i) = 0 \tag{3.29}$$

odnosno nakon pojednostavljenja

$$\boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_{i+1} + \boldsymbol{b}) = 0. \tag{3.30}$$

Gornji izraz nam pokazuje kako je gradijent u točki $\boldsymbol{x}_{i+1}$ (izraz u zagradi) okomit na smjer $\boldsymbol{v}_{i+1}$ kojim se došlo do te točke, budući da je skalarni produkt jednak nuli. Kako bismo ilustrirali drugo svojstvo, polazimo iz dvije različite točke $\boldsymbol{x}_i'$ i $\boldsymbol{x}_i''$ u smjeru $\boldsymbol{v}_{i+1}$ određujemo minimum na pravcu. Uvrštavanjem u prethodni izraz, za dvije početne točke dobivamo

$$\boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_{i+1}' + \boldsymbol{b}) = 0 \tag{3.31}$$

$$\boldsymbol{v}_{i+1}^T(\boldsymbol{A}\boldsymbol{x}_{i+1}'' + \boldsymbol{b}) = 0 \tag{3.32}$$

Oduzimanjem ove dvije jednadžbe dobiva se

$$\boldsymbol{v}_{i+1}^T\boldsymbol{A}(\boldsymbol{x}_{i+1}'' - \boldsymbol{x}_{i+1}') = 0, \tag{3.33}$$

te uz definiranje novoga smjera kao spojnice točaka minimuma

$$\boldsymbol{v}_{i+2} = \frac{\boldsymbol{x}_{i+1}'' - \boldsymbol{x}_{i+1}'}{\| \boldsymbol{x}_{i+1}'' - \boldsymbol{x}_{i+1}' \|}, \tag{3.34}$$

konačno dobivamo

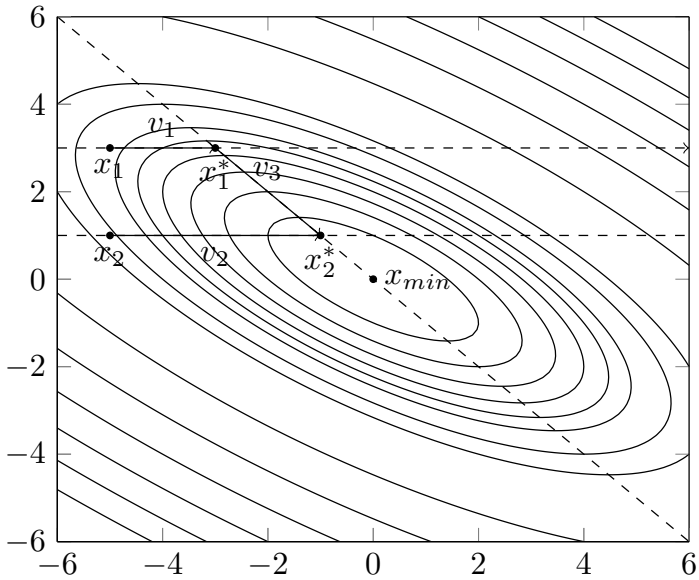
$$\boldsymbol{v}_{i+1}^T\boldsymbol{A}\boldsymbol{v}_{i+2} = 0,$$

što, po samoj definiciji, dokazuje da su smjerovi $\boldsymbol{v}_{i+1}$ i $\boldsymbol{v}_{i+2}$ međusobno konjugirani.

Prethodna dva svojstva ilustrirana su na slici 3.7, koja prikazuje primjer gdje je iz točaka $\boldsymbol{x}_1$ i $\boldsymbol{x}_2$ određen minimum u smjeru $\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, te su tako dobivene točke $\boldsymbol{x}_1^*$ i $\boldsymbol{x}_2^*$. Točkama $\boldsymbol{x}_1^*$ i $\boldsymbol{x}_2^*$ određen je novi smjer $\boldsymbol{v}_2$, koji je konjugiran prethodnom smjeru $\boldsymbol{v}_1$. Osim toga, može se vidjeti kako se, počevši bilo iz točke $\boldsymbol{x}_1$ ili točke $\boldsymbol{x}_2$ te traženjem po smjeru $\boldsymbol{v}_1$ te u smjeru $\boldsymbol{v}_2$, dolazi do minimuma kvadratne funkcije dvije varijable.

Powellova metoda, temeljena na prethodno opisanim svojstvima kvadratne funkcije, može se primijeniti i na funkcije koje nisu kvadratne, uz pretpostavku da se u promatranom području ponašaju približno kvadratno. U slučaju n-dimenzijske funkcije cilja, pretpostavljamo da je zadan početni skup nezavisnih smjerova $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$; u praksi, za početni skup smjerova najčešće se uzimaju smjerovi koordinatnih osi, $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n$. Uz ovaj pristup, Powellov postupak može se opisati kao algoritam 3.2.

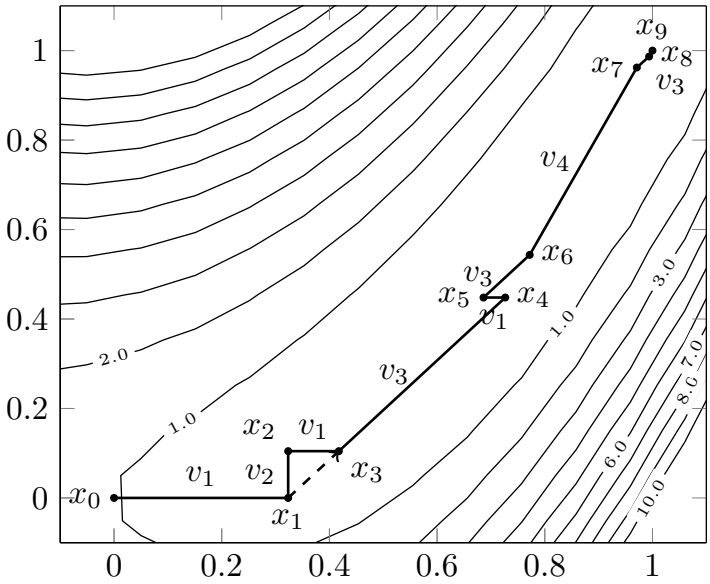
Primjer rada Powellove metode prikazan je na slici 3.8. Početno su zadani smjerovi $\boldsymbol{v}_1$ i $\boldsymbol{v}_2$ koji odgovaraju smjerovima koordinatnih osi. Postupak započinje pretragu iz točke $\boldsymbol{x}_0$ u smjeru $\boldsymbol{v}_1$ te dolazi do točke $\boldsymbol{x}_1$. Ta točka predstavlja minimum na prvom pravcu u smjeru $\boldsymbol{v}_1$. Nakon toga postupak se od točke $\boldsymbol{x}_1$ pomiče u smjeru $\boldsymbol{v}_2$ do točke $\boldsymbol{x}_2$. Ta točka opet predstavlja polazište za pretragu po smjeru $\boldsymbol{v}_1$ kako bi došli do točke $\boldsymbol{x}_3$. Na taj način dobili smo minimume $\boldsymbol{x}_1$ i $\boldsymbol{x}_3$ koji se nalaze na dva paralelna pravca u smjeru $\boldsymbol{v}_1$. Na temelju ta dva minimuma određuje se novi vektor smjera, $\boldsymbol{v}_3$ koji je "konjugiran"



Slika 3.7: Ilustracija dvaju svojstava kvadratne funkcije: 1. minimum kvadratne funkcije može se odrediti pretraživanjem po n konjugiranih smjerova, 2. konjugirani smjer smjeru $\boldsymbol{v}_i$ moguće je dobiti spajanjem minimuma na dva paralelna pravca smjera $\boldsymbol{v}_i$.

Algoritam 3.2: Algoritam Powellovog postupka

Function `Powell(`f, x_0, ε`)`**:**
  Podatci: f: funkcija koja se minimizira
    x_0: početna točka
  za svaki $i \in [1, n]$ **čini**
    $v_i = e_i$;
  $x = x_0$;
  ponavljaj
    $x_s = x$;
    $\lambda = \arg\min_\lambda(f(x + \lambda v_n))$;
    $x_p = x + \lambda v_n$
    za svaki $i \in [1, n]$ **čini**
      $v_i = e_i$;
    za svaki $i \in [1, n]$ **čini**
      $\lambda = \arg\min_\lambda(f(x + \lambda v_1))$;
      $x = x + e_i + \lambda v_i$;
    za svaki $i \in [1, n-1]$ **čini**
      $v_{i+1} = v_i$;
    $v_n = x - x_p$;
  dok ne bude $|x - x_s| < \varepsilon$;
  vrati x;



Slika 3.8: Rad Powellove metode

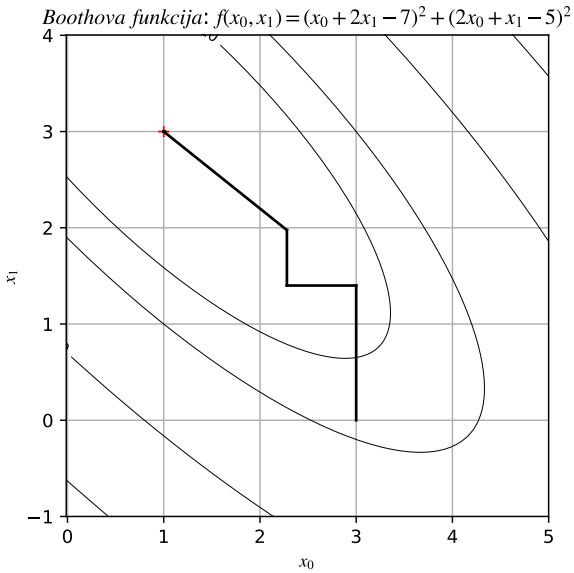
(samo približno, ako funkcija nije kvadratna) vektoru $\boldsymbol{v}_1$. Vektor $\boldsymbol{v}_2$ se odbacuje i umjesto njega se uzima vektor $\boldsymbol{v}_3$, čime završava prva iteracija postupka. U sljedećoj iteraciji postupak iz točke $\boldsymbol{x}_3$ traži minimum u novom smjeru $\boldsymbol{v}_3$ te dolazi u točku $\boldsymbol{x}_4$. Nakon toga se uz pomoć drugog smjera izvodi pomak do točke $\boldsymbol{x}_5$ iz koje se pretraživanjem u smjeru $\boldsymbol{v}_3$ dolazi do točke $\boldsymbol{x}_6$. Točke $\boldsymbol{x}_4$ i $\boldsymbol{x}_6$ predstavljaju minimume na dva paralelna pravca te definiraju novi smjer pretraživanja $\boldsymbol{v}_4$ koji je konjugiran smjeru $\boldsymbol{v}_3$. Postupak se ponavlja dok udaljenost dviju uzastopnih točaka ne bude dovoljno mala, odnosno manja od željene preciznosti ε.

Kada se powellova metoda primjenjuje na funkcije koje nisu kvadratne, može se dogoditi da smjerovi postanu međusobno linearno zavisni. U tom slučaju oni ne razapinju n-dimenzijski prostor te će postupak stagnirati. Zbog toga se, kod optimizacija funkcija koje nisu kvadratne, skup smjerova opetovano postavlja na početne smjerove koordinatnih osi. Na taj način postupak ponovo konstruira konjugirane smjerove te može pretraživati funkciju ako se nije već dovoljno približio točki minimuma. Postoje različite strategije pomoću kojih je moguće odrediti u kojem trenutku je potrebno obnoviti smjerove, a najjednostavniji je da se nakon n iteracija, odnosno kada je konstruiran skup konjugiranih smjerova, ponovo krene od smjerova koordinatnih osi. S druge strane, smjerove je moguće je obnoviti i nakon što iz trenutne točke nije napravljen dovoljno veliki pomak.

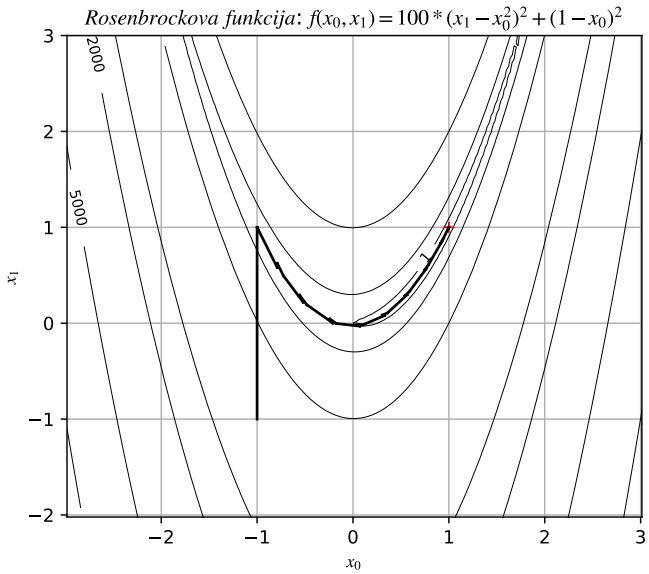
Drugi primjer primjene ovoga postupka možemo vidjeti na slici 3.10 gdje se optimira Rosenbrockova funkcija. Iako ova funkcija predstavlja velik problem za pretraživanje po koordinatnim osima, Powellov postupak u svakoj iteraciji gradi nove smjerove pretraživanja i na taj način vrlo učinkovito dolazi do minimuma u točki $[1, 1]$.

3.3 Rosenbrockova metoda

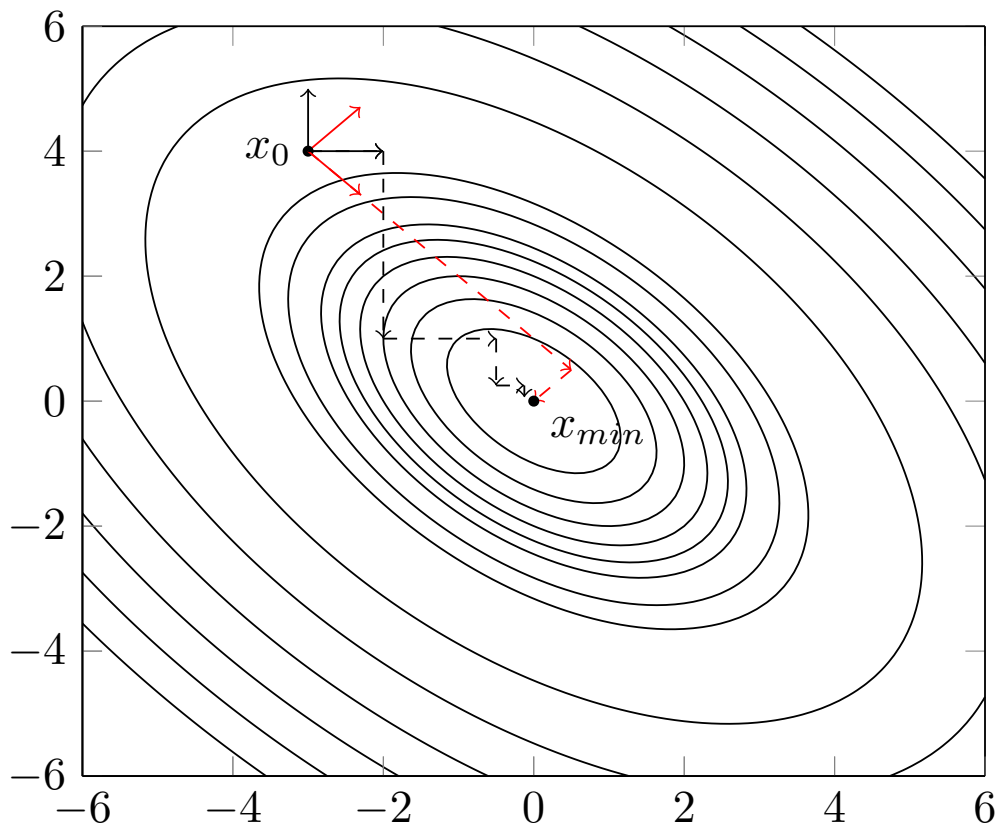
Rosenbrockova metoda rotiranih koordinatnih osi nastoji riješiti problem koji se je pojavio kod pretraživanja po koordinatnim osima. Naime, kod funckija kod kojih konture nisu usmjerene u smjeru neke od osi, odnosno u kojima postoji međuovisnost varijabli, ta je metoda funkcionirala iznimno loše



Slika 3.9: Primjena Powellovog postupka na Boothovu funkciju



Slika 3.10: Primjena Powellovog postupka na Rosenbrockovu funkciju

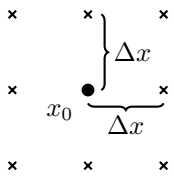


Slika 3.11: Primjer pretaživanja po običnom i rotiranom koordinatnom sustavu

zbog toga. Primjer toga se može vidjeti i na slici 3.11, gje možemo vidjeti da ako su smjerovi pretraživanja zadani u smjeru koordinatnih osi da je postupku potrebno nekoliko koraka da bi se približio minimumu funkcije. No kada bi smjerove koordinatnih osi rotirali, možemo vidjeti kako bi postupak u jednom koraku, pretraživanjem prvo po jednoj pa po drugoj osi, uspio doći do minimuma funkcije. Upravo to i je ideja Rosenbrockove metode.

Rosenbrockova metoda radi na način da se ona prvo pretražuje u smjerovima koordinatnih osi. No, specifičnost Rosenbrockove metode ja ta da se prilikom pretraživanja po smjerovima ne koristi linijsko pretraživanje, već fiksni se koriste fiksni pomaci. Ako je pomak u određenom smjeru bio uspješan, odnosno pronađena je točka s boljom funkcijskom vrijednosti, onda se postupak pomiče u tu točku te se iznos smjera pretrage povečava (predlaže se za faktor 3). U slučaju da nije pronađena točka s boljom funkcijskom vrijednošći, onda se točka ne prihvača i smjer pretrage se moži s negativnim faktorom kako bi se u idućem koraku pretraživanje odvijalu u suprotnom smjeru (obično se množis s faktorom -0.5). Nakon što je napravljen pomak po jednoj osi, prelazi se na drugu os te se postupak pomiče i po njoj. Pvaj postupak ponavlja se dok god po svakoj osi nije zabilježen uspješan pomak praćen (ne nužno odmah) jednim neuspjelim pomakom. Kada se to desi, postupak prekida pretraživanje po osima i prelazi u koram u kojem određuje nove osi po kojima će obavljati pretraživanje.

Nakon što je obavljena pretraga u prvom dijelu postupka. Na temelju informacija koje je postupak prikupio u pretrazi on određuje nove koordinatne osi po kojima će obavljati pretragu. Prvi smjer određuje



Slika 3.12: Točke koje će se potencijalno ispitati u fazi istraživanja

se kao linearna kombinacija ukupnih pomaka koji su napravljeni tijekom pretrage

$$w_1 = \sum_{i=1}^{n} d_i v_i^k \tag{3.35}$$

pri čemu d_i predstavlja ukupni pomak po osi v_i^k koji je napravljen u koraku k. Smjer se još dodatno normalizira i time dobivamo prvu os rotiranog koordinatnog sustava u koraku $k+1$

$$v_1 = \frac{w_1}{||w_1||} \tag{3.36}$$

Nakon što imamo prvu os novog koordinatnog sustava možemo korištenjem Gramm-Schmidt ortogonalizacijkog postupka odrediti sve ostale osi. Ostale osi možemo dobiti preko izraza

$$w_i = v_i^k - \sum_{j=1}^{n-1} v_j^k v_j^{k+1} v_j^{k+1} \tag{3.37}$$

te se te sve osi također moraju onda i normalizirati

$$v_i^{k+1} = \frac{w_i}{||w_i||}. \tag{3.38}$$

Ideja ovog postupka je da se svaka od trenutnih osi oduzme od projekcije te osi na sve do sada izračunate osi novog koordinatnog sustava. Kada se izračunaju sve osi novog koordinatnog sustava, postupak ponovo kreće obavljati pretragu na isti način kao i s prethodnim skupom osi, dok opet ne bude zadoboljen kriterij za određidanje osi novog koordinatnog sustava.

3.4 Postupak Hooke-Jeeves

Za razliku od prethodno opisanih postupaka, u algoritmu po Hookeu i Jeevesu pretraživanje se ne temelji na minimizaciji funkcije na pravcu. Umjesto toga, postupak nastoji odrediti prikladan smjer kretanja na temelju prethodnih koraka i postaviti nove točke uzorkovanja u prostoru pretraživanja. Postupak koristi tri točke: baznu točku $\boldsymbol{x}_B$, početnu točku $\boldsymbol{x}_P$ i novu točku $\boldsymbol{x}_N$. U početku se sve tri točke postavljaju na zadanu početnu točku $\boldsymbol{x}_0$. Svaka iteracija postupka sastoji se od dvije faze: lokalnog istraživanja te kretanja u odabranom smjeru.

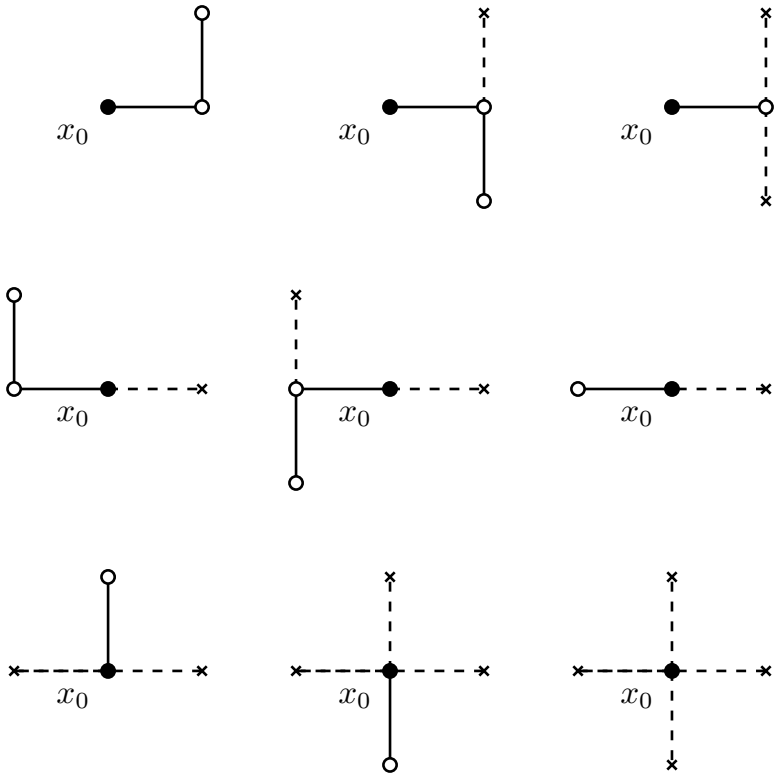
U prvoj fazi se iz početne točke $\boldsymbol{x}_P$ pretražuje njena neposredna okolina. To se obavlja pomicanjem od početne točke za određeni pomak Δx po svakoj od dimenzija (u općenitom slučaju, iznos pomaka može biti i različit za pojedine varijable). Pomak nije određen linijskim pretraživanjem na pravcu, već je njegov iznos zadan na početku postupka u obliku parametra, te se tijekom vremena ažurira. Slika 3.12 prikazuje susjedne točke, označene s križićima, koje je u fazi istraživanja moguće ispitati u slučaju dvodimenzijske funkcije cilja.

Iako potpuno susjedstvo čine 8 točaka, faza istraživanja nikada ne ispituje sve prikazane točke, već se odvija prema algoritmu prikazanom kao algoritam3.3. U fazi pretraživanja, za sve se varijable (osi pomaka) provjerava iznos funkcije cilja prilikom pozitivnog pomaka za Δx, te, ukoliko taj pomak nije rezultirao poboljšanjem, vrijednost funkcije za pomak u suprotnom smjeru. Tek nakon prihvata pomaka

Algoritam 3.3: Algoritam Hooke-Jeeves

```
Function HookeJeeves(f, x_0, Δx, ε):
    Podatci: f: funkcija koja se minimizira
             x_0: početna točka
             Δx: početni pomak po dimenzijama
             ε: željena preciznost unimodalnog intervala
    x_P = x_B = x_0;
    ponavljaj
        x_N = Istrazi(x_P, Δx) ;
        ako F(x_N) < F(x_B) onda
            x_P = 2x_N − x_B ;
            x_B = x_N ;
        inače
            Δx = Δx/2
    dok ne bude Δx < ε;
    vrati x_B;

Function Istrazi(f, x_P, Δx):
    Podatci: f: funkcija koja se minimizira
             x_0: početna točka pretraživanja
             Δx: trenutni pomak po dimenzijama
    za i ∈ [1, n] čini
        P = f(x);
        x_i = x_i + Δx;
        N = f(x);
        ako N > P onda
            x_i = x_i − 2Δx;
            N = f(x);
            ako N > P onda
                x_i = x_i + Δx;
    vrati x;
```



Slika 3.13: Mogući ishodi faze pretraživanja u dvije dimenzije

u jednoj dimenziji (po jednoj osi), istraživanje se nastavlja na isti način u ostalim dimenzijama, a ova faza rezultira novom točkom $\boldsymbol{x}_N$. Faza istraživanja smatra se *uspješnom* ukoliko je vrijednost funkcije cilja u novoj točki $\boldsymbol{x}_N$ strogo manja (odnosno bolja) od one u trenutnoj baznoj točki $\boldsymbol{x}_B$.

Slika 3.13 prikazuje 9 slučajeva koji se mogu dogoditi u fazi pretraživanja za neku dvodimenzionalnu funkciju. Crnim kružićem označena je početna točka x_0, bijelim kružićem točka za koju je prihvaćen pomak, a znakom x točka za koju pomak nije prihvaćen. Pretraživanje se prvo pomiče za iznos Δx po jednoj osi i ispituje funkcijsku vrijednost u toj točki u usporedbi s početnom točkom x_0. Ako je pronađeno bolje rješenje, pomak se prihvaća. Ti su slučajevi prikazani u prvom redu, gdje možemo vidjeti kako se postupak pomaknuo u točku x_1. Čim je pronađena bolja vrijednost funkcije cilja, postupak se prebacuje na iduću (uspravnu) os. Po toj osi pokušava se nadodati Δx i, ako dolazi do točke koja ima bolju funkcijsku vrijednost, postupak je prihvaća i završava pretragu i po drugoj osi. Ovaj slučaj prikazan je u sredini prvog reda, gdje se postupak po prvoj osi pomaknuo u točku x_1 te potom u točku x_2 jer je ona imala bolju funkcijsku vrijednost od točke x_1. U slučaju da točka x_2 nema bolju funkcijsku vrijednost od točke x_1, po istoj se osi ispituje i točka x_3 koja se nalazi za $-\Delta x$ pomak od x_1 (u suprotnom smjeru). Ako točka ima bolju funkcijsku vrijednost od točke x_1 ona se prihvaća (prikazano desno u prvom redu), dok se u suprotnom odustaje od te točke te zadržava točka x_1. Drugi redak prikazuje ishode u kojima je točka x_1 bila lošija od početne točke, te se po istoj osi postupak pomiče za $-\Delta x$ do točke x_2 te, ako je funkcijska vrijednost bolja od početne, iz nje obavlja pretraživanje po uspravnoj osi. Konačno, treći redak prikazuje slučaj kada su po prvoj osi obje ispitane točke bile lošije od početne, u kojem slučaju postupak ostaje u početnoj točki. Naravno, kao što pokazuje zadnji primjer, moguće je da su sve ispitane točke lošije od početne, u kojem slučaju postupak onda ostaje u istoj točki.

Potrebno je napomenuti kako algoritam ne pretražuje cjelokupno susjedstvo zbog uštede u broju

evaluacija, uz pretpostavku optimizacije unimodalne funkcije. Ako se napravi pomak u jednom smjeru i ustanovi da je u tom smjeru funkcijska vrijednost manja, tada po pretpostavci unimodalnosti funkcijska vrijednost u suprotnom smjeru raste. Osim toga, pretraživanje na pohlepni način opisan u algoritmu 3.3 zahtijeva u najgorem slučaju (ako u svakoj dimenziji ispitujemo oba pomaka) $2n$ evaluacija funkcije cilja, što znači da broj evaluacija raste linearno s brojem varijabli. S druge strane, ukoliko bi faza istraživanja provjeravala *cjelokupno* susjedstvo, bilo bi potrebno ukupno 3^n evaluacija funkcije, budući da bi bilo potrebno ispitati dva dodatna pomaka za svaku novu ispitnu točku. U tom slučaju broj ispitivanja raste eksponencijalno, što je izuzetno neučinkovito za relativno veći broj varijabli funkcije cilja.

U drugoj fazi postupka radi se operacija refleksije (kretanja) u smjeru u kojemu je detektirano najveće smanjenje funkcije cilja. Naime, točka $\boldsymbol{x}_N$ pronađena u fazi lokalnog istraživanja pokazuje koji je najbolji smjer u okolici početne točke, odnosno predstavlja grubu aproksimaciju gradijenta. Faza kretanja koristi ovu informaciju kako bi prebacila trenutnu baznu točku $\boldsymbol{x}_B$ preko nove točke $\boldsymbol{x}_N$ i time dobila točku lokalnog pretraživanja za iduću iteraciju. Sljedeća početna točka se računa kao

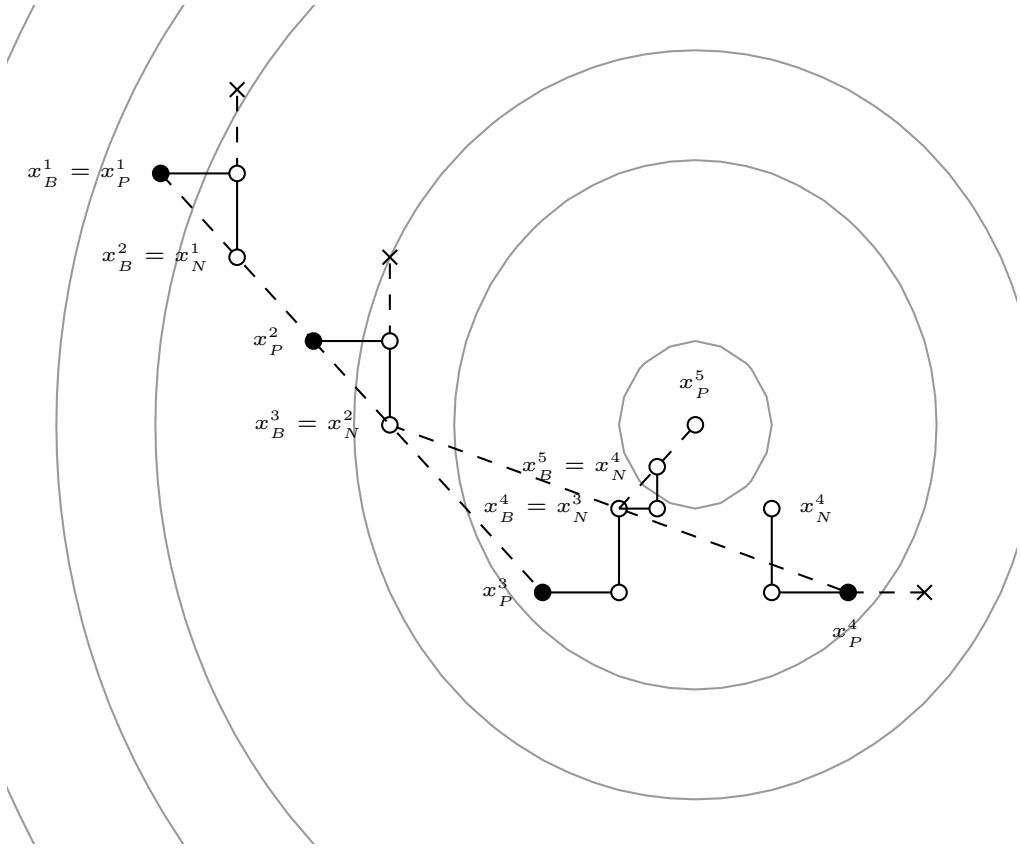
$$\boldsymbol{x}_P = 2\boldsymbol{x}_N - \boldsymbol{x}_B. \tag{3.39}$$

Nakon što je određena sljedeća početna točka, trenutna iteracija završava i postupak se nastavlja fazom lokalnog pretraživanja u sljedećoj iteraciji. Ukoliko je faza pretraživanja u uzastopnim iteracijama uspješna, postupak će u fazi kretanja provoditi sve veće skokove, što omogućava brže kretanje prema udaljenom minmumu.

Međutim, moguće je da faza pretraživanja rezultira točkom $\boldsymbol{x}_N$ koja nije bolja od trenutne bazne točke $\boldsymbol{x}_B$. Uz pretpostavku unimodalnosti, to može značiti ili da smo prethodnom refleksijom već preskočili minimum, ili je lokalni pomak Δx prevelik da omogući približavanje minimumu u fazi istraživanja (u kojem slučaju će pomak uvijek pogoršati vrijednost funkcije). U tom slučaju algoritam *smanjuje pomak* Δx za pola, te postavlja točku $\boldsymbol{x}_P$ na trenutnu baznu točku i nastavlja lokalnim pretraživanjem iz te točke. Na taj način pokušava se ponovo odrediti smjer pretraživanja, no ovaj put s manjim lokalnim pomacima. Ovako opisani postupak se ponavlja dok se pomak Δx ne smanji ispod neke željene preciznosti ε, jer to sugerira da je udaljenost do minimuma reda veličine trenutnog iznosa pomaka.

Slika 3.14 prikazuje primjer optimizacije funkcije dvije varijable korištenjem postupka Hooke-Jeeves. Crnim točkama prikazane su početne točke u svakoj iteraciji, bijelim kružićima točke koje su ispitane u fazi lokalnog istraživanja i koje su prihvaćene. Od početne točke x_p^1 postupak se prvo pomiče za Δx u pozitivnom smjeru osi apscisa, što dovodi do točke s manjom funkcijskom vrijednošću koja se odmah prihvaća. Iz ove točke radi se pretraga po osi ordinata te se za pozitivni pomak dobiva točka s lošijom funkcijskom vrijednosti. Zbog toga se ta točka ne prihvaća nego se ispituje pomak u negativnom smjeru, kojim se dolazi do točke s boljom funkcijskom vrijednošću. Kako smo prošli sve osi, faza istraživanja završava te pronađena točka $\boldsymbol{x}_N^1$ ujedno predstavlja baznu točku u idućoj iteraciji. Budući da je faza istraživanja bila uspješna, provodi se refleksija kojom se trenutna bazna točka prebacuje preko točke $\boldsymbol{x}_N^1$ i time dobiva početna točka $\boldsymbol{x}_P^2$ za iduću iteraciju.

U drugoj iteraciji u fazi istraživanja po osi apscisa dolazi se do bolje točke, dok je po osi ordinata prva ispitna točka bila lošija te je bilo potrebno pomaknuti se u suprotnom smjeru. Nakon toga, opet se izvodi refleksija trenutne bazne točke preko nove točke te se dobiva početna točka iduće iteracije. Ovdje se može primijetiti kako je napravljen veći skok nego u prošloj iteraciji. U trećoj iteraciji je po svakoj dimenziji prvi pomak vodio prema boljoj vrijednosti te se tako dolazi do nove točke $\boldsymbol{x}_N^3$ preko koje se opet obavlja refleksija. U četvrtoj iteraciji u fazi istraživanja postupak dolazi do točke $\boldsymbol{x}_N^4$, koja nema bolju funkcijsku vrijednost od trenutne bazne točke $\boldsymbol{x}_B^4$. Zbog toga se nova točka odbacuje, a bazna točka x_B^4



Slika 3.14: Primjer rada postupka Hooke-Jeeves

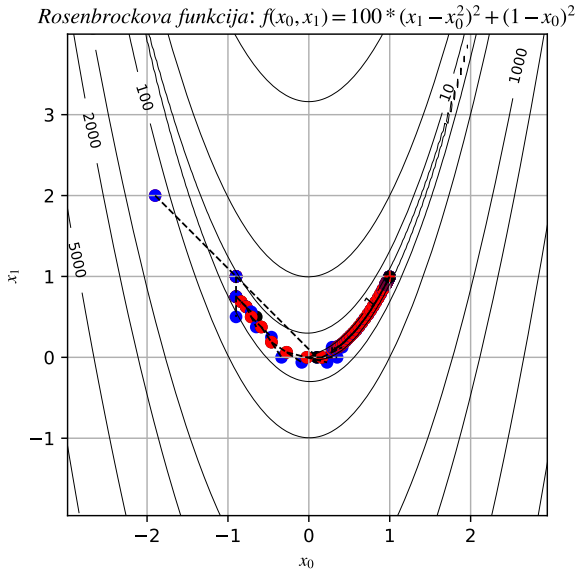
postaje početna točka pretrage u sljedećoj iteraciji. Također, iznos pomaka u fazi istraživanja se smanjuje kako bi pretraga bila preciznija.

3.5 Simpleks postupak po Nelderu i Meadu

Jedan od popularnijih postupaka nelinearne optimizacije, koji se koristi kao prvi izbor u mnogim programskim paketima (primjerice, MATLAB ili `scipy`), jest postupak po Nelderu i Meadu. Iako u svojem nazivu i izvedbi koristi "simpleks", potrebno je napomenuti da u ovom kontekstu to nema nikakve veze s poznatom metodom simpleks u linearnom programiranju.

Ovaj postupak optimizaciju provodi korištenjem skupa točaka koje tvore "simpleks", što je definirano kao n-dimenzijska tvorevina koja se sastoji od $n + 1$ točke. Primjerice, u jednoj dimenziji simpleks su 2 točke koje čine pravac, u dvije dimenzije to su 3 točke koje čine trokut, a u tri dimenzije 4 točke koje definiraju tetraedar. U definiciji simpleksa treba uočiti da odabrane točke čine n-dimenzijsku tvorevinu; primjerice, tri kolinearne točke u dvodimenzijskom prostoru ne tvore simpleks. Osnovna ideja postupka je da se u svakoj iteraciji računa vrijednost funkcije cilja u svakoj točki simpleksa; točka s najlošijom vrijednošću se odbacuje i zamjenjuje novom točkom uporabom jednog od definiranih operatora (npr. refleksijom preko središnje točke simpleksa).

Točke simpleksa moguće je generirati na više načina, no u praksi se obično koristi *pravilan* simpleks, u kojemu su sve točke jednako međusobno udaljene. se obično generira na jedan od dva načina koji su



Slika 3.15: Primjena postupka Hooke-Jeeves na Rosenbrockovu funkciju

prikazani na slici 3.16. Prvi način jest generirati pravilni simpleks kod kojeg su sve točke međusobno jednako udaljene. Konstrukcija pravilnog simpleksa kreće od neke zadane početne točke $\boldsymbol{x}_0$ te stvara n točaka na jednakoj međusobnoj udaljenosti. To je moguće napraviti uz dvije pomoćne vrijednosti:

$$p = \frac{1}{n\sqrt{2}}\left(n - 1 + \sqrt{n+1}\right) \tag{3.40}$$

$$q = \frac{1}{n\sqrt{2}}\left(\sqrt{n+1} - 1\right) \tag{3.41}$$

Na temelju tih vrijednosti, preostalih n točaka simpleksa moguće je konstruirati izrazima

$$(x_i)_j = \begin{cases} x(0)_j + lp, \text{ ako } j = i - 1 \\ x(0)_j + lq, \text{ ako } j \neq i - 1 \end{cases}, \tag{3.42}$$

pri čemu i označava indeks točke koja se generira, j indeks komponente (dimenzije), a l željenu udaljenost između točaka. U postupku je moguće koristiti i nepravilni simpleks; jednostavan simpleks može se dobiti pomakom početne točke u svakoj dimenziji za zadani iznos l, što bi značilo da točke simpleksa generiramo kao

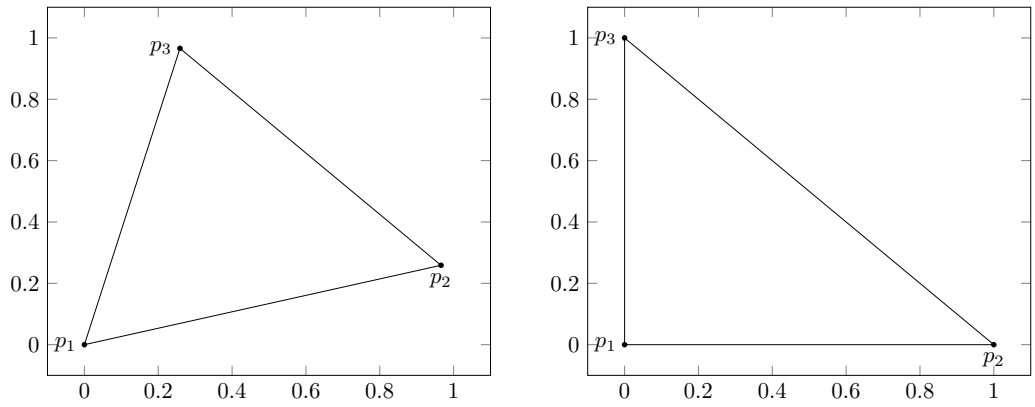
$$(x_i)_j = (x_0)_j + l, \text{ ako } j = i - 1. \tag{3.43}$$

Nakon stvaranja početnog simpleksa, u svaka iteracija postupka započinje određivanjem najbolje točke simpleksa $\boldsymbol{x}_l$ (tj. točke s namanjom vrijednošću funkcije cilja) i najgore točka simpleksa $\boldsymbol{x}_h$ te računanjem centroida svih točaka *osim najgore točke* $\boldsymbol{x}_h$:

$$\boldsymbol{x}_c = \frac{1}{n}\sum_{i \neq h} \boldsymbol{x}_i. \tag{3.44}$$

Nakon računanja centroida, glavni dio algoritma obavlja transformacije nad točkama kako bi se cijeli simpleks što više približio minimumu. Operacije korištene u ovom postupku su refleksija, ekspanzija, kontrakcija i sažimanje.

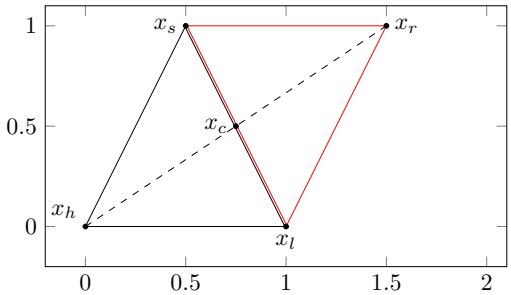
Refleksija je prva operacija koja se obavlja i jedina operacija koja se obavlja bezuvjetno u svakoj iteraciji algoritma. Refleksija izvodi prebacivanje najgore točke simpleksa $\boldsymbol{x}_h$ preko centroida $\boldsymbol{x}_c$, a operacija je prikazana na slici 3.17, gdje $\boldsymbol{x}_r$ predstavlja točku dobivenu refleksijom. Iz geometrijskog prikaza možemo odrediti i izraz za refleksiju; definiramo li omjer udaljenosti točaka $\boldsymbol{x}_r$ i $\boldsymbol{x}_c$ i točaka $\boldsymbol{x}_c$ i



(a) Pravilni simpleks

(b) Nepravilni simpleks

Slika 3.16: Primjer mogućih inicijalnih simpleksa za dvodimenzijski prostor



Slika 3.17: Primjer operacije refleksije

$\boldsymbol{x}_h$ kao faktor α,

$$\frac{x_r - x_c}{x_c - x_h} = \alpha, \tag{3.45}$$

iz toga možemo dobiti izraz za računanje $\boldsymbol{x}_r$ kao

$$\boldsymbol{x}_r = \boldsymbol{x}_c + \alpha(\boldsymbol{x}_c - \boldsymbol{x}_h) \tag{3.46}$$

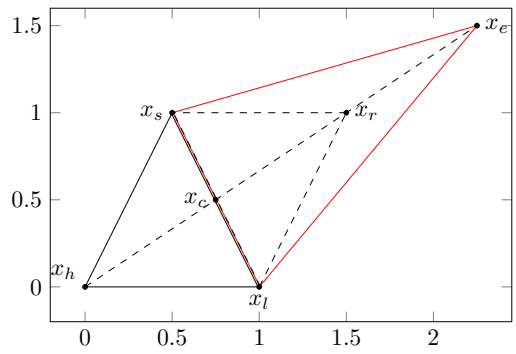
ili jednostavnije

$$\boldsymbol{x}_r = (1 + \alpha)\boldsymbol{x}_c - \alpha\boldsymbol{x}_h. \tag{3.47}$$

Koeficijent α naziva se *koeficijentom refleksije* koji definira na koju udaljenost od centroida $\boldsymbol{x}_c$ će se točka $\boldsymbol{x}_h$ prebaciti. Uobičajena vrijednost tog faktora je 1.

O rezultatu refleksije ovisi ostatak tijeka algoritma. U slučaju da vrijedi $f(\boldsymbol{x}_l) < f(\boldsymbol{x}_h) < f(\boldsymbol{x}_s)$, odnosno da reflektirana točka nije bolja od najbolje točke $\boldsymbol{x}_l$ i da nije gora od druge najlošije $\boldsymbol{x}_s$, točka $\boldsymbol{x}_r$ se prihvaća te se ubacuje u simpleks umjesto točke $\boldsymbol{x}_h$. U ovom bi slučaju novi simpleks imao oblik kao onaj prikazan crvenom linijom na slici 3.17 te bi postupak krenuo u novu iteraciju. U svakom drugom slučaju, postupak nastavlja dalje i izvodi neku od preostalih operacija.

Ako reflektirana točka $\boldsymbol{x}_r$ ima bolju funkcijsku vrijednost čak i od najbolje točke $\boldsymbol{x}_l$, to znači da je postupak pogodio smjer u kojemu se funkcija najviše smanjuje i u kojemu bi se trebao nalaziti minimum. U tom slučaju postupak još više eksploatira taj smjer, odnosno provodi još veći pomak u tom smjeru kako bi pronašao još bolje rješenje. Tada se izvodi operacija *ekspanzije*, koja dodatno pomiče točku $\boldsymbol{x}_r$ u smjeru koji je određen refleksijom. Slika 3.18 prikazuje primjer operacije ekspanzije, gdje $\boldsymbol{x}_e$ predstavlja točku dobivenu ekspanzijom. Izraz za računanje točke ekspanzije $\boldsymbol{x}_e$ možemo odrediti na sličan način kao za



Slika 3.18: Primjer operacije ekspanzije

točku refleksije. Ako omjer udaljenosti između točaka $\boldsymbol{x}_e$ i $\boldsymbol{x}_c$ i točaka $\boldsymbol{x}_r$ i $\boldsymbol{x}_c$ definiramo kao faktor γ

$$\frac{\boldsymbol{x}_e - \boldsymbol{x}_c}{\boldsymbol{x}_r - \boldsymbol{x}_c} = \gamma, \tag{3.48}$$

iz toga možemo dobiti izraz za računanje $\boldsymbol{x}_e$ kao

$$\boldsymbol{x}_e = \boldsymbol{x}_c + \gamma(\boldsymbol{x}_r - \boldsymbol{x}_c) \tag{3.49}$$

ili drugačije zapisano

$$\boldsymbol{x}_e = (1-\gamma)\boldsymbol{x}_c + \gamma\boldsymbol{x}_r. \tag{3.50}$$

Faktor γ naziva se faktorom ekspanzije te određuje koliko će se postupak pomaknuti od točke $\boldsymbol{x}_c$ u smjeru točke $\boldsymbol{x}_r$. Kako se ekspanzijom želimo pomaknuti za još veći iznos u smjeru refleksije, ovaj faktor mora biti veći od faktora refleksije te se obično postavlja na vrijednost 2. Ako dobivena točka $\boldsymbol{x}_e$ ima manju funkcijsku vrijednost od trenutno najbolje točke $\boldsymbol{x}_l$, tada se $\boldsymbol{x}_e$ prihvaća te ubacuje u simpleks umjesto točke $\boldsymbol{x}_h$. U suprotnom dobivena točka nije zadovoljavajuće kvalitete te se u simpleks umjesto točke $\boldsymbol{x}_h$ dodaje točka refleksije $\boldsymbol{x}_r$. Alternativno, moguće je izravno usporediti $\boldsymbol{x}_e$ i $\boldsymbol{x}_r$ te bolju od njih dvije staviti u simpleks. Nakon što je odrađena ekspanzija, algoritam kreće s novom iteracijom.

Naposljetku, posljednji ishod refleksije podrazumijeva da je funkcijska vrijednost točke $\boldsymbol{x}_r$ lošija i od druge najlošije točke u simpleksu $\boldsymbol{x}_s$, što sugerira da smjer refleksije ne vodi k minimumu. Ova situacija se često sreće kada su točke već u blizini minimuma, ali ga se zbog velike međusobne udaljenosti točaka ne može preciznije locirati. Iz tog razloga, provodi se operacija *kontrakcije* kako bi se omogućila preciznija pretraga u području koje je simpleks obuhvatio. Primjer operacije kontrakcije prikazan je na slici 3.19. Kontrakcija se izvodi na dva moguća načina, ovisno o funkcijskim vrijednostima točaka $\boldsymbol{x}_r$ i $\boldsymbol{x}_h$. Ako $\boldsymbol{x}_r$ ima bolju (manju) funkcijsku vrijednost od $\boldsymbol{x}_h$, tada se $\boldsymbol{x}_r$ približava prema centroidu i ovu inačicu često nazivamo i vanjskom kontrakcijom. Izraz za ovu operaciju možemo dobiti na način da zapišemo omjere udaljenosti između točaka $\boldsymbol{x}_k$ i $\boldsymbol{x}_c$ i točaka $\boldsymbol{x}_r$ i $\boldsymbol{x}_k$

$$\frac{\boldsymbol{x}_k - \boldsymbol{x}_c}{\boldsymbol{x}_r - \boldsymbol{x}_c} = \beta, \tag{3.51}$$

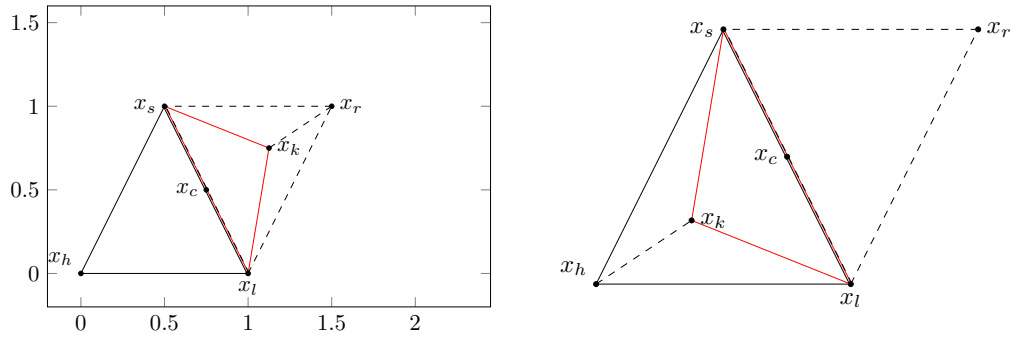
iz čega možemo lako dobiti izraz za računanje $\boldsymbol{x}_r$

$$\boldsymbol{x}_k = \boldsymbol{x}_c + \beta(\boldsymbol{x}_r - \boldsymbol{x}_c), \tag{3.52}$$

ili drugačije zapisano

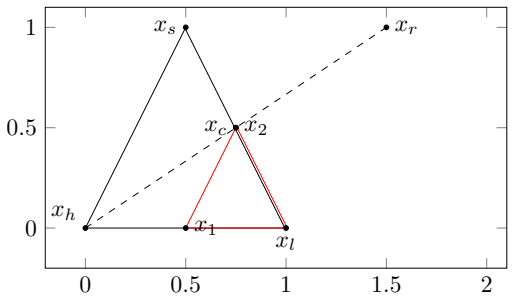
$$\boldsymbol{x}_k = (1-\beta)\boldsymbol{x}_c - \beta\boldsymbol{x}_r. \tag{3.53}$$

U suprotnom, ako točka $\boldsymbol{x}_r$ ima lošiju funkcijsku vrijednost od točke $\boldsymbol{x}_h$, onda nema smisla koristiti točku refleksije jer se ona ionako nalazi u području gdje su funkcijske vrijednosti lošije. U ovom slučaju



(a) Vanjska kontrakcija **(b)** Unutarnja kontrakcija

Slika 3.19: Operacija kontrakcije



Slika 3.20: Operacija sažimanja

postupak provodi kontrakciju nad točkom $\boldsymbol{x}_h$ za što se koristi izraz

$$\boldsymbol{x}_k = \boldsymbol{x}_c + \beta(\boldsymbol{x}_h - \boldsymbol{x}_c), \tag{3.54}$$

ili jednostavnije

$$\boldsymbol{x}_k = (1-\beta)\boldsymbol{x}_c + \beta\boldsymbol{x}_h. \tag{3.55}$$

Funkcijska vrijednost nove točke $\boldsymbol{x}_k$ se potom uspoređuje s vrijednošću točke $\boldsymbol{x}_h$. Ako je dobivena točka s boljom funkcijskom vrijednosti, točka $\boldsymbol{x}_k$ se ubacuje u simpleks umjesto točke $\boldsymbol{x}_h$ i iteracija završava. U suprotnom, točka $\boldsymbol{x}_k$ se odbacuje i obavlja se operacija sažimanja.

Operacija *sažimanja* obavlja se kada niti nakon kontrakcije nije dobivena bolja vrijednost od najgore točke $\boldsymbol{x}_h$. Iako se ta situacija ne događa često, moguća je zbog nepovoljnog rasporeda točaka i izobličenosti funkcije cilja.

Slika 3.20 prikazuje operaciju sažimanja. U toj operaciji se *sve* točke približavaju točki $\boldsymbol{x}_l$ kako bi se područje pretrage usmjerilo na okolicu najbolje točke. Izraz po kojem se obavlja sažimanje je

$$\boldsymbol{x}_j = \boldsymbol{x}_l + \delta(\boldsymbol{x}_j - \boldsymbol{x}_l), \forall j = 0, \ldots, n \text{ i } j \neq l. \tag{3.56}$$

Faktor δ predstavlja faktor sažimanja koji obično poprima vrijednost 0.5. Nakon sažimanja, trenutna iteracija završava i postupak prelazi u novu iteraciju.

U originalnoj inačici algoritma, kriterij zaustavljanja provjerava jesu li funkcijske vrijednosti svih točaka međusobno međusobno dovoljno bliske vrijednosti točke centroida:

$$\sqrt{\frac{1}{n}\sum_{j=0}^{n} n\left(f(\boldsymbol{x}_j) - f(\boldsymbol{x}_c)^2\right)} \leq \varepsilon. \tag{3.57}$$

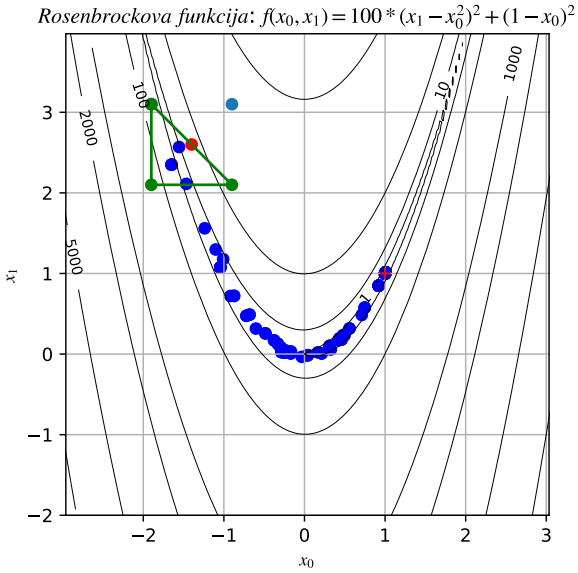
Nažalost, prethodni kriterij zaustavljanja često se pokazuje kao loš odabir; za određene konfiguracije točaka

ovaj uvjet može biti zadovoljen dok smo još relativno daleko od minimuma, pa zbog toga algoritam može preuranjeno završiti s izvođenjem. Zbog toga je uputno provjeravati i udaljenost točaka u simpleksu, primjerice kao udaljenost svih točaka do centroida, te tu mjeru također koristiti kao dodatni kriterij zaustavljanja.

Algoritam 3.4 prikazuje korake simpleks postupka po Nelderu i Meadu.

Algoritam 3.4: Algoritam simpleksa po Neldearu i Meadu

Function `Simpleks`(f, x_0, α, β, γ, δ, Δ ε)**:**
- **Podatci:** f: funkcija koja se minimizira
  - a: početna granica unimodalng intervala
  - x_0: početna točka
  - α: faktor refleksije
  - β: faktor kontrakcije
  - γ: faktor ekspanzije
  - δ: faktor sažimanja
  - δ: željena udaljenost točaka inijcijalnog simpleksa
  - ε: željena preciznost unimodalnog intervala
- $X = \text{GenerirajPocetniSimpleks}(x_0, \Delta)$;
- **ponavljaj**
  - $l = \arg\min_{i\in[1,n+1]}(f(X[i]))$;
  - $h = \arg\max_{i\in[1,n+1]}(f(X[i]))$;
  - $s = \arg\min_{i\in[1,n+1],i\neq h}(f(X[i]))$;
  - $k = \frac{1+\sqrt{5}}{2}$;
  - $x_c = \frac{1}{n}\sum_{i\neq h} x_i.$;
  - $x_r = (1+\alpha)x_c - \alpha x_h$;
  - **ako** $f(x_r) < f(X[l])$ **onda**
    - $x_e = (1-\gamma)x_c - \gamma x_r$;
    - **ako** $f(x_e) < f(X[l])$ **onda**
      - $X[h] = x_e$;
    - **inače**
      - $X[h] = x_r$;
  - **inače**
    - **ako** $f(x_r) < f(X[i]), \forall i = 0,\ldots,n, i \neq h$ **onda**
      - $X[h] = x_r$;
    - **inače**
      - **ako** $f(x_r < f(X[h]))$ **onda**
        - $x_k = (1-\beta)x_c - \alpha x_r$;
      - **inače**
        - $x_k = (1-\beta)x_c - \alpha x_h$;
      - **ako** $f(x_k) < f(X[h])$ **onda**
        - $X[h] = x_k$;
      - **inače**
        - **za svaki** $i \in [0,n], i \neq l$ **čini**
          - $X[i] = \frac{1}{2}(X[i] + X[l])$;
- **dok ne bude** $\sqrt{\frac{1}{2}\sum_{i=1}^{n}((f(x_i) - f(x_c))^2} < \varepsilon$;
- **vrati** $X[l]$;



Slika 3.21: Primjena postupka po Nelderu i Meadu na Rosenbrockovu funkciju

POGLAVLJE 4

Optimiranje uz uporabu gradijenta

U početnom razmatranju pojma optimizacije uveli smo pojam derivacije, odnosno gradijenta funkcije više varijabli, kao pokazatelja brzine promjene vrijednosti funkcije i osnove za definiciju lokalnog minimuma. Prethodno opisani postupci optimizacije nisu uzimali u obzir gradijent funkcije, što ih čini primjenjivima u svim slučajevima kada ta informacija nije dostupna. Međutim, ako je moguće analitički ili numerički odrediti parcijalne derivacije po svim varijablama, tada možemo primijeniti gradijentne postupke optimizacije koji će učinkovitije pretražiti prostor rješenja.

Ako je iznos parcijalnih derivacija dostupan u analitičkom obliku (1.3), tu informaciju možemo iskoristiti u optimizaciji bez većeg dodatnog vremenskog utroška. Međutim, čak i ako gradijent nije dostupan u analitičkom obliku, moguće je korištenjem metode konačnih diferencija aproksimirati prve i druge derivacije funkcije. Prisjetimo se da je derivacija funkcije jedne varijable definirana kao

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{4.1}$$

Vrijednost derivacije funkcije f' u točki x dobivamo malenim pomakom h koji u idealnom slučaju teži prema nuli. Fiksiramo li h na neku konačnu vrijednost, dobivamo izraz za unaprijednu diferenciju:

$$\frac{f(x+h) - f(x)}{h}. \tag{4.2}$$

Slično unaprijednoj diferenciji kod koje je pomak u pozitivnom smjeru, možemo definirati i unazadnu diferenciju kod koje se pomičemo u negativnom smjeru od točke x:

$$\frac{f(x) - f(x-h)}{h}. \tag{4.3}$$

Obje prethodne aproksimacije su metode prvog reda i imaju grešku koja je proporcionalna s $O(h)$. No može se definirati i središnja diferencija kao

$$\frac{f(x+h) - f(x-h)}{2h}, \tag{4.4}$$

čija je greška proporcionalna s $O(h^2)$, što znači da je preciznija od prethodne dvije metode. Parametar h predstavlja pomak od trenutne točke x kojim se aproksimira derivacija. Što je taj pomak manji to je aproksimacija preciznija, no proizvoljno malenu vrijednost nije moguće odabrati zbog numeričkih grešaka tijekom izračuna. Stoga je u praksi potrebno odrediti vrijednost pomaka za koju će rezultati biti dovoljno precizni. Slično kao za prve derivacije, moguće je definirati i izraze konačne diferencije za aproksimaciju drugih derivacija. Unaprijedna konačna diferencija definirana je kao

$$\frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}, \tag{4.5}$$

unazadna konačna diferencija kao

$$\frac{f(x) - 2f(x-h) + f(x-2h)}{h^2} \tag{4.6}$$

te središnja konačna diferencija kao

$$\frac{f(x+h)-2f(x)+f(x-h)}{h^2}. \tag{4.7}$$

Koncept konačnih diferencija može se proširiti i na višedimenzijski slučaj. Tada se konačne diferencije prvih derivacija varijabli x_i i x_j računaju kao

$$f_{x_i}(x_i, x_j) = \frac{f(x_i+h, x_j) - f(x_i-h, x_j)}{2h} \tag{4.8}$$

za prvu derivaciju po varijabli x_i, odnosno kao

$$f_{x_j}(x_i, x_j) = \frac{f(x_i, x_j+k) - f(x_i, x_j-k)}{2k} \tag{4.9}$$

za prvu derivaciju po varijabli x_j. Kao što je iz izraza vidljivo, za svaku varijablu može se koristiti različiti iznos pomaka, primjerice za varijablu x_i pomak h, a za varijablu x_j pomak k. Isti izrazi mogu se proširiti i na druge derivacije, pa se dvostruka derivacija po varijabli x_i može aproksimirati kao

$$f_{x_i x_i}(x_i, x_j) = \frac{f(x_i+h, x_j) - 2f(x_i, x_j) + f(x_i+h, x_j)}{h^2}, \tag{4.10}$$

dvostruka derivacija po x_j kao

$$f_{x_j x_j}(x_i, x_j) = \frac{f(x_i, x_j+j) - 2f(x_i, x_j) + f(x_i, x_j+k)}{k^2} \tag{4.11}$$

te derivacija po varijablama x_i i x_j kao

$$f_{x_i x_j}(x_i, x_j) = \frac{f(x_i+h, x_j+j) - f(x_i+h, x_j) - f(x_i-h, x_j+k) + f(x_i-h, x_j-k)}{4hk}. \tag{4.12}$$

Iako su u prethodnim izrazima naznačene samo dvije varijable, izrazi vrijede za proizvoljan broj varijabli, budući da se u svakom trenutku mijenjaju samo jedna ili dvije varijable za koje se računa derivacija, tako da su sve preostale varijable u tim izrazima konstantne.

Uzmemo li u obzir da je za aproksimaciju parcijalne derivacije po svakoj varijabli potrebno izračunati vrijednost funkcije cilja u jednoj ili više dodatnih točaka, gradijentne metode po svojoj učinkovitosti mogu biti usporedive s izravnim metodama koje ne koriste derivacije. Bez obzira na način dobivanja gradijenta, bilo analitički ili aproksimacijom konačnim diferencijama, sljedeći postupci pretpostavljaju kako je ta informacija dostupna u svakoj iteraciji optimizacije.

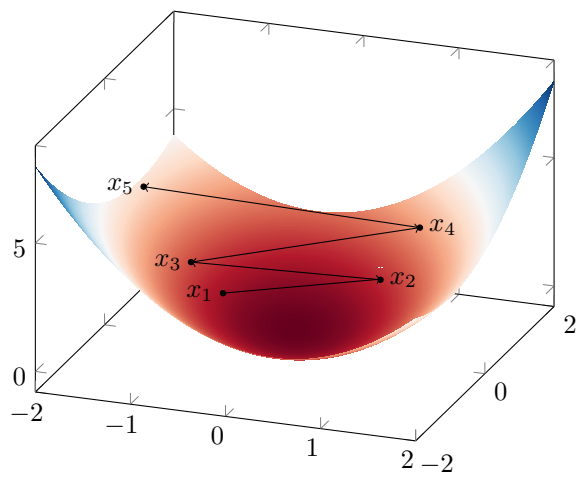
4.1 Gradijentni spust

Gradijentni spust, poznat i kao najbrži spust, predstavlja najjednostavniju metodu optimizacije koja koristi derivacije funkcije. Kao što je ranije spomenuto, gradijent funkcije predstavlja smjer u kojemu funkcija cilja najbrže raste. Trivijalna strategija odabrat će smjer pretrage jednak suprotnom smjeru gradijenta, budući da je to smjer u kojemu vrijednost funkcije najbrže pada. Dakle, u nekoj točki $\boldsymbol{x}_0$ gradijentni spust se pomiče u novu točku $\boldsymbol{x}_n$ korištenjem izraza

$$\boldsymbol{x}_n = \boldsymbol{x}_0 - \lambda \nabla f(\boldsymbol{x}_0), \tag{4.13}$$

gdje λ predstavlja koeficijent kojim se množi iznos gradijenta. Iznos gradijenta $\nabla f(\boldsymbol{x}_0)$ daje nam smjer, no ne daje informaciju za određivanje veličine pomaka u danom smjeru. Pomak jednak iznosu gradijenta nikako ne preporučuje jer najčešće dovodi do oscilacije ili divergencije postupka. Primjer takvog ponašanja prikazan je na slici 4.1 na kojoj se može vidjeti kako je svaka nova točka sve udaljenija od minimuma. Zbog toga se u praksi koristi koeficijent λ kojim se množi gradijent, a iznos koeficijenta se može se odabrati na različite načine.

Jedna mogućnost je da on bude konstantan tijekom čitavog postupka. Prednost ovog pristupa je u



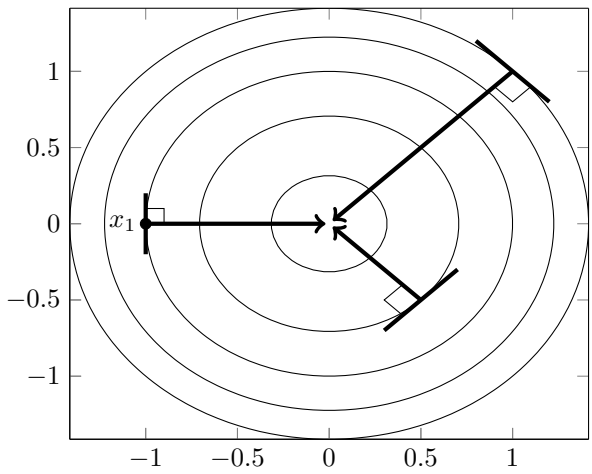
Slika 4.1: Divergencija gradijentnog spusta uz korištenje punog iznosa gradijenta kao veličine pomaka

tome što se složenost postupka ne povećava, no iznimno je teško unaprijed procijeniti dobru vrijednost parametra λ. Naime, uz preveliki iznos, algoritam neće moći odrediti položaj minimuma za dovoljno veliku preciznost ili će čak divergirati. S druge strane, ako je iznos parametra premali, algoritam može jako sporo konvergirati i mora napraviti veliki broj koraka prije nego se dovoljno približi minimumu. U stvarnim uvjetima, konstantni iznos koeficijenta najčešće nije prikladan tijekom čitavog gradijentnog spusta, budući da je na početku možda pogodnije raditi veće korake kako bi se što prije približili području gdje se nalazi minimum, nakon čega je potrebno smanjiti veličinu koraka kako bi se minimum odredio s dovoljnom preciznošću.

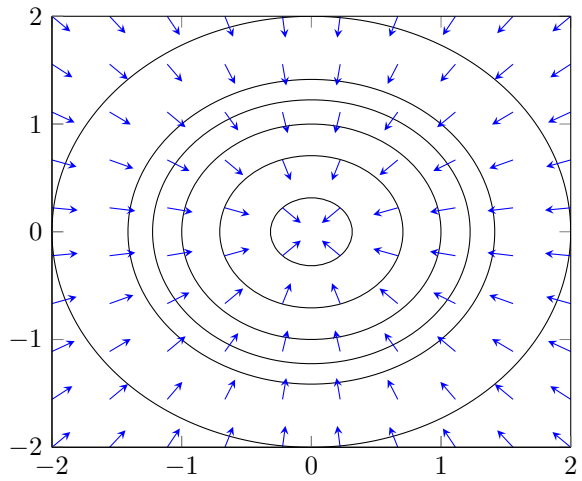
Umjesto konstantnog iznosa, faktor λ se obično određuje uporabom neke od metoda linijskog pretraživanja (npr. metodom zlatnog reza ili kvadratne interpolacije) čime se pronalazi optimum funkcije na pravcu određenom smjerom gradijenta. Pretpostavimo li ovaj pristup, gdje određujemo optimalan iznos pomaka λ, zanimljivo je promotriti kakav je odnos između smjerova gradijenata u dva uzastopna koraka postupka. Ako je promatrana funkcija cilja kvadratna (ili se ponaša približno kvadratnoj funkciji), moguće je pokazati sljedeće svojstvo: gradijent u točki u koju smo došli optimizacijom funkcije na pravcu bit će okomit na smjer toga pravca, odnosno gradijenta u prethodnoj točki. Ovo ponašanje pokazano je na analizi ponašanja Powellovog postupka i vidljivo u izrazu 3.30. Naravno, ova činjenica vrijedi samo ako koristimo pomak optimalnog iznosa u smjeru gradijenta, no može imati dalekosežne posljedice.

Iako je lokalno smanjenje funkcije cilja optimalno, budući da se pomičemo u smjeru u kojem se vrijednost funkcije najbrže smanjuje, postupak gradijentnog spusta globalno ima loša svojstva jer se ponaša kao pohlepna heuristika. Smjer najbržeg pada je uvijek okomit na konturu funkcije u trenutnoj točki, što se može vidjeti na slici 4.2a na kojoj su prikazane konture funkcije $f(x_0, x_1) = x_0^2 + x_1^2$. Kod ove funkcije također vrijedi da negativni gradijent u bilo kojoj točki funkcije uvijek pokazuje prema minimumu funkcije, što se može vidjeti iz slike 4.2b koja pokazuje vektorsko polje negativnih gradijenata te funkcije. No ova poželjna pojava očituje se samo kada je u pitanju sferična kvadratna funkcija.

Već i za jednostavne kvadratne funkcije kod kojih su konture jednakih vrijednosti izduženije po jednoj osi, smjer negativnog gradijenta neće pokazivati prema minimumu funkcije. Naime, ukoliko se konture sferne funkcije "rastegnu" po jednoj osi tako da čine elipse, vektori okomiti na konture više ne pokazuju prema minimumu. Osim toga, kako sada osi imaju različite omjere, to također znači da će i u vektoru gradijenta pojedini elementi biti različitih veličina, što će uzrokovati sporije pomake. Slika 4.3 pokazuje

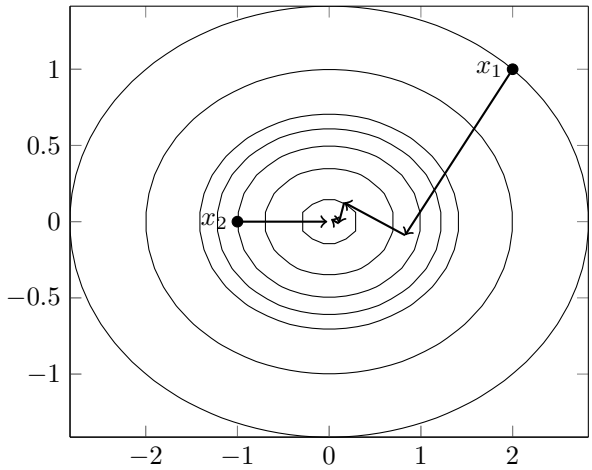


(a) Smjerovi gradijenta sferne kvadratne funkcije

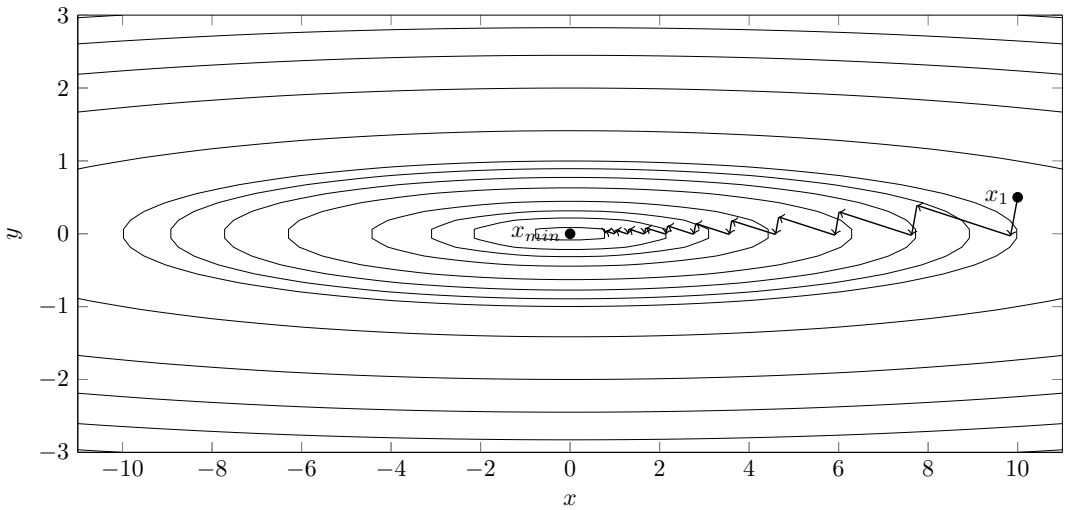


(b) Smjerovi gradijenta sferne kvadratne funkcije

Slika 4.2: Konture i smjerovi gradijenta sferne kvadratne funkcije $f(x_0, x_1) = x_0^2 + x_1^2)$



Slika 4.3: Konture i smjerovi gradijenta izdužene kvadratne funkcije ($f(x_0, x_1) = x_0^2 + 4x_1^2$)

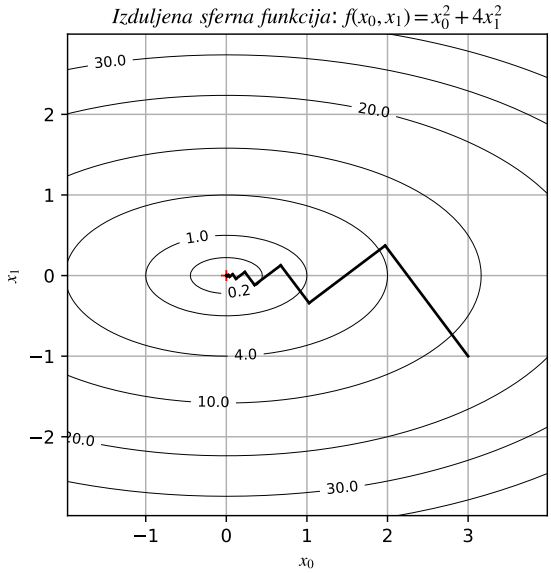


Slika 4.4: Kretanje gradijentnog spusta na funkciji cilja $f(x_0, x_1) = x_0^2 + 100x_1^2$

primjer kretanja postupka gradijentnog spusta za funkciju $f(x_0, x_1) = x_0^2 + 4x_1^2$ iz dviju početnih točaka. Ako postupak kreće iz točke x_1, početni smjer ne pokazuje prema minimumu. U toj situaciji, postupak se pomiče do točke s najmanjom vrijednosti funkcije cilja na pravcu te u njoj ponovo računa gradijent, koji je *okomit* na prethodni smjer. Postupak nastavlja pretraživanje u novom smjeru koji također ne pokazuje točno prema minimumu.

Ako su omjeri među osima još veći, ovakvo iterativno ponašanje sve više dolazi do izražaja, što je vidljivo na slici 4.4 gdje se optimira funkcija $f(x_0, x_1) = x_0^2 + 100x_1^2$. Čak i iz točke $x_1 = (10, 0.5)$ koja je relativno blizu minimuma, postupak jako sporo konvergira. Također, smjerovi u kojima se odvija pretraživanje se ponavljaju i međusobno su okomiti: cijelo vrijeme se izmjenjuju dva smjera, što je problematično jer niti jedan ne pokazuje prema minimumu, te se postupak u "cik-cak" kretanju približava minimumu.

Zbog opisanog ponašanja i problematike određivanja veličine pomaka, metoda gradijentnog spusta se u pravilu nikada ne koristi samostalno. Umjesto korištenja optimalnog pomaka na pravcu, za čije je određivanje također potrebno dodatno evaluirati funkciju cilja u svim iteracijama postupka pretrage na



Slika 4.5: Primjena postupka najbržeg spusta na kvadratnu funkciju cilja čije konture nisu kružnice

pravcu, u praksi se koristi ili konstantni relativno maleni iznos pomaka, ili se iznos prilagođava trenutnim uvjetima po nekoj predefiniranoj strategiji (engl. *step size strategy*). Osim variranja iznosa koraka, postoje mnoge inačice ovoga postupka koje modificiraju smjer pomaka, koristeći više derivacije funkcije ili informacije o prethodnim iteracijama. Usprkos nedostacima, gradijentni spust je osnova mnogih unaprijeđenih optimizacijskih postupaka u strojnom učenju u scenarijima optimizacije vrlo velikog broja varijabli. Također, iako konvergencija može biti izuzetno spora, ovaj postupak sigurno napreduje prema minimumu; to je svojstvo koje se koristi u nekim naprednijim postupcima u slučaju kada brža heuristika divergira, o čemu će biti riječi u sljedećim poglavljima.

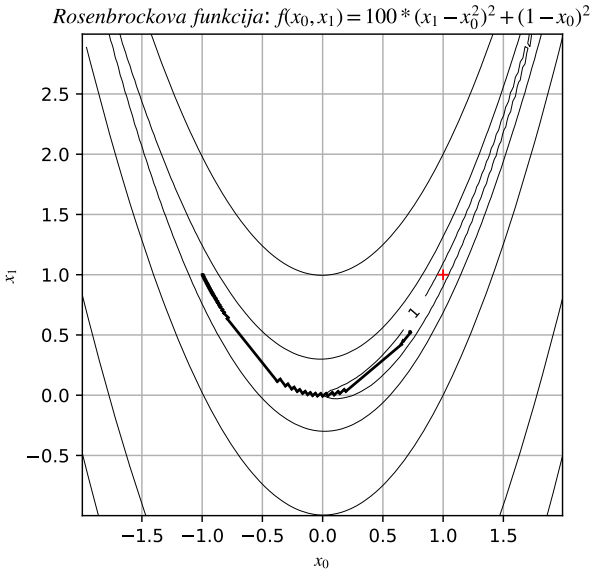
Algoritam 4.1 prikazuje korake metode gradijentog spusta.

Algoritam 4.1: Metoda gradijentnog spusta

Function `GradijetniSpust(`f, x_0, ε`):`
  Podatci: f: funkcija koja se minimizira
    x_0: početna točka
    ε: željena preciznost unimodalnog intervala
  $x = x_0$;
  ponavljaj
    $x_s = x$;
    za $i = 1, 2, \ldots, n$ **čini**
      $\lambda = argmin_\lambda(F(x + \lambda \nabla f(x)))$;
      $x = x + \lambda * \nabla f(x)$;
  dok ne bude $|\nabla f(x)| > \varepsilon$;
  vrati x;

4.2 Konjugirani gradijenti

Metoda najbržeg stupsta, koja je temeljena na kretanju u smjeru gradijenta, pokazala se je iznimno lošim metodom u općenitom slučaju. Razlog je veoma sličan kao kod pretraživanja po koordinatnim



Slika 4.6: Primjena najbržeg spusta na Rosenbrockovu funkciju

osima, naime ovisno o početnoj točki gradijenti ne moraju pokazivati prema smjeru minimuma i postupak jako sporo napreduje prema minimum. Zbog toga se javlja ideja koja je slična Powellovoj metodi, a to je da umjesto da se koriste smjerovi gradijenta, da se na temelju njih konstruira smjer konjugiranih smjerova kojima će se brzže doći do minimuma.

Metoda konjugiranih gradijenata bazira se na kvadratnim funkcijama, dakle pretpostavlja se da se funkcija f može zapisati kao

$$f(x) = \frac{1}{2}x^T Ax + x^T b + c \tag{4.14}$$

Tada gradijent te funkcije možemo zapisati kao

$$\Delta f(x) = Ax + b. \tag{4.15}$$

Ideja postupka je da prvi smjer pretrage bude jednak smjeru gradijenta u početnoj točki t_0

$$p_0 = \nabla f(x_0). \tag{4.16}$$

U tom smjeru se pomaknemo do nove točke x_1

$$x_1 = x_0 - \alpha p_0, \tag{4.17}$$

gdje α predstavlja iznos za koji se je potrebno pomaknuti da dođemo do minimu na tom pravcu. Nakon toga računamo pomak za idući korak, no za razliku od gradijedtnog spusta gdje bi taj smjer bio jednak gradijentu u novoj točki, ovdje želimo odrediti novi smjet p_{k+1} takav da je on konjugiran s prethodni smjer p_k, odnosno ad vrijedi

$$p_{k+1} A p_k = 0. \tag{4.18}$$

To čemo napraviti na način da pretpostavimo da je naš novi smjer jednak

$$p_k + 1 = \nabla f(x_{k+1}) - \beta p_k \tag{4.19}$$

gdje β predstavlja konstantu s kojom bi trebali pomnožiti prethodni smjer kako bi dobili da novi smjer bude konjugiran s prethodnim. Tu konstantu možemo dobiti dako da uvrstimo ... u ...

$$(\nabla f(x_{k+1}) - \beta p_k) A p_k = 0. \tag{4.20}$$

Nakon toga možemo izlučiti β na lijevo strani te dobijemo izraz

$$\beta = \frac{\nabla f(x_{k+1}) A p_k}{A p_k}. \tag{4.21}$$

Time smo dobili izraz za određivanje parametra β, no problem je taj što imamo na raspolaganju samo gradijente funckije f, stoga nam matrica A nije poznata i moramo ju nekako ukloniti iz prethodnog izraza kako bi dobili izraz koji stvarno možemo koristiti. Krenimo prvo od izraza za trenutni gradijent

$$\nabla f(x_{k+1}) = A x_{k+1} + b. \tag{4.22}$$

Ovaj izraz možemo preoblikovati na način da umejsto x_{k+1} uvrstimo izraz pomoću kojeg smo tu točku izračunali iz prethodne

$$\nabla f(x_{k+1}) = A(x_k - \alpha_k p_k) + b. \tag{4.23}$$

Nakon sređivanja dobijemo

$$\nabla f(x_{k+1}) = A x_k + b - A \alpha_k p_k. \tag{4.24}$$

Izraz $Ax_k + b$ predstavlja gradijent u točki x_k pa onda možemo taj izraz zamijeniti s $\nabla f(x_k)$ te s jedne strane izlučiti Ap_k

$$A p_k = \frac{\nabla f(x_k) - \nabla f(x_{k+1})}{\alpha_k}. \tag{4.25}$$

Osim toga, trebat će nam još i jedan izraz za nazivnik koji ćemo dobiti na način da izraz $o_k^T A p_k$ preoblikujemo tako da p_k^T zamijenimo s izrazom s ...

$$p_k^T A p_k = (\nabla f(x_k) + \beta_{k-1} p_{k-1}) A p_k = \nabla f(x_k) A p_k + \beta_{k-1} p_{k-1} A p_k. \tag{4.26}$$

Prisjetimo se da za smjeove pretrage želimo da vrijedi $p_{k-1} A p_k = 0$, što znaći da drugi član u tom izrazu nestaje. Također, izraz za Ap_k smo izveli u....

$$p_k^T A p_k = \frac{\nabla f(x_k)(\nabla f(x_k) - \nabla f(x_{k+1}))}{\alpha_k}. \tag{4.27}$$

Sada možemo uvrstiti ... i ... i dobiti

$$\beta_k = \frac{\frac{\nabla f(x_k) - \nabla f(x_{k+1})}{\alpha_k}}{\frac{\nabla f(x_k)(\nabla f(x_k) - \nabla f(x_{k+1}))}{\alpha_k}} \tag{4.28}$$

U navedenom izrazu možemo pokratiti faktor α te raspositanjem nazivnika na $\nabla f(x_k)^T \nabla f(x_k) - \nabla f(x_k)^T \nabla f(x_{k+1}))$ tako da se konačno dobije izraz

$$\beta_k = \frac{\nabla f(x_k + 1) * \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)} \tag{4.29}$$

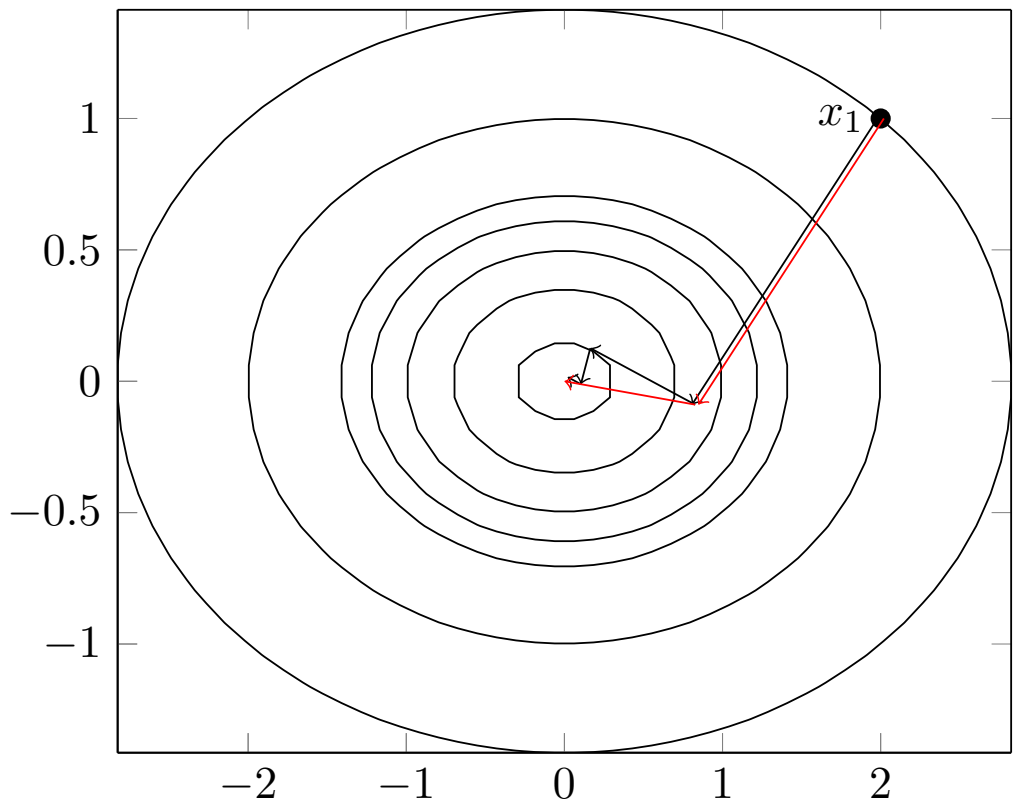
Ako optimiramo kvadratnu funkciju, postupak će nakon n iteracija konstruirati skup od n konjugiranih smjerova. Kako znamo da do minimuma kvadratne funkcije možemo doći pretraživanjem po n konjugiranih smjerova, onda također vrijedi da će postupak također nakon n iteracija pronači i minimum razmatrane funkcije.

Iako je opisana metoda konjugiranih gradijenata razvijena za kvadratne probleme, ona je primjenjiva i za proizvoljnoo nelinearne funkcije. Kada se postupak primjeni za optimizaciju funkcije koja nije kvadratna, onda postupak ne dolazi nužno do minimuma funkcije nakon n koraka. Osim toga, tijekom iteracija

Iako se izraz ... može koristiti i za nelinearne metode, predložena je modifikacija originalnog izraza na način da se brojnik ne skrati kao u slučaju kvadratnih funkcija pa se dobiva izraz

$$\beta_k^{PR} = \frac{\nabla f(x_{k+1})^T (\nabla f(x_k + 1) - \nabla f(x_{k+1}))}{\nabla f(x_k)^T \nabla f(x_k)} \tag{4.30}$$

Razlog zašto se preporučuje korištenje ove varijante jest iz razloga što se ponaša puno robusnije. Razlog



Slika 4.7: Usporedba metode najbržeg spusta (crno) i metode konjugiranih gradijenata (crveno)

tome je taj što ako su smjerovi $\nabla f(x_{k+1})$ i $\nabla f(x_k)$ približno jednaki, tada će $\beta \approx 0$, što znači da će postupak krenuti od početka graditi konjugirane smjerove iz trenutne točke. Na taj način sam postupak će, kada smjerovi postanu neefikasni, ponovo krenuti graditi konjugirane smjerove od početka.

Slika 4.7 prikazuje razliku između ponašanja gradijentnog spusta i metode konjugiranih gradijenata primijenjene nad jednom kvadratnom funkcijom. Linije crne boje prikazuju smjerove pretrage postupka gradijentnog spusta, dok crvene linije prikazuju smjerove metode konjugiranih gradijenata. Kao što se može vidjeti gradijentni postupak radi pretragu po okomitim smjerovima, te se polako približava prema minimumu. Kod postupka konjugiranih gradijenata možemo vidjeti kako je prvi smjer postupka jednak smjeru gradijenta. No nakon toga, idući smjer pretrage pokazuje točno prema minimumu. Tako ovaj postupak za danu funkciju pronalazi minimum u samo dva koraka.

4.3 Newtonova metoda

Za razliku od gradijentnog spusta, u kojem se koriste samo prve derivacije, Newtonova metoda (koja se još naziva i Newton-Raphsonov postupak) pretpostavlja postojanje i drugih derivacija funkcije. Korištenjem drugih derivacija moguće je izgraditi kvadratnu aproksimaciju funkcije cilja. Naime, ako se neka funkcija $f(\boldsymbol{x})$ aproksimira korištenjem Taylorovog reda do drugog stupnja oko početke točke $\boldsymbol{x}_k$, dobiva se

$$f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x}_k)\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T \boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}, \tag{4.31}$$

gdje $\Delta\boldsymbol{x}$ predstavljda odmak od trenutne točke $\boldsymbol{x}_k$ do nove točke $\boldsymbol{x}$ za koju želimo izračunati vrijednost funkcije. Prisjetimo se da za kvadratnu funkciju možemo iznimno lako odrediti minimum na način da ju deriviramo i izjednačimo s nulom (vidi izraz 1.25). Posljedično, prethodnu aproksimaciju možemo derivirati i izjednačiti s nulom te dobiti

$$\nabla f(\boldsymbol{x}_k) + \boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x} = 0.$$

Iz ovog izraza možemo izlučiti $\Delta\boldsymbol{x}$ čime dobijamo

$$\Delta\boldsymbol{x} = -\boldsymbol{H}(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k).$$

Prethodni izraz nam govori koji je potreban pomak $\Delta\boldsymbol{x}$ za koji se potrebno pomaknuti od točke $\boldsymbol{x}_k$ kako bi došli do minimuma kvadratne aproksimacije funkcije f. U pratičnom ostvarenju, do pomaka $\Delta\boldsymbol{x}$ dolazi se rješavanjem sustava linearnih jednadžbi predstavljenog kao

$$\boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x} = -\nabla f(\boldsymbol{x}_k). \tag{4.32}$$

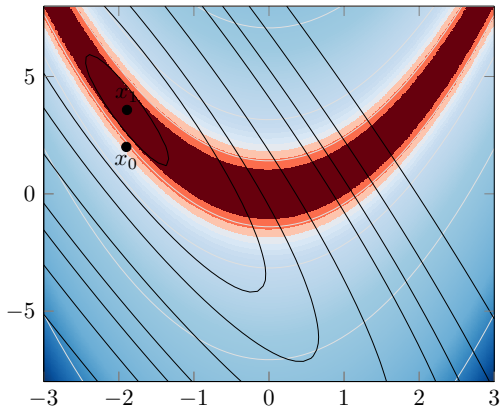
Newtonov postupak temelji se na izgradnji kvadratne aproksimacije u trenutnoj točki te određivanju smjera prema minimumu te kvadratne aproksimacije. Iz ovoga možemo vidjeti kako će postupak, ako se primjeni za optimizaciju kvadratnih funkcija, odrediti minimum izravno, u samo jednoj iteraciji. U tom je slučaju Taylorova aproksimacija egzaktna te dolazimo do minimuma kvadratne funkcije analitički. U slučaju da je funkcija približno kvadratna, tada će aproksimacija dovoljno dobro opisivati originalnu funkciju i bit će moguće izuzetno brzo doći do minimuma. Za razliku od gradijentnog spusta, gdje je potrebno odrediti iznos pomaka u zadanom smjeru, kod Newtonovog postupka to nije nužno. Naime, vektor $\Delta\boldsymbol{x}$ određen izrazom 4.32 definira i smjer i veličinu pomaka potrebnog za dostizanje minimuma kvadratne aproksimacije. Stoga se u najjednostavnijoj inačici može koristiti nepromijenjeni iznos pomaka, koji u općenitom slučaju daje iterativni postupak:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta\boldsymbol{x}_i. \tag{4.33}$$

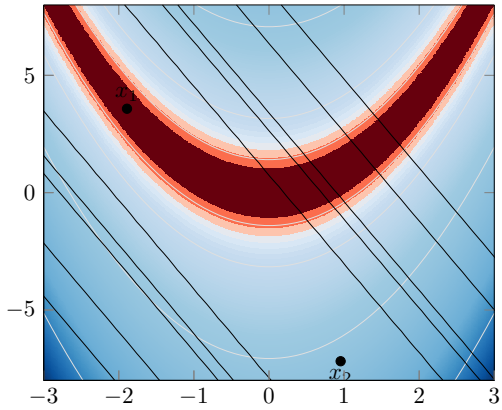
Slika 4.8 prikazuje tri koraka Newtonovog postupka na Rosenbrockovoj funkciji u dvije dimenzije. Osim kontura funkcije koje su naznačene na slici, označene su i konture kvadratnih funkcija kojima Newtonov postupak u svakoj iteraciji aproksimira originalnu funkciju. Možemo vidjeti kako je već nakon tri iteracije postupak došao iznimno blizu točke minimuma. U prvoj iteraciji postupak pravi mali pomak prema minimumu kvadratne aproksimacije te se smješta u dolinu u kojoj se nalazi i minimum. U drugoj iteraciji nova kvadratna aproksimacija posjeduje minimum daleko izvan doline. Razlog tome je što je dolina u kojoj se nalazi točka $\boldsymbol{x}_1$ uska i zavijena. Nažalost, ne postoji kvadratna funkcija koja bi se mogla dobro prilagoditi takvom obliku funkcije cilja, tako da je aproksimacija vrlo izdužena te se minimum nalazi relativno daleko. No u točki $\boldsymbol{x}_2$ postupak je mogao napraviti novu kvadratnu aproksimaciju koja dobro opisuje zakrivljenost funkcije cilja, pa minimum aproksimirane funkcije leži veoma blizu stvarnog minimuma.

Nažalost, Newtonov postupak u ovom najjednostavnijem obliku (4.32) ne jamči konvergenciju za svaku funkciju cilja i svaku početnu točku. Naime, ako je okolina početne točke takva da kvadratna aproksimacija loše opisuje funkciju cilja, postupak može i divergirati, što se očituje povećanjem iznosa funkcije cilja u sljedećim točkama te sve većim pomacima. Može se pokazati da će postupak konvergirati ukoliko je u svakom koraku matrica $\boldsymbol{H}$ kvadratne aproksimacije pozitivno definitna. U slučaju da taj uvjet nije ispunjen, ne postoji jamstvo pronalaska minimuma zadane funkcije.

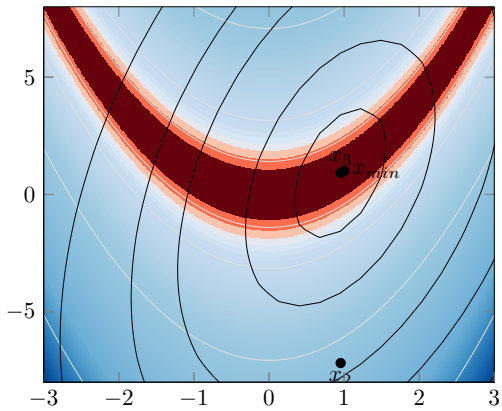
Srećom, ovaj problem moguće je relativno jednostavno riješiti: umjesto translacije za puni iznos pomaka $\Delta\boldsymbol{x}$, moguće je umjesto toga tražiti minimum na pravcu određenim trenutnom točkom i *smjerom*



(a) Prvi korak

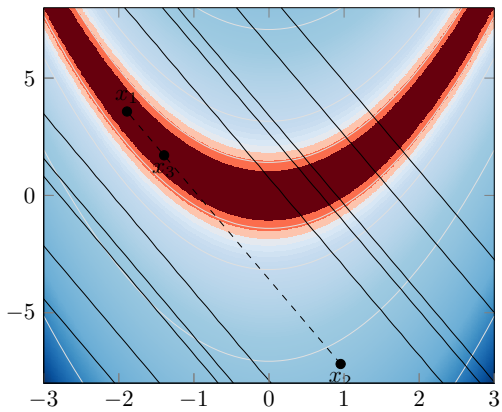


(b) Drugi korak

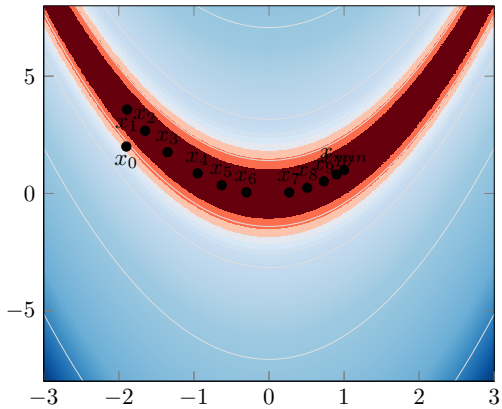


(c) Treći korak

Slika 4.8: Ilustracija prva tri koraka Newtonovog postupka



Slika 4.9: Usporedba pomaka za puni iznos (točka $\boldsymbol{x}_2$) te uz pomoć optimalnog pomaka (točka $\boldsymbol{x}_3$)



Slika 4.10: Tijek Newtonovog postupka uz optimalni pomak na pravcu

$\Delta\boldsymbol{x}$, pa postupak prelazi u sljedeći iterativni oblik:

$$\lambda = argmin_{\lambda}(f(\boldsymbol{x}_i + \lambda\Delta\boldsymbol{x}_i))$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \lambda\Delta\boldsymbol{x}_i$$

Osim traženja optimalnog pomaka na pravcu, postoje i drugi načini prilagodbe; jednostavan pristup je korištenje fiksnog (smanjenog) iznosa pomaka u zadanom smjeru, a moguća je i privremena primjena smjera gradijentnog spusta (odnosno $-\nabla f(\boldsymbol{x}_k)$). U svakom slučaju, uvjet osiguravanja stabilnosti postupka je smanjenje funkcije cilja u sljedećem koraku, koje se može postići opisanim prilagodbama u slučaju da osnovni oblik rezultira pogoršanjem.

Na prethodnom primjeru može se pokazati utjecaj linijskog pretraživanja na pravcu. Slika 4.9 prikazuje drugu iteraciju postupka s označenom točkom $\boldsymbol{x}_2$ koja predstavlja novu točku uz čitav iznos pomaka $\Delta\boldsymbol{x}$, dok točka $\boldsymbol{x}_3$ prikazuje točku koja bi se odabrala kada bi se računao optimalan iznos pomaka. Može se vidjeti kako se korištenjem linijskog pretraživanja osigurava da nova točka nema lošiju funkcijsku vrijednost od trenutne točke. S druge strane, to može imati loš utjecaj na brzinu konvergencije. Konkretno, na primjeru ove funkcije postupak bi sporije konvergirao prema minimumu krećući se zakrivljenom dolinom, što se može vidjeti na slici 4.10. Korištenje pretraživanja na pravcu u Newtonovom postupku ne mora nužno voditi k bržoj konvergenciji, ali zato osigurava da svaka novopronađena točka ima manju funkcijsku vrijednost od prethodne.

Algoritam 4.2 prikazuje korake Newtonove metode. U svakoj se iteraciji smjer pomaka računa

na temelju gradijenta i Hesseove matrice kako je prikazano u retku 6. Nakon toga moguće je odrediti optimalni pomak (linija 7) u novu točku. Postupak se ponavlja dok iznos pomaka ne postane dovoljno malen te se tada kao rezultat uzima posljednja točka.

Algoritam 4.2: Algoritam Newtonove metode

```
1  Function Newton(f, x_0, ε):
       Podatci: f: funkcija koja se minimizira
                x_0: početna točka
                ε: željena preciznost unimodalnog intervala
2      x = x_0;
3      ponavljaj
4          x_s = x ;
5          za i = 1, 2, ..., n čini
6              Δx = −H^{−1} ∇f(x) ;
7              λ = argmin_λ(F(x + λΔx)) ;
8              x = x + λ ∗ Δx;
9      dok ne bude |∇f(x)| > ε;
10     vrati x;
```

Primijenimo li ovaj postupak na Rosenbrockovu funkciju uz puni iznos pomaka, do rješenja se dolazi u samo četiri koraka, kao što je vidljivo na slici 4.11a. U praksi se ipak, zbog moguće nestabilnosti, koristi ograničavanje pomaka na neki faktor ili traženje minimuma na pravcu; takav način pronalaženja optimuma prikazan je na slici 4.11b. Primjena ovoga postupka na kvadratne funkcije, kakva je sferna ili rotirana funkcija, rezultirala bi pronalaženjem minimuma u jednoj iteraciji, budući da u tom slučaju Newtonova aproksimacija odgovara zadanoj funkciji cilja.

4.4 Kvazi-Newtonove metode

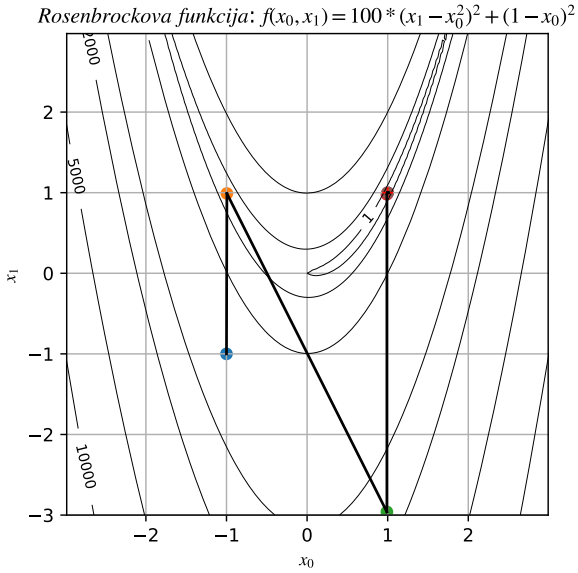
Vidjeli smo kako se pomak (ili smjer pomaka) u Newtonovoj metodi definira izrazom

$$\boldsymbol{H}\Delta\boldsymbol{x} = -\nabla f. \tag{4.34}$$

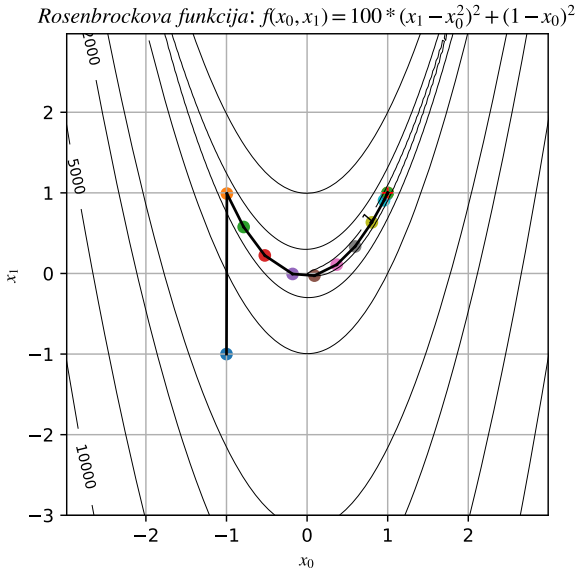
Prema tome, u svakom koraku postupka potrebno je izračunati ne samo Hesseovu matricu funkcije f u trenutnoj točki, nego i invertirati tu matricu ili riješiti linearni sustav kako bi odredili iznos pomaka $\Delta\boldsymbol{x}$. Kako bismo smanjili količinu potrebnog računanja, osnovna ideja skupine optimizacijskih postupaka koji se nazivaju kvazi-Newtonove metode je da se matrica $\boldsymbol{H}$ ne računa eksplicitno, već aproksimira na temelju iznosa gradijenata u točkama pretrage. Kvazi-Newtonove metode pokazale su se najučinkovitijim postupcima optimiranja uz uporabu gradijenta, te se često koriste kao pretpostavljeni postupak optimiranja u mnogim numeričkim programskim paketima.

Aproksimacija Hesseove matrice, koju možemo označiti s $\boldsymbol{B} \approx \boldsymbol{H}$, trebala bi imati neka poželjna svojstva: primjerice, budući da je Hesseova matrica uvijek simetrična, korisno je da i njena aproksimacija bude simetrična. Također, aproksimacija bi trebala voditi k minimumu, pa bi aproksimirana matrica trebala biti pozitivno definitna. Zajednička ideja svih kvazi-Newtonovih postupaka je polazak od inicijalne aproksimacije Hesseove matrice, $\boldsymbol{B}_0$, koja se potom iz koraka u korak ažurira informacijama koje smo dobili tijekom provedbe postupka. U svakom postupku polazimo s kvadratnom aproksimacijom zadane funkcije f:

$$f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x}_k)\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}, \tag{4.35}$$



(a) Newtonova metoda uz puni iznos pomaka



(b) Newtonova metoda uz traženje minimuma na pravcu

Slika 4.11: Primjena Newtonovog postupka na Rosenbrockovu funkciju

gdje nas pomak $\Delta \boldsymbol{x}$ vodi nas u sljedeću točku $\boldsymbol{x}_{k+1}$, odnosno $\boldsymbol{x}_k + \Delta \boldsymbol{x} = \boldsymbol{x}_{k+1}$. Gradijent ove Taylorove aproksimacije po pomaku $\Delta \boldsymbol{x}$ je

$$\nabla f(\boldsymbol{x}_{k+1}) = \nabla f(\boldsymbol{x}_k) + \boldsymbol{H}\Delta \boldsymbol{x}. \tag{4.36}$$

U slučaju kvadratne funkcije, Hesseova matrica bila bi konstantna; za funkcije koje se ponađaju približno kvadratno možemo pretpostaviti da se ta matrica mijenja sporo iz koraka u korak. Uz ovu pretpostavku, aproksimacija Hesseove matrice u sljedećem koraku može se definirati kao $\boldsymbol{B}_{k+1} \approx \boldsymbol{H}_k$, odnosno

$$\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) = \boldsymbol{B}_{k+1}\Delta \boldsymbol{x}. \tag{4.37}$$

Prikažemo li aproksimaciju kao zbroj njene trenutne vrijednosti i razlike prema sljedećem koraku, $\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \Delta \boldsymbol{B}_k$, iz prethodnog izraza dobivamo

$$\Delta \boldsymbol{B}_k \Delta \boldsymbol{x} = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) - \boldsymbol{B}_k \Delta \boldsymbol{x} \tag{4.38}$$

U ovoj jednadžbi nepoznanica je jedino matrica $\Delta \boldsymbol{B}_k$, potrebna za računanje nove vrijednosti aproksimacije. Rješenje ove matrične jednadžbe, koje predstavlja način ažuriranja aproksimirane Hesseove matrice, može se dobiti na više načina, o čemu će ovisiti i svojstva konkretno odabranog postupka i dobivene aproksimacije. Metoda SR1 (engl. *Symmetric rank 1*) je najjednostavnija kvazi-Newtonova metoda, ali i dalje uporabljiva te je možemo iskoristiti za ilustraciju osnovne ideje.

Metoda SR1 definira ažuriranje aproksimacije Hesseove matrice uz pomoć izraza

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \boldsymbol{u}\boldsymbol{u}^T, \tag{4.39}$$

pri čemu je $\boldsymbol{u}$ vektor dimenzije $n \times 1$. Motivacija za korištenje ovog izraza za ažuriranje leži u činjenici da, ako je početna aproksimacija $\boldsymbol{B}_0$ simetrična, ona će ostati takva nakon svakog ažuriranja. Jedino što preostaje jest odrediti na koji način je potrebno izračunati $\boldsymbol{u}$. Cijeli izraz možemo pomnožiti s $\Delta \boldsymbol{x}$

$$\boldsymbol{B}_{k+1}\Delta \boldsymbol{x} = \boldsymbol{B}_k \Delta \boldsymbol{x} + \boldsymbol{u}\boldsymbol{u}^T \Delta \boldsymbol{x}, \tag{4.40}$$

nakon čega čitav izraz možemo uvrstiti u 4.37 i dobiti

$$\nabla f(\boldsymbol{x} + \Delta \boldsymbol{x}) = \nabla f(\boldsymbol{x}) + \boldsymbol{B}_k \Delta \boldsymbol{x} + \boldsymbol{u}\boldsymbol{u}^T \Delta \boldsymbol{x}. \tag{4.41}$$

Rješenje gornje jednadžbe za umnožak $\boldsymbol{u}\boldsymbol{u}^T$ može se dobiti kao

$$\boldsymbol{u}\boldsymbol{u}^T = \frac{(\nabla f(\boldsymbol{x} + \Delta \boldsymbol{x}) - \nabla f(\boldsymbol{x}) - \boldsymbol{B}_k \Delta \boldsymbol{x})(\nabla f(\boldsymbol{x} + \Delta \boldsymbol{x}) - \nabla f(\boldsymbol{x}) - \boldsymbol{B}_k \Delta \boldsymbol{x})^T}{(\nabla f(\boldsymbol{x} + \Delta \boldsymbol{x}) - \nabla f(\boldsymbol{x}) - \boldsymbol{B}_k \Delta \boldsymbol{x})^T \Delta \boldsymbol{x}} \tag{4.42}$$

Zbog jednostavnosti zapisa uvodimo pokratu

$$\boldsymbol{s} = \nabla f(\boldsymbol{x} + \Delta \boldsymbol{x}) - \nabla f(\boldsymbol{x}) \tag{4.43}$$

radi čitljivosti i time dobivamo izraz

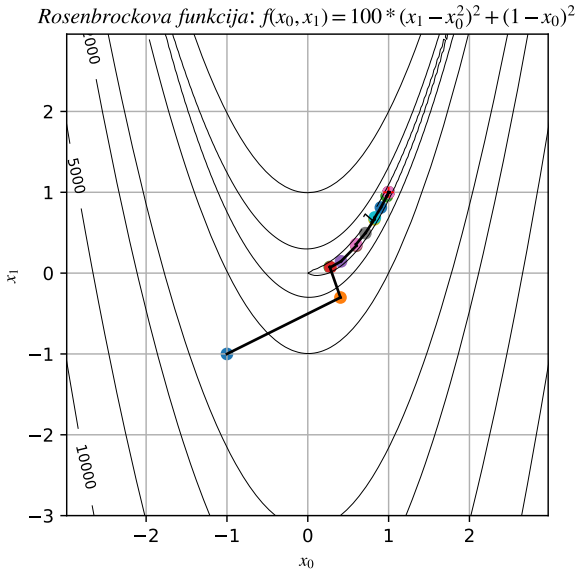
$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{(\boldsymbol{s} - \boldsymbol{B}_k \Delta \boldsymbol{x})(\boldsymbol{s} - \boldsymbol{B}_k \Delta \boldsymbol{x})^T}{(\boldsymbol{s} - \boldsymbol{B}_k \Delta \boldsymbol{x})^T \Delta \boldsymbol{x}} \tag{4.44}$$

Prethodni izraz definira aproksimaciju Hesseove matrice u svakom koraku postupka. No za primjenu Newtonovog postupka i dalje je potrebno izračunati inverz aproksimacije kako bi odredili smjer pomaka. Postavlja se pitanje, kada već aproksimiramo Hesseovu matricu, je li moguće izravno aproksimirati njen inverz? Ako se izraz 4.37 napiše kao $\Delta \boldsymbol{x} = \boldsymbol{B}_{k+1}^{-1}(\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k))$, moguće je doći do izraza za aproksimaciju inverzne Hesseove matrice u obliku

$$\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{B}_k^{-1} + \frac{(\Delta \boldsymbol{x} - \boldsymbol{B}_k^{-1} s)(\Delta \boldsymbol{x} - \boldsymbol{B}_k^{-1} \boldsymbol{s})^T}{(\Delta \boldsymbol{x} - \boldsymbol{B}_k^{-1}(\boldsymbol{s}))^T \boldsymbol{s}} \tag{4.45}$$

Na taj način postupak je moguće ubrzati jer neće biti potrebno u svakoj iteraciji računati matrični inverz.

Prednosti metode SR1 su njezina jednostavnost i svojstvo da je aproksimirana matrica uvijek sime-



Slika 4.12: Primjena kvazi-Newtonovog postupka Davidon-Fletcher-Powell na Rosenbrockovu funkciju

trična, no metoda ima određene nedostatke. Prvi nedostatak je taj da nazivnik može postati približno jednak 0, što onemogućuje izračun nove aproksimacije. U tom slučaju moguće je jednostavno preskočiti računanje aproksimacije u tom koraku, odnosno zadržati trenutnu vrijednost matrice. Drugi nedostatak je što ova metoda ne jamči da će aproksimirana Hesseova matrica biti pozitivno definitna, što može uzrokovati probleme slične kao i kod primjene Newtonove metode. Ipak, praksa je pokazala kako metoda ažuriranja definirana ovom metodom može vrlo dobro aproksimirati Hesseovu matricu.

Osim prethodno opisanog načina, postoje i druge metode kojima je moguće aproksimirati Hesseovu matricu, odnosno njen inverz. Dvije poznatije metode su Davidon-Fletcher-Powel (DFP) i Broyden-Fletcher-Goldfarb-Shanno (BFGS) metode. Kod DFP metode aproksimacija Hesseove matrice računa se kao

$$\boldsymbol{B}_{k+1} = \left(I - \frac{\boldsymbol{s}\Delta\boldsymbol{x}^T}{\boldsymbol{s}\Delta\boldsymbol{x}}\right)\boldsymbol{B}_k\left(I - \frac{\boldsymbol{s}\Delta\boldsymbol{x}^T}{\boldsymbol{s}\Delta\boldsymbol{x}}\right) + \frac{\boldsymbol{s}\boldsymbol{s}^T}{\boldsymbol{s}^T\Delta\boldsymbol{x}} \tag{4.46}$$

odnosno njen inverz kao

$$\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{B}_k^{-1} + \frac{\Delta\boldsymbol{x}\Delta\boldsymbol{x}^T}{\Delta\boldsymbol{x}^T\boldsymbol{s}} - \frac{\boldsymbol{B}_k^{-1}\boldsymbol{s}\boldsymbol{s}^T\boldsymbol{B}_k^{-1}}{\boldsymbol{s}^T\boldsymbol{B}_k^{-1}\boldsymbol{s}}. \tag{4.47}$$

S druge strane, kod BFGS metode aproksimacija Hesseove matrice računa se kao

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{s}\boldsymbol{s}^t}{\boldsymbol{x}\Delta\boldsymbol{x}} - \frac{\boldsymbol{B}_k\Delta\boldsymbol{x}\left(\boldsymbol{B}_k\Delta\boldsymbol{x}\right)^T}{\Delta\boldsymbol{x}^T\boldsymbol{B}_k\Delta\boldsymbol{x}} \tag{4.48}$$

odnosno aproksimacija njenog inverza kao

$$\boldsymbol{B}_{k+1}^{-1} = \left(\boldsymbol{I} - \frac{\Delta\boldsymbol{x}\boldsymbol{s}^T}{\boldsymbol{s}^T\Delta\boldsymbol{x}}\right)\boldsymbol{B}_k^{-1}\left(\boldsymbol{I} - \frac{\boldsymbol{s}\Delta\boldsymbol{x}^T}{\boldsymbol{s}^T\Delta\boldsymbol{x}}\right) + \frac{\Delta\boldsymbol{x}\Delta\boldsymbol{x}^T}{\boldsymbol{s}^T\Delta\boldsymbol{x}}. \tag{4.49}$$

Prednost DFP i BFGS metoda nad SR1 metodom je što su njihove aproksimacije uvijek pozitivno definitne. Iskustvo pokazuje kako BFGS metoda općenito ostvaruje bolje rezultate te je iz tog razloga i najčešće korištena kvazi-Newtonova metoda.

Jedno od mnogih svojstava kvazi-Newtonovih metoda jamči da, ako se koriste za optimizaciju kvadratnih funkcija, potrebno im je najviše $n + 1$ iteracija kako bi odredili optimum. U najgorem slučaju, potrebno je najviše n iteracija kako bi se točno aproksimirala Hesseova matrica (odnosno njen inverz) te zatim još jedan korak kako bi se pomoću te matrice odredio pomak prema minimumu.

Kvazi-Newtonovi postupci u pretpostavljeni odabir optimiranja u mnogim programskim paketima, uključujući i funkciju `minimize` iz programskog paketa `scipy.optimize`. Ukoliko se funkcija `minimize` pozove bez navođenja konkretnog postupka optimiranja, koristit će se prethodno opisana metoda BFGS. Sljedeći primjer pokazuje korištenje ovog postupka u paketu `scipy.optimize` za minimizaciju Rosenbrockove funkcije, koja je također dostupna u istom paketu:

```
# ukljucivanje objekta minimize te Rosenbrockove funkcije cilja
from scipy.optimize import minimize, rosen, rosen_der
result = minimize(rosen, [-1.9,2.1], method='BFGS')
print(result)
```

Iz ispisa rezultata može se vidjeti ukupan broja evaluacija funkcije cilja `nfev`, ukupan broj evaluacija gradijenta `njev` te konačne iznose vektora gradijenta i aproksimacije inverzne Hesseove matrice, `hess_inv`:

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 2.055843111863571e-11
        x: [ 1.000e+00  1.000e+00]
      nit: 31
      jac: [ 3.480e-07 -2.295e-07]
 hess_inv: [[ 4.993e-01  9.982e-01]
            [ 9.982e-01  2.000e+00]]
     nfev: 111
     njev: 37
```

Budući da nismo definirali gradijent funkcije cilja, funkcija `minimize` će aproksimirati gradijent uz trošak dodatnih evaluacija funkcije cilja. U ovom slučaju, gradijent Rosenbrockove funkcije je već implementiran u ugrađenoj funkciji `rosen_der` paketa `scipy_optimize`. Ako nam je gradijent na raspolaganju u eksplicitnom obliku, isti se može definirati prilikom poziva `minimize` uz navođenje parametra `jac`:

```
>>> result = minimize(rosen, [-1.9,2.1], method='BFGS', jac=rosen_der)
>>> print(result)
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 3.2143924630823e-15
        x: [ 1.000e+00  1.000e+00]
      nit: 31
      jac: [ 3.406e-07 -2.259e-07]
 hess_inv: [[ 4.993e-01  9.982e-01]
            [ 9.982e-01  2.000e+00]]
     nfev: 37
     njev: 37
```

Primijetimo kako je u ovom slučaju broj pozivanja funkcije cilja znatno manji, budući da nije bilo potrebno aproksimirati gradijent dodatnim evaluacijama. Zaključno, iz opisanih primjera vidljivo je kako su kvazi-Newtonove metode, a posebice metoda BFGS, preporučeni odabir prilikom optimiranja uz uporabu gradijenta.

4.5 Metoda područja povjerenja

Ideja metode područja povjerenja je da se izgradi kvadratni model funkcije f

$$f(x) = \frac{1}{2}x^t Ax + b^t x + c \tag{4.50}$$

No za razliku od Newtonovog i sličnih postupaka nečemo dozvoliti algoritmu da se pomakne za čitav iznos pomaka niti da se radi linijsko pretraživanje, već se definira područje povjerenja za kvadratni model, te se dozvoljava pomak samo unutar tog područja. U svakoj iteraciji se onda rješava dani problem

$$minm(p) = \frac{1}{2}p^t Ap + b^t p + c \tag{4.51}$$

$$||p|| <= \Delta. \tag{4.52}$$

Dakle, minimizamo kvadratni model koji smo konstruirali u točki x_k, ali na način da pomak ne bude veći od neke zadane vriejdnosti.

Ideja čitavog postupka svoid se na to da se područje povjerenja adaptivno mijenja tijekom rada postupka. Naime, ako je postupak uspio dobro minimizirati razmatranu funkciju, onda nakon što se pomaknemo u tom smjeru povečavamo područje povjerenja te time proširujemo prostor pretrage. S druge strane, ako je model loše aproskimirao funkciju, onda se ne pomičemo u tom smjeru , već smanjujemo prostor pretrage i u idućem koraku radimo precizniju pretragu u području povjerenja. Način na koji provjeravamo je li pomak uspješan ili je da izračunamo parametar ρ

$$\rho = \frac{f(x_k) - f(x_k + p)}{m_k(0) - m_k(p_k)}. \tag{4.53}$$

Dakle, ako smo dobili puno veće smanjenje funkcije u kvadratnom modelu nego što je to slučaj kod originalne funkcije, onda smatramo da pomak nije bio dobar.

Jedna stvar koju je još potrebno definirati jest na koji način se obavlja pomak. Obično se koriste dva načina određivanja pomaka. Prvi je takozvani pomak Cauchyjemo točkom gdje se pomičemo u smjeru gradijenta, no za iznos ne veći od područja povjerenja. Dakle smjer pomicanja jednak je smjeru gradijenta $\frac{\nabla f(x_k)}{|\nabla f(x_k)|}$. Iznos za koji se pomičemo odredimo na način da provjerimo ispitamo iznos $\nabla f(x_k)^T H_k \nabla f(x_k)$. Ako je taj izraz manji od 0, onda vrijedi da je model funckije negatifvno definitan, što znaći da kvadratni model neam eksplicitni minimum. Stoga se u danom smjeru pomičemo za maksimalni dozvoljeni iznos za koji ćemo ostati u području povjerenja

$$p_k = -\Delta_k \frac{\nabla f(x_k)}{|\nabla f(x_k)|}. \tag{4.54}$$

U suprotnom, ako je matrica kvadratnog modela pozitivno definitna, onda funkcija ima eksplicitni minimum. Tada možemo eksplicitno odrediti iznos za koji se možemo pomaknuti da dođemo do minimuma na pravcu. Naime, tražimo minimum na pravcu kvadratnog modela

$$m_k(\tau p_k) = f(x_k) + \tau \nabla f(x_k)^T p_k + \frac{\tau}{2} p_k^T H_k p_k. \tag{4.55}$$

Ako prethodni izraz deriviramo po τ i izjednačimo s 0 dobit ćemo

$$\nabla f(x_k) p_k + \tau p_k^T H_k p_k = 0 \tag{4.56}$$

odnosno

$$\tau = \frac{\nabla f(x_k) p_k}{p_k^T H_k p_k}. \tag{4.57}$$

Sada još zamijenimo p_k s izrazom ... i dobivamo

$$\tau = \frac{1||\nabla f(x_k) p_k||^3}{\Delta_k p_k^T H_k p_k}. \tag{4.58}$$

No ovdje se može desiti da dobivena vrijednost za τ ispadne i veća od 1 ako se minimum na traženom pravcu nalazi izvan područja povjerenja. Zbog toga se za τ uzima

$$\tau = \max\left(1, \frac{1||\nabla f(x_k)p_k||^3}{\Delta_k p_k^T H_k p_k}\right). \tag{4.59}$$

No prethodno opisana metoda ne pruža nikakve pogodnosti, jer se zapravo svodi na metodu gradijentnof spusta za koju već znamo da ima relativno loša svojstva kod optimizacije mnogih funkcija. Ubog toga se umjesto prethodnog načina izračuna koristi takozvana dogleg metoda. Ta metoda konstruira pomak na temelju gradijenta i na temelju gradijenta ali i smjeru prema minimumu kvadratnog modela. Na taj način iskorištava se i informacija o drugim derivacijama. Ideja ove metode je sljedeća. Prvo se računa smjer gradijenta te iznos za koji se je moguće pomaknuti, slično kao i kod prethodne metode. Ako τ ispadne veći od 1, to znaći da je minimum na pravcu bio izvan područja povjerenja i pomičemo se u točki koja se nalazi na granici. S druge strane, ako smo za τ dobili iznos koji je veći ili jednak 1, to znači da se minimum na pravcu nalazi unutar područja povjerenja. U tom slučaju čemo se još dodatno pomaknuti u smjeru minimuma našeg kvadratnog modela, i to opet pazeći da ne izađemo iz područja povjerenja.

$$p_k = \begin{cases} \tau p_k^g & 0 \leq \tau \leq 1 \\ p_k^g + (\tau - 1)(p_k^N - p_k^g) & 1 < \tau \leq 2 \end{cases}. \tag{4.60}$$

gdje p_k^U pokazuje smjer gradijenta

$$\nabla f(x_k) \frac{\nabla f(x_k) p_k}{p_k^T H_k p_k} \tag{4.61}$$

dok p_k^N predstavlja smjer prema minimumu kvadratnog modela.

Naravno, jedna bitna stvar koju je tu potrebno napraviti jest odrediti iznos za koji se možemo pomaknuti, odnsno mora vrijediti

$$|p_k^g + (\tau - 1)(p_k^N - p_k^g)| = \Delta^2 \tag{4.62}$$

Iz čega se dobiva kvadratna jednadžba

$$(p_k^U)^2 + 2\tau(p_B - p^U)p^U + \tau^2(p^B - p^U) = \Delta^2 \tag{4.63}$$

iz čega dobivamo kvadratnu jednadžub

$$\tau = \frac{-2(p_B - p^U)p^U + 2(p^B - p^U)\Delta}{2(p^B - p^U)} = \frac{\Delta - p^U}{(p^B - p^U)} \tag{4.64}$$

POGLAVLJE 5

Metode za probleme najmanjih kvadrata

Jedan česti problem koji se javlja u nelinearnoj optimizaciji jest problem najmanjih kvadrata. Ovaj problem javlja se kada imamo skup od m nelinearnih jednadžbi i n varijabli te je potrebno pronaći skup variabli za koje se postiže najmanja greška na damom skupu nelinarnih jednadžbi.

$$f_1(x) = 0 \tag{5.1}$$

$$f_2(x) = 0 \tag{5.2}$$

$$\vdots$$

$$f_m(x) = 0 \tag{5.3}$$

$$\tag{5.4}$$

Ovaj problem možemo zapisati kao optimizacijski problem preko sume kvadrata nelinearnih jednadjžbi

$$f(x) = \sum_{i=1}^{m} f_m(x). \tag{5.5}$$

Ako sustav nelinearnih jednadžbi ima rješenje koje u potpunosti zadovoljava svaku jednadžbu, onda će minimum imati vrijednost 0. No najčešće ne postoji rješenje kod kojeg bi sve jednakosti bile u potpunosti zadovoljene, pa je cilj pronaći ono rješenje koje najbolje zadovoljava zadane nelinearne jednadžbe. Ovako zadani problem možemo riješiti bilo kojim od postupaka nelinearne optimizacije koji su se do sada obradili. Zbog posebnog oblika ovog problema, razvijene su i metode prikladne upravo za rješavanje ovakvih problema. Dvije popularne metode za rješavanje nelinearnih problema najmanjih kvadrata su Gauss-Newtonova i Levenberg-Marquardtova metoda.

5.1 Gauss-Newtonova metoda

Gauss-Newtonova metoda je bazirana na Newton-Raphsonovoj metodi, ali posebno prilagođena za rješavanje nelinearnog problema najmanjih kvadrata. Naime, za razliku od Newtonove metode, Gauss-Newtonova metoda ne koristi druge derivacije, već nastoji aproksimirati Hesseovu matricu koja je potrebna za izračun.

Prisjetimo se da je kod Newtonoe metode pomak računao kao

$$\Delta x = H^{-1} \nabla f. \tag{5.6}$$

Funkcija f je pri tome definirana kao suma od m kvadriranih funkcija g_i

$$f(x) = \sum_{i=1}^{m} (g_m(x))^2. \tag{5.7}$$

U tom slučaju, gradijent funkcije f možemo zapisati kao

$$\nabla f = 2\sum_{i=0}^{m} g_m(x)\frac{\partial g_m(x)}{\partial x_j}. \tag{5.8}$$

Gradijent možemo kraće zapisati i kao

$$\nabla f = 2Jg, \tag{5.9}$$

gdje J predstavlja Jakobijevu matricu

$$J = \begin{bmatrix} \frac{\partial g_0}{\partial x_0} & \cdots & \frac{\partial g_0}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_0} & \cdots & \frac{\partial g_m}{\partial x_n} \end{bmatrix}, \tag{5.10}$$

u kojoj svaki redak predstavlja pojedinu nelinearnu jednadžbu derivirano po svakoj varijabli zasebno. S druge strane, g predstavlja vektor nelinearnih jednadžbi

$$g = \begin{bmatrix} g_1 \\ \vdots \\ g_m \end{bmatrix}. \tag{5.11}$$

S druge strane, Hesseovu matricu možemo zapisati kao

$$H = \begin{bmatrix} 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_1}\frac{\partial g_i}{\partial x_1} + g_i\frac{\partial^2 g_i}{\partial x_1 \partial x_1}\right) & \cdots & 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_1}\frac{\partial g_i}{\partial x_n} + g_i\frac{\partial^2 g_i}{\partial x_1 \partial x_n}\right) \\ \vdots & \ddots & \vdots \\ 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_n}\frac{\partial g_i}{\partial x_1} + g_i\frac{\partial^2 g_i}{\partial x_n \partial x_1}\right) & \cdots & 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_n}\frac{\partial g_i}{\partial x_n} + g_i\frac{\partial^2 g_i}{\partial x_n \partial x_n}\right) \end{bmatrix} \tag{5.12}$$

Kao što možemo vidjeti svaki element Hesseove matrice racuna se kao suma dviju parcijalnih derivacija. Ideja Gauss-Newtonove metode je da se u svakom elementu element drugih derivacija zanemari, odnosno da se Hesseova matrica aproksimira izrazom

$$H \approx \begin{bmatrix} 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_1}\frac{\partial g_i}{\partial x_1}\right) & \cdots & 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_1}\frac{\partial g_i}{\partial x_n}\right) \\ \vdots & \ddots & \vdots \\ 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_n}\frac{\partial g_i}{\partial x_1}\right) & \cdots & 2\sum_{i=1}^{m}\left(\frac{\partial g_i}{\partial x_n}\frac{\partial g_i}{\partial x_n}\right) \end{bmatrix}. \tag{5.13}$$

Ova aproksimacija se još može zapisati kao

$$H \approx 2J^T J. \tag{5.14}$$

Iz toga slijedi da izraz za računanje pomaka koji je kod Newtonovog postupka bio

$$H\Delta x = -\nabla f \tag{5.15}$$

se kod Gauss-Newtona uz aproksimacuju Hesseove matrice može zapisati kao

$$J^T J\Delta x = -J^T g. \tag{5.16}$$

Odnosno, kada pomak izrazimo na lijevoj strani

$$\Delta x = -(J^T J)^{-1} J^T g. \tag{5.17}$$

U slučaju kada je broj varijabli i broj jendadžbi jednak, onda se izraz pojednostavljuje

$$\Delta x = -J^{-1} g, \tag{5.18}$$

jer vrijedi da je $(J^T J)^{-1} = J^{-1}(J^T)^{-1}$.

Važno je spomenuti da ovaj postupak radi samo pod pretpostavkom da je broj jednadžbi veći ili jednak od broja varijabli, odnosno da vrijedi $m \geq n$. U suprotnome, matrica $J^T J$ biti će singularna te neće biti moguće odrediti njezin inverz.

Algoritam 5.1 prikazuje korake, Gauss-Newtonovog postupka. Kao što se može vidjeti postupak je sličan Newtonovom postupku.

Algoritam 5.1: Algoritam Gauss-Newtonove metode

```
1 Function GaussNewton(f, ε):
      Podatci: f: funkcija koja se minimizira
               x_0: početna točka
               ε: preciznost za uvjet zaustavljanja
2     x = x_0 ;
3     ponavljaj
4         Δx = −(J(x)^T J(x))^{−1} J(x)^T G(x) ;
5         x = x + Δx ;
6     dok ne bude J(x)^T g(x) < ε;
7     vrati x ;
```

5.2 Metoda po Levenberg–Marquardtu

Metoda po Levenberg-Marquardtu funkcionira na sličan način kao i Gauss-Newtonov postupak, no polušava riješiti određene probleme kod te metode. Glavna razlika između postupaka je u izrazu koji se koristi za izračun pomaka. Kod Levenberg-Marquardt postupka koristi se izraz

$$(J^T J + \lambda I)\Delta x = J^T g, \tag{5.19}$$

odnosno

$$\Delta x = (J^T J + \lambda I)^{-1} J^T g, \tag{5.20}$$

gdje I predstavlja jediničnu matricu, a λ pozitivni faktor prigušenja. Ideja iza ovog postupka je da se parametar λ adaptivno mijenja tijekom rada postupka kako bi se na taj način povečala njegova efikasnost. Naime, ako je faktor λ jako malen, onda se gubi utjecaj drugog člana u izrazu i postupak se ponaša kao Gauss-Newtonov postupka. S druge strane, ako je λ velik, tada jedinična matrica ima najveći utjecaj te se gubi utjecaj člana $J^T J$. U tom slučaju se zapravo ponaša kao gradijentni spust. Dakle, ideja postupka da, ovisno o situaciji, se ponaša ili kao gradijentni spust ili kao Gauss-Newtonov postupak. Razlog tome je što se može desiti da kvadratna aproksimacija koju radi Gauss-Newtonova metoda loše aproksimira funkciju i zbog toga nije moguće napraviti veliki pomak u tom smjeru. U tom slučaju, sigurnije je napraviti pomak u smjeru gradijenta, koji predstavlja smjer nabrže promjene funkcije u toj točki. Ideja je dakle da se postupak proba pomaknuti za izračunati pomak. No, prije nego se ta točka prihvati ili odbaci, algoritam određuje koliko je ovaj pomak bio uspješan. To se radi preko izraza

$$\rho = \frac{f(x) - f(x + \Delta x)}{M(0) - M(\Delta x)} \tag{5.21}$$

u kojem gledamo omjer pravog smjanjenja funkcije i smanjenja kvadratnog modela funkcije. Ako je $\rho > 0$ to znači da smo se pomaknuli u smjer u kojem se vrijednost originalne funkcije smanjuje. U tom slučaju prihvačamo novu točku te smanjujemo vrijednost parametra μ. U suptrotnome, ako je $\rho < 0$, to znači da je pomak uzrokovao pogoršanje u vrijednosti originalne funkcije, i da pomak nije bio uspješan. U tom slučaju se vrijednost parametra λ povečava. Također, valja spomenuti da je član u nazivnku moguće

raspisati i da se mnogi članovi mogu pokratiti

$$M(0) - M(\Delta x) = f(x) - f(x) - \Delta x^T J(x)^T g(x) - \frac{1}{2}\Delta x^T J^T J \Delta x \tag{5.22}$$

$$= -\frac{1}{2}\Delta x^T (2J(x))^T g(x) + (J^T J + \mu I - \mu I)\Delta x) = \frac{1}{2}\Delta x^T (\mu \Delta x - J(x)^T g(x)). \tag{5.23}$$

Stoga će se nadalje koristiti ovaj kraći izraz.

Postoji još jedna bitna stvar koju je moguće poboljšati u algoritmu. Naime, ako pobliže pogledamo izraz 5.20 onda možemo vidjeti kako se svi elementi jedinične matrice I uniformno skaliraju faktorom μ. To znači da će pomak po svim elementima gradijenta biti jednak. To može biti problematično kada su pojedini članovi gradijenti relativno maleni te se onda po tim dimenzijama postupak ne bi pomaknuo puno. Zato je predloženo da se ipak iskoristi informacija o zakrivljenosti funckije, i da se umjesto s jediničnom matricom taj parametar pomnoži s izrazom $diag(J^T J)$, odnosno od dijagonalnih elemenata matričnog umnoška $J^T J$. Na taj način svi elementi gradijenta neće biti jednako skalirani te će se postupak moći više pomaknuti u smjeru manjih gradijenata. Zbog toga se umjesto izraza 5.20 češće koristi izraz

$$\Delta x = (J^T J + \lambda diag(J^T J))^{-1} J^T g, \tag{5.24}$$

Algoritam 5.2 prikazuje korake Levenberg-Marquardt metode.

Algoritam 5.2: Algoritam Levenberg Marquardt metode

Function `Simpleks(`f, ε_1, ε_2, T`)`**:**
  Podatci: f: funkcija koja se minimizira
    x_0: početna točka
    ε_1: preciznost za uvjet zaustavljanja
    ε_2: preciznost za uvjet zaustavljanja
    k_{max}: maksimalni dozvoljeni broj iteracija
  $k = 0$;
  $u = 2$;
  $\mu = T * \max_i((J(x)^T J(x))_{ii})$;
  ponavljaj
    $k = k + 1$;
    $\Delta x = -(J(x)^T J(x) + \mu I)^{-1} J(x)^T G(x)$;
    ako $||\Delta x|| \leq \varepsilon_2(||x|| + \varepsilon_2)$ **onda**
      prekini;
    inače
      $x_n = x + \Delta x$;
      $\rho = \frac{f(x) - f(x_n)}{\frac{1}{2}\Delta x^T(\mu \Delta x - J(x)^T g(x)}$;
      ako $\rho > 0$ **onda**
        $x = x_n$;
        $\mu = \mu \max(\frac{1}{3}, 1 - (2\rho - 1)^3)$;
        $u = 2$;
      inače
        $\mu = \mu * u$;
        $u = u * 2$
  dok ne bude $J(x)^T g(x) <$€;
  vrati x ;

POGLAVLJE 6

Optimiranje uz ograničenja

Mnogi optimizacijski problemi ne uključuju samo pronalazak optimalnog rješenja, već onoga rješenja koje zadovoljava određene uvjete postavljene nad pojedinim varijablama. Primjerice, u proizvodnim djelatnostima potrebno je pronaći najbolji omjer broja pojedinih proizvoda koji će donijeti najveću moguću zaradu, ali uzimajući u obzir ograničenja radne snage, materijala te mogućnosti plasiranja proizvoda. U problemima raspoređivanja aktivnosti u vremenu, potrebno je uzeti u obzir moguća ograničenja u redoslijedu aktivnosti, količini sredstava na raspolaganju za pojedinu aktivnost, nužne rokove dovršetka aktivnosti i drugo. Zbog toga se u optimizacijskim problemima vrlo često definiraju ograničenja koja traženo rješenje mora zadovoljiti.

Ograničenja obično dijelimo na dva oblika: eksplicitna i implicitna. *Eksplicitna* ograničenja javljaju se u obliku intervala za pojedine varijable

$$x_i \in [xd_i, xg_i]\,, \tag{6.1}$$

gdje xd_i predstavlja donju, dok xg_i predstavlja gornju granicu dozvoljenog područja za varijablu x_i. *Implicitna* ograničenja se s druge strane dijele na ograničenja jednakosti

$$h(\boldsymbol{x}) = 0 \tag{6.2}$$

te ograničenja nejednakosti

$$g(\boldsymbol{x}) \geq 0, \tag{6.3}$$

gdje su $h()$ i $g()$ proizvoljne funkcije.

Općenito, optimizacijski problem s ograničenjima možemo zapisati kao

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{uz} \quad & x_i \in [xd_i, xg_i]\,, \quad i = 1, \ldots, n \\ & g_j(\boldsymbol{x}) \geq 0, \qquad j = 1, \ldots, m \\ & h_k(\boldsymbol{x}) = 0, \qquad k = 1, \ldots, o \end{aligned} \tag{6.4}$$

gdje n predstavlja broj varijabli, m broj ograničenja nejednakosti i o broj ograničenja jednakosti.

Vidljivo je kako se eksplicitna ograničenja mogu predstaviti implicitnim ograničenjima u obliku nejednakosti, te se zbog toga mogu doimati redundantnima. No eksplicitna ograničenja se često tretiraju zasebno jer je trivijalno odrediti kada nisu zadovoljena, ali i lako popraviti potencijalno rješenje ako pojedine varijable ne zadovoljavaju ograničenje (primjerice, pomakom na najbližu granicu). Zbog toga, mnogi algoritmi eksplicitna i implicitna ograničenja razmatraju odvojeno. Najvažnije je primijetiti kako je eksplicitna ograničenja moguće vrlo jednostavno ukloniti; naime, ako je zadano eksplicitno ograničenje

$$x_i \in [xd_i, xg_i], \tag{6.5}$$

to se ograničenje može odstraniti uvođenjem nove neograničene varijable y_i, koja poprima vrijednosti iz intervala $[-\infty, \infty]$. Tada se umjesto varijable x_i optimira nadomjesna varijabla y_i koja se prilikom svake evaluacije funkcije cilja prethodno transformira u ograničenu varijablu x_i. Ovo je moguće postići na više načina; primjerice, korištenjem jednostavne linearne transformacije i pomoćne funkcije $f()$:

$$x_i = xd_i + (xg_i - xd_i)f(y_i). \tag{6.6}$$

Funkcija $f(y_i)$ je proizvoljna funkcija koja uvijek poprima vrijednost iz intervala $[0, 1]$. Na taj način osigurano je da će se bilo koja realna vrijednost (y_i) preslikati u dozvoljeni interval $[xd_i, xg_i]$. U tu svrhu mogu se iskoristiti razne funkcije, kao na primjer

$$f(y_i) = sin^2(y_i) \tag{6.7}$$

$$f(y_i) = \frac{1}{\pi} arctg(y_i) + \frac{1}{2} \tag{6.8}$$

$$f(y_i) = \frac{1}{1 + e^{-\sigma y_i}}, \tag{6.9}$$

gdje σ predstavlja koeficijent kojim je moguće kontrolirati strminu posljednje funkcije. Varijabla y_i može poprimiti sve realne vrijednosti te nije potrebno paziti na vrijednosti koje ona poprima tijekom optimizacije, jer se u konačnici skalira u dozvoljeno područje za varijablu x_i i time eksplicitna ograničenja neće biti narušena. Na ovaj način eksplicitna se ograničenja uvijek mogu ukloniti, te je u tom slučaju moguće koristiti prethodno opisane postupke optimizacije koji ne uzimaju ograničenja u obzir.

Slika 6.1 prikazuje tri navedene pomoćne funkcije. Prva funkcija koja koristi kvadrat sinusa je periodična, što može biti problematično u pojedinim situacijama jer će funkcija imati jednaku vrijednost za više točaka. Druge dvije funkcije nisu periodične te su na cijeloj domeni monotono rastuće. Iako izgledaju slično, postoji jedna bitna razlika; funkcija definirana uz pomoć trigonometrijske funkcije arkus tangens puno sporije teži rubnim vrijednostima -1 i 1 u usporedbi sa sigmoidom. Naime, funkcija koja koristi arkus tangens teži prema rubovima slično kao $\frac{1}{x}$, dok sigmoida teži eksponencijalno (što je i vidljivo iz njene definicije), pa se shodno tome i brže približava asimptotama. Sigmoidu je doduše moguće skalirati zadanim faktorom kako bi se usporilo ili ubrzalo ketanje prema asimptotskim vrijednostima, no zbog eksponencijalnog rasta u jednom će trenutku nadmašiti iznos funkcije arkus tangens.

Kod eksplicitnih ograničenja moguća su i dva posebna slučaja: kada je zadana samo donja granica $x_i \in [xd_i, \infty]$ ili samo gornja granica $x_i \in [-\infty, xg_i]$. Ograničenje u obliku $x_i \in [xd_i, \infty]$ možemo jednostavno ukloniti uvođenjem nove varijable y_i koja poprima sve realne vrijednosti te koju pretvaramo u varijablu x_i korištenjem nekih od sljedećih izraza:

$$x_i = f(y_i) = xd_i + y_i^2 \tag{6.10}$$

$$x_i = f(y_i) = xd_i + |y_i| \tag{6.11}$$

$$x_i = f(y_i) = xd_i + e^{y_i}. \tag{6.12}$$

Jednako tako mogu se definirati i izrazi za slučaj kada je varijabla x_i definirana na intervalu $[-\infty, xg_i]$; jedina razlika je što se sada od gornje granice oduzima neka monotono rastuća funkcija

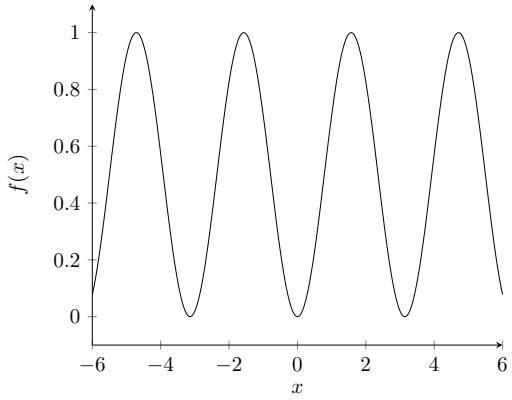
$$x_i = f(y_i) = xg_i - y_i^2 \tag{6.13}$$

$$x_i = f(y_i) = xg_i - |y_i| \tag{6.14}$$

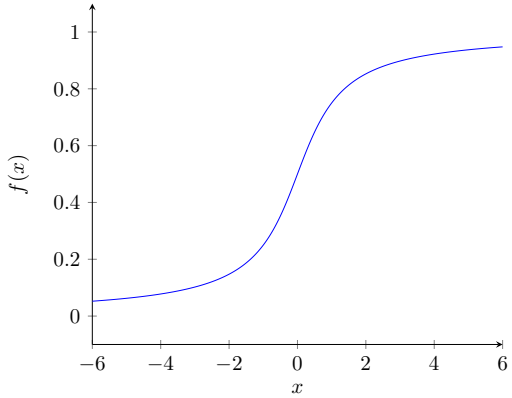
$$x_i = f(y_i) = xg_i - e^{y_i}. \tag{6.15}$$

Kod ove transformacije ideja je da je funkcija f koja pretvara varijablu y u x uvijek pozitivna te da se njena vrijednost nadodaje na donju granicu u prvom, odnosno oduzima od gornje granice u drugom slučaju.

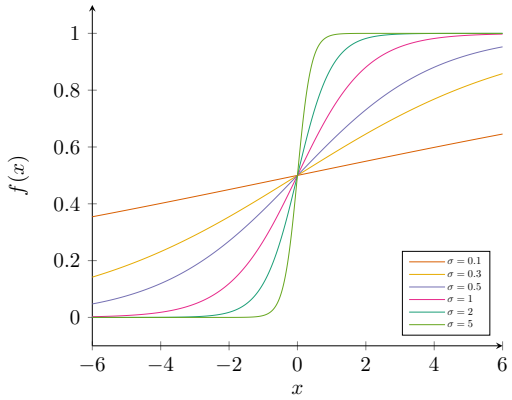
Dodavanjem ograničenja oblika jednakosti smanjuje se stupanj slobode optimizacijskog problema.



(a) Funkcija $f(y_i) = sin^2(y_i)$

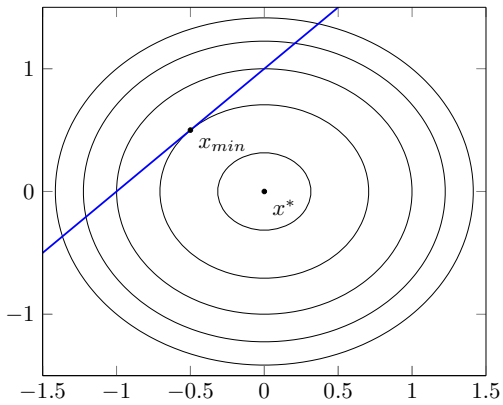


(b) Funkcija $f(y_i) = \frac{1}{\pi} arctg(y_i) + \frac{1}{2}$

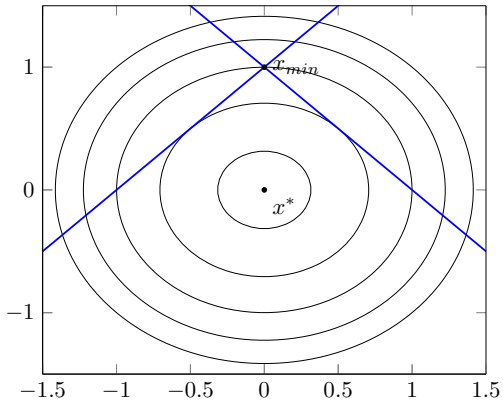


(c) Funkcija $f(y_i) = \frac{1}{1+e^{-\sigma y_i}}$

Slika 6.1: Primjeri pomoćnih funkcija za uklanjanje eksplicitnih ograničenja



(a) Primjer problema s jednim ograničenjem jednakosti



(b) Primjer problema s dva ograničenja jednakosti

Slika 6.2: Primjer optimizacijskog problema s ograničenjima jednakosti

Time se dozvoljeno područje u kojemu se mogu pronaći rješenja uvelike reducira. Osim toga, u većini slučajeva dodavanjem ograničenja jednakosti položaj minimuma optimizacijskog problema se mijenja (osim u iznimnim slučajevima kada minimum funkcije cilja već zadovoljava sva zadana ograničenja jednakosti), tako da ona generalno utječu na položaj minimuma optimizacijskog problema. Iako se ograničenjima jednakosti smanjuje stupanj slobode problema, to nikako ne znači da optimizacijski problem postaje jednostavniji. Naime, problem s ograničenjima jednakosti jest što je teško odrediti točku za koju su sva postavljena ograničenja zadovoljena. Osim toga, teško je natjerati postupak da se fokusira na pretraživanje samo onih točaka koje zadovoljavaju ograničenja. Primjer utjecaja ograničenja jednakosti na položaj minimuma prikazan je na slici 6.2. Prva slika prikazuje optimizacijski problem kada je zadano samo jedno ograničenje jednakosti u obliku jednadžbe pravca. U tom slučaju potencijalna rješenja se nužno moraju nalaziti na tom pravcu. Kako minimum funkcije x^* ne zadovoljava ograničenje jednakosti, onda se pravi minimum nalazi u točki x_{min} koja predstavlja točku koja zadovoljava ograničenje jednakosti i nalazi se najbliže minimumu funkcije cilja (budući da su konture funkcije cilja kružnice). Druga slika prikazuje primjer u kojem je dodano još jedno ograničenje jednakosti. U ovom primjeru može se vidjeti kako postoji samo jedna točka koja zadovoljava oba ograničenja te ona predstavlja minimum optimizacijskog problema. Nažalost, kako se dozvoljeno područje smanjuje, tako je i algoritmima pretraživanja teže pretraživati samo u dozvoljenom području i pronaći optimalno rješenje.

Svako ograničenje jednakosti moguće je zapisati i u obliku dva ograničenja nejednakosti. To možemo

napraviti tako da uvedemo toleranciju ε te ograničenje jednakosti $h_i(\boldsymbol{x})$ zapišemo kao

$$|h_i(\boldsymbol{x})| \leq \varepsilon, \tag{6.16}$$

odnosno nakon razdvajanja na dva slučaja

$$h_i(\boldsymbol{x}) \leq \varepsilon \tag{6.17}$$

$$h_i(\boldsymbol{x}) \geq -\varepsilon. \tag{6.18}$$

Kada je $\varepsilon = 0$ tada bi se ograničenja nejednakosti u potpunosti ponašala kao ograničenje jednakosti, odnosno problem bi imao rješenje koje se nalazi točno na granici oba ograničenja. No uz pomoć ε možemo postaviti određenu toleranciju na iznos odstupanja od ograničenja jednakosti. Naravno, što je dozvoljeno odstupanje veće, to će i postupcima biti jednostavnije pronaći rješenje koje zadovoljava ograničenja.

Za razliku od ograničenja jednakosti, ograničenja nejednakosti ne smanjuju stupanj slobode optimizacijskog problema, već samo smanjuju veličinu dozvoljenog područja. Ograničenja nejednakosti su puno manje restriktivna, budući da dijele prostor rješenja na dva dijela. Također, ovisno o položaju ograničenja nejednakosti ona mogu, a i ne moraju, utjecati na položaj minimuma optimizacijskog problema. Primjer ove pojave prikazan je na slici 6.3. Gornja slika prikazuje situaciju u kojoj se minimum funkcije cilja već nalazi unutar dozvoljenog područja, što znači da nametanje ograničenja ne utječe na položaj minimuma te on "ostaje" na istom mjestu. S druge strane, slika 6.3b prikazuje primjer u kojemu se minimum funkcije cilja nalazi u nedozvoljenom području. U tom slučaju, minimum optimizacijskog problema s ograničenjima nalazi se negdje na granici ograničenja nejednakosti, odnosno u onoj točki na granici koja ima najmanju funkcijsku vrijednost. Općenito vrijedi sljedeće: nametanjem ograničenja može se ili ne mora promijeniti položaj rješenja optimizacijskog problema u odnosu na optimum funkcije cilja bez ograničenja. No ukoliko se položaj rješenja mijenja nametanjem ograničenja, te u slučaju unimodalne funkcije cilja, rješenje optimizacijskog problema uvijek leži na *granici* dozvoljenog područja. Ovu činjenicu koriste neki od optimizacijskih postupaka za pronalaženje rješenja s ograničenjima.

Postupci za rješavanje optimizacijskih problema s ograničenjima mogu se u pravilu podijeliti na dvije skupine:

- postupci koji transformiraju problem u oblik *bez* ograničenja te omogućuju primjenu postojećih postupaka optimiranja;
- postupci koji uzimaju postojanje ograničenja u obzir tijekom svakog koraka.

U nastavku će biti prikazani predstavnici obje vrste pristupa rješavanju problema s ograničenjima.

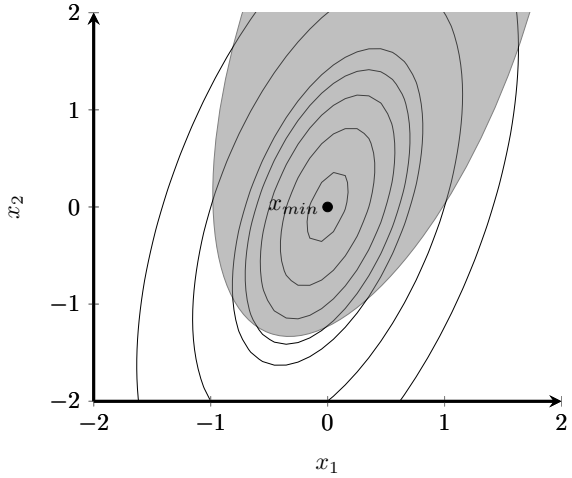
6.1 Traženje unutarnje točke

Kod optimizacijskih postupaka koji razmatraju ograničenja nejednakosti, često je potrebno započeti s točkom koja se nalazi unutar područja koje zadovoljava sva ograničenja nejednakosti. Ako zadana početna točka ne zadovoljava ovaj uvjet, potrebno je odrediti novu početnu točku prije primjene postupka optimizacije. Pretpostavimo da su ograničenja nejednakosti zadana u obliku

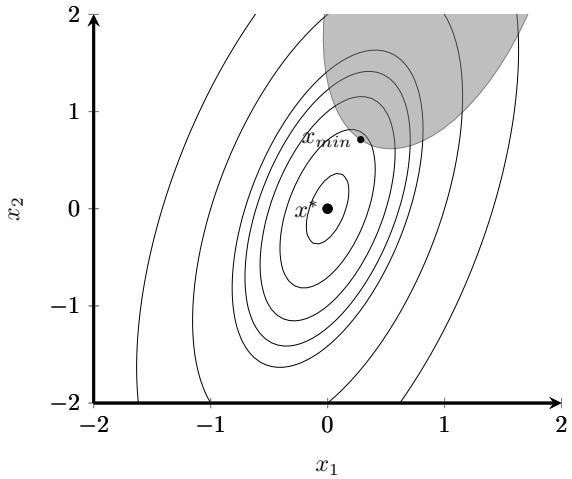
$$g_i(\boldsymbol{x}) \geq 0. \tag{6.19}$$

Drugim riječima, iznos ograničenja bit će negativan dok god ograničenje nije zadovoljeno. Također, pretpostavljamo kako vrijednost $g_i(\boldsymbol{x})$ postaje to manja što je točka $\boldsymbol{x}$ bliža granici ograničenja nejednakosti.

Na temelju prethodnih pretpostavki, prihvatljivu početnu točku možemo odrediti tako da definiramo



(a) Prvi slučaj: minimum funkcije cilja nalazi se u dozvoljenom području



(b) Drugi slučaj: optimalno rješenje problema nalazi se na granici dozvoljenog područja

Slika 6.3: Promjena položaja optimalnog rješenja uz nametanje ograničenja (dozvoljeno područje je zasivljeno)

pomoćni optimizacijski problem s funkcijom cilja

$$G(\boldsymbol{x}) = -\sum_{i=0}^{i<n} t_i g_i(\boldsymbol{x}), \tag{6.20}$$

pri čemu je t_i definiran kao

$$t_i = \begin{cases} 0 & \text{ako } g(\boldsymbol{x}) \geq 0 \\ 1 & \text{ako } g(\boldsymbol{x}) < 0 \end{cases}. \tag{6.21}$$

Ideja ovog pomoćnog problema, za kojega možemo primijeniti bilo koji algoritam optimizacije, jest zbrajanje iznosa svih ograničenja nejednakosti koja u trenutno promatranoj točki $\boldsymbol{x}$ nisu zadovoljena. Kako se približavamo granici ograničenja, tako će se vrijednost ove funkcije smanjivati. Na taj način postupak optimizacije nastojat će točku približavati prema granici dok ne zadovoljimo sva ograničenja. Koja ograničenja su uključena u zbroj kontrolira se parametrom t_i, koji se određuje posebno za svako pojedino ograničenje. Ako ograničenje nije zadovoljeno, njegov pripadni parametar t_i jednak je 1 te se na taj način vrijednost (narušenog) ograničenja dodaje u zbroj. U suprotnom, kada je ograničenje zadovoljeno, parametar t_i postavlja se na 0 i time se ograničenje na uzima u obzir. Budući da ograničenja poprimaju negativne vrijednosti kada nisu zadovoljena, ispred zbroja stavlja se negacija kako bi se definirala funkcija cilja koju je potrebno minimizirati. Minimalna vrijednost ove pomoćne funkcije je 0; dostignemo li tu funkcijsku vrijednost, znamo da smo pronašli točku koja zadovoljava sva ograničenja. Ovaj pomoćni optimizacijski problem možemo riješiti bilo kojim od prethodno opisanih postupaka optimizacije bez ograničenja.

Algoritam 6.1 prikazuje korake postupka kojim se pronalazi unutarnja točka koja zadovoljava zadana ograničenja nejednakosti.

Algoritam 6.1: Algoritam traženja unutarnje točke

```
1  Funkcija UnutarnjaTočka(g, h, t, x_0):
       Podatci: g: polje ograničenja nejednakosti
                x_0: početna točka
2      x = x_0;
3      ponavljaj
4          x_s = x ;
5          x = arg min(G(g, x)) ;
6      dok ne bude ||x_s − x||;
7      vrati x;

8  G(g, x):
       Podatci: g: polje ograničenja nejednakosti
                x: točka za koju se računa vrijednost funkcije
9      G = 0 ;
10     za g_i ∈ g čini
11         ako g_i(x) < 0 onda
12             G = G − g_i(x);
13     vrati G;
```

6.2 Metode kazne i barijere

Najjednostavniji način rješavanja problema s ograničenjima jest definiranje nadomjesne funkcije cilja koja na neki način uključuje zadana ograničenja; drugim riječima, originalni problem s ograničenjima transformira se u problem optimizacije bez ograničenja. Na taj način omogućuje se korištenje bilo kojeg postupka optimizacije koji ne uzima ograničenja u obzir. Uz pretpostavku minimizacije, transformacija se temelji na jednostavnoj ideji: ako neka točka ne zadovoljava ograničenja, pozitivni iznos nezadovoljenja dodaje se na funkciju cilja. Ovaj iznos obično je proporcionalan udaljenosti od dozvoljenog područja te doprinosi rastu funkcije cilja, pa se optimizacijom traži minimalna vrijednost, odnosno približavanje dozvoljenom području. S druge strane, ako su ograničenja zadovoljena, njihov utjecaj na vrijednost nadomjesne funkcije trebao bi biti zanemariv.

Metoda kazne koristi se kada je potrebno zadovoljiti neka ograničenja jednakosti. Nadomjesna funkcija kod metode kazne definira se kao

$$F(\boldsymbol{x}, t) = f(\boldsymbol{x}) + t \sum_i h_i^2(\boldsymbol{x}), \tag{6.22}$$

gdje f predstavlja originalnu funkciju cilja, h_i ograničenje jednakosti, a parametar t određuje utjecaj nezadovoljenja ograničenja jednakosti. Iz definicije je vidljivo kako se ograničenja jednakosti ugrađuju u nadomjesnu funkciju tako da se njihova vrijednost kvadrira. Umjesto kvadratne funkcije može se koristiti i neki drugi oblik, uz uvjet da vraća iznos 0 kada je ograničenje zadovoljeno, odnosno pozitivnu vrijednost koja se povećava što je udaljenost od ograničenja veća. Kvadratna funkcija je iznimno pogodna jer je glatka i derivabilna, što olakšava traženje minimuma te dozvoljava korištenje postupaka temeljenih na gradijentima. Slika 6.4 prikazuje izgled nadomjesnih funkcija cilja uz ograničenje oblika pravaca koji prepolavljaju sva četiri kvadranta ($x_0^2 - x_1^2 = 0$). U gornjoj slici prikazano je korištenje kvadratne funkcije za ugradnju ograničenja u nadomjesnu funkciju, odnosno apsolutne vrijednosti u donjoj slici. Korištenjem apsolutne vrijednosti dobivamo nadomjesnu funkciju s oštrim rubovima, što može djelovati nepovoljno na optimizacijske postupke. S druge strane, korištenjem kvadrata nadomjesna funkcija zadržava svoju zakrivljenost.

U nadomjesnoj funkciji parametar t igra važnu ulogu. Ovaj parametar određuje koliki se naglasak stavlja na ograničenja u odnosu na originalnu funkciju cilja. Ako je parametar t postavljen na premalu vrijednost, ograničenja neće imati dovoljno velik utjecaj te dobiveno rješenje neće nužno zadovoljavati ograničenja.

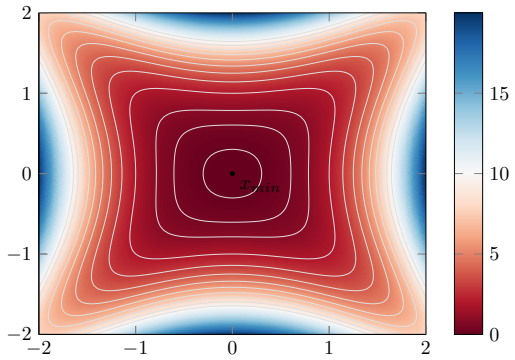
Slika 6.5 prikazuje utjecaj parametra t na izgled, ali i položaja minimuma nadomjesne funkcije. Primerice, za vrijednost $t = 0.1$ ograničenje ima jako maleni utjecaj i minimum nadomjesne funkcije je daleko od minimuma problema s ograničenjima, koji se nalazi u točki $\boldsymbol{x}_{min} = (-1, -1)$. No postupnim povećanjem vrijednosti parametra t ograničenje dobiva sve veći utjecaj i minimum nadomjesne funkcije sve se više približava traženom minimumu problema s ograničenjem.

definicija nadomjesne fje

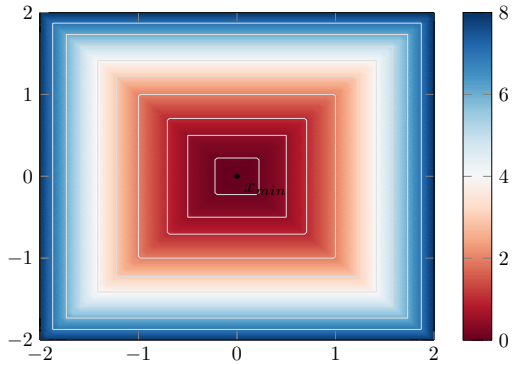
Metoda barijere primjenjuje se u slučaju problema s ograničenjima nejednakosti. Kod ove metode, nadomjesna funkcija definira se kao

$$F(\boldsymbol{x}, t) = f(\boldsymbol{x}) - \frac{1}{t} \sum_j log(g_j(\boldsymbol{x})), \tag{6.23}$$

pri čemu g_i predstavlja ograničenje nejednakosti. U slučaju ograničenja nejednakosti, ideja metode barijere je zadržavanje pretrage u dozvoljenom području. Prema tome, komponenta nadomjesne funkcije koja modelira ograničenja trebala bi rasti iznosom (idealno, težiti u beskonačnost) kako se približavamo

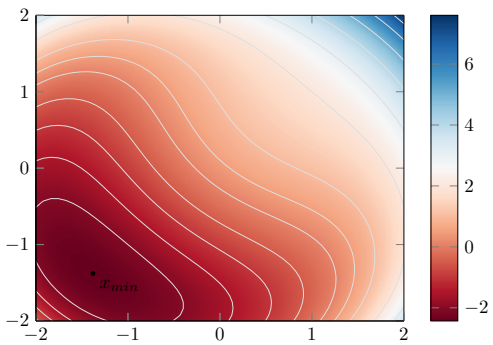


(a) Nadomjesna funkcija uz kvadrat iznosa nezadovoljenja

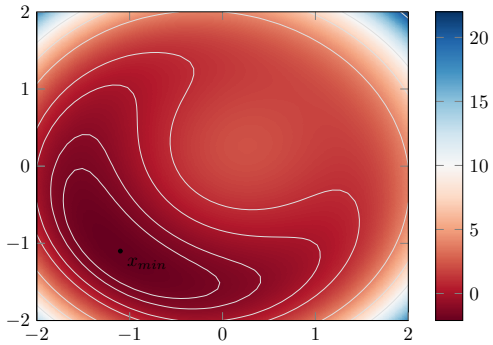


(b) Nadomjesna funkcija uz apsolutni iznos nezadovoljenja

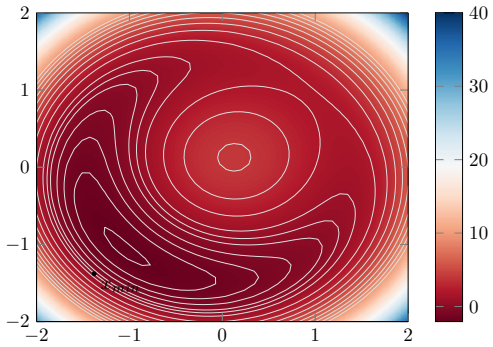
Slika 6.4: Primjer dviju funkcija kazne za funkciju cilja $f(\boldsymbol{x}) = x_0^2 + x_1^2$ i ograničenja $x_0^2 - x_1^2 = 0$



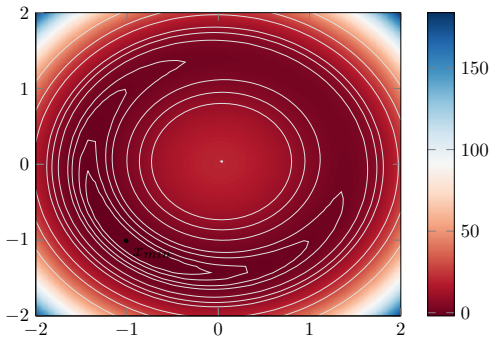
(a) Prvi slučaj



(b) Drugi slučaj

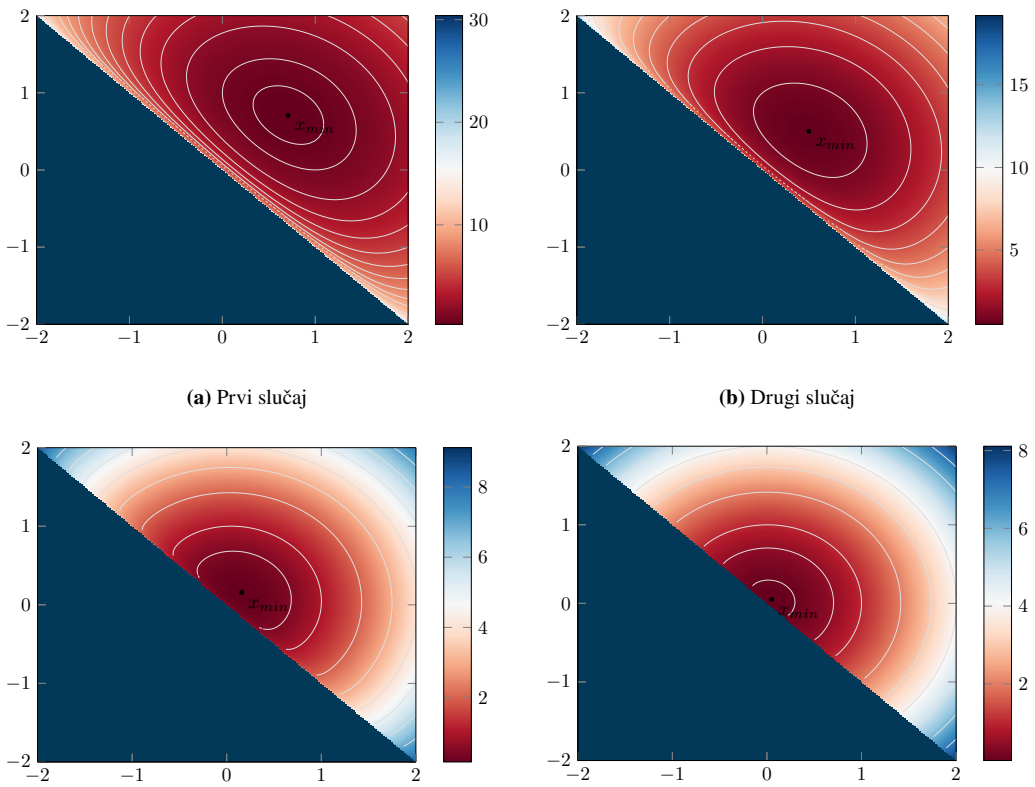


(c) Treći slučaj



(d) Četvrto slučaj

Slika 6.5: Nedostaje definicija nadomjesne funkcije



(a) Prvi slučaj

(b) Drugi slučaj

(c) Treći slučaj

(d) Četvrti slučaj

Slika 6.6: Utjecaj parametra t na nadomjesnu funkciju $F(\boldsymbol{x}, t) = x_0^2 + x_1^2 - \frac{1}{t}log(-x_0 - x_1)$.

granici dozvoljenog područja i na taj način odvraća postupak pretrage od granice. Jedna od funkcija koja odgovara ovim zahtjevima je logaritamska funkcija jer vrijednost logaritma teži u $-\inf$ kada se argument logaritamske funkcije približava 0. Budući da želimo da se vrijednost povećava prema $\inf$, ispred sume s ograničenjima nejednakosti stavljamo minus kako bi promijenili predznak. S druge strane, kada je ograničenje zadovoljeno, logaritamska funkcija će neznatno utjecati na originalnu funkciju cilja jer prigušuje velike vrijednosti koje se mogu dobiti ograničenjima nejednakosti.

Ovdje je odmah moguće uočiti jedan problem; ako minimum leži na granici dozvoljenog područja (što često i jest slučaj u problemima s ograničenjem), ovakva definicija nadomjesne funkcije neće dozvoliti da dođemo do njega. Taj se problem, slično kao kod metode kazne, rješava parametrom t koji određuje utjecaj ograničenja. Povećavamo li vrijednost parametra t, utjecaj ograničenja nejednakosti će opadati i time se možemo približiti granici na proizvoljno malenu udaljenost.

Prilikom implementacije ovoga pristupa, potrebno je definirati što se događa u slučajevima kada se nadomjesna funkcija računa za točku koja ne zadovoljava ograničenja nejednakosti. Logaritamska funkcija za takve slučajeve nije definirana, pa se u tom slučaju obično pretpostavlja da cijela nadomjesna funkcija vraća vrijednost $\inf$. Na taj način postupci optimizacije u slučaju izlaska izvan dozvoljenog prostora dobivaju funkcijske vrijednosti koje su nužno veće nego one od točaka unutar dozvoljenog područja.

Slika 6.6 prikazuje utjecaj vrijednosti parametra t na metodu barijere primijenjenu na sfernu funkciju cilja $f(\boldsymbol{x}) = x_0^2 + x_1^2$ te ograničenje $g(\boldsymbol{x}) = -x_0 - x_1 > 0$.

Konačno, često se javlja situacija kada su u problemu optimizacije prisutna ograničenja jednakosti i nejednakosti. Tada je moguće definirati nadomjesnu funkciju korištenjem metode kazne i barijere,

odnosno

$$F(\boldsymbol{x}, t) = f(\boldsymbol{x}) + t \sum_i h_i^2(\boldsymbol{x}) - \frac{1}{t} \sum_j log(g_j(\boldsymbol{x})). \tag{6.24}$$

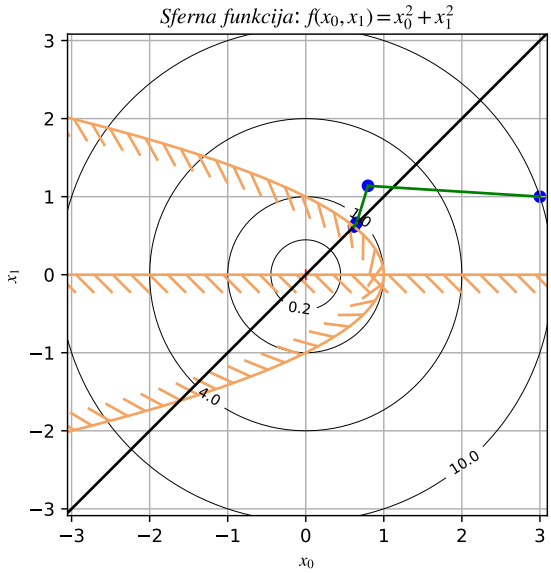
Uočimo kako je isti parametar t moguće koristiti za određivanje utjecaja obaju vrsti ograničenja (uz korištenje recipročne vrijednosti kod ograničenja nejednakosti). Ovaj odabir je prikladan, jer kada stavljamo veći naglasak na ograničenja jednakosti s ciljem pronalaska rješenja koje ih zadovoljava, ujedno želimo i da ograničenja nejednakosti imaju što manji utjecaj kako bi se što više približili njihovoj granici, ako na njoj leži minimum optimizacijskog problema.

Budući da utjecaj iznosa ograničenja nije unaprijed poznat, položaj optimuma nadomjesne funkcije cilja ovisit će o odabranoj vrijednosti parametra t. Zbog toga se u praktičnoj primjeni obično kreće s manjim početnim iznosom tog parametra, te se optimiranje provodi opetovano uz povećavanje t u svakoj novoj iteraciji. Pri tome, rješenje koje je optimizacijski postupak pronašao za prethodnu vrijednost parametra t koristi se kao početna točka u optimiranju uz novu vrijednost t. Ponovljeno optimiranje uz povećavanje parametra t provodi se dok razlika optimuma nadomjesne funkcije ne postane dovoljno mala (kako je prikazano u algoritmu 6.2) ili do dovoljno velike vrijednosti t.

Algoritam 6.2: Algoritam optimizacije uz metodu kazne i barijere

```
1  Funkcija KaznaBarijera(f, g, h, t, x_0):
       Podatci: f: funkcija koja se minimizira
                g: polje ograničenja nejednakosti
                h: polje ograničenja jednakosti
                t_0: početna vrijednost parametra t koju određuje jačinu utjecaja ograničenja
                x_0: početna točka
                ε: preciznost
2      x = x_0;
3      t = t_0;
4      F = f(x);
5      x_P = x_B = x_0;
6      ponavljaj
7          x_s = x ;
8          x = arg min(F(f, g, h, t, x)) ;
9          t = 10 * t ;
10     dok ne bude ||x_s − x|| < ε;
11     vrati x;

12 F(f, g, h, t, x):
       Podatci: f: funkcija koja se minimizira
                g: polje ograničenja nejednakosti
                h: polje ograničenja jednakosti
                t: parametar koji određuje utjecaj pojedinih ograničenja
                x: točka za koju se računa vrijednost funkcije
13     F = f(x);
14     za g_i ∈ g čini
15         ako g_(x) ≤ 0 onda
16             vrati ∞;
17         inače
18             F = F − (1/t) ln(g_i) ;
19     za h_i ∈ h čini
20         F = F + t(h_i)^2 ;
21     vrati F;
```



Slika 6.7: Prikaz optimizacije problema s ograničenjima metodom kazne i barijere

Primjenu ovog postupka možemo pokazati na sljedećem primjeru optimizacijskog problema s ograničenjima:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) = x_0^2 + x_1^2 \\ & g_0(\boldsymbol{x}) : x_0 - 1 + x_1^2 \geq 0 \\ & g_1(\boldsymbol{x}) : x_1 \geq 0 \\ & h_0(\boldsymbol{x}) : x_1 - x_0 = 0 \end{aligned} \tag{6.25}$$

Primjer implementacije postupka prikazan u programu **??**; u ovom programu, koristimo algoritam po Nelderu i Meadu za optimiranje nadomjesne funkcije cilja (moguće je koristiti i neki drugi postupak bez ograničenja). Pokretanjem programa dobiva se sljedeći ispis:

```
t: 1, minimum: [0.8068145962061494, 1.0121817244796416]
t: 10, minimum: [0.6519846061511476, 0.6606345846392545]
t: 100, minimum: [0.6216872186140838, 0.6223623569808703]
t: 1000, minimum: [0.6184013024814232, 0.6184667915504598]
t: 10000, minimum: [0.6180713998344347, 0.6180771155427742]
t: 100000, minimum: [0.6180352191166382, 0.6180408214566979]
Rezultat: f([0.6180314149885924, 0.6180370223805335]) = 0.7639325909457977
```

Iz ispisa je vidljivo kako se minimum nadomjesne funkcije cilja uz povećanje parametra t postupno približava rješenju problema s ograničenjima, koji se nalazi na sjecištu pravca definiranog ograničenjem h_0 i parabole g_0. Ako je potrebno, parametar t može se dodatno povećati kako bi se postupak približio na još manju udaljenost od granice dozvoljenog područja. Ilustracija problema s označenim rezultatima optimiranja u svakom koraku prikazana je na slici 6.7.

6.3 Boxov postupak

Prethodno opisane metode kazne i barijere primjer su transformacije problema u oblik bez ograničenja, što omogućuje primjenu bilo kojeg postupka optimiranja koji ne uzima ograničenja u obzir. Boxov pos-

tupak je, s druge strane, jedan od predstavnika postupaka oblikovanih uzimajući postojanje ograničenja u obzir. Boxov postupak je optimizacijska metoda temeljena na metodi simpleksa po Nelderu i Meadu, no prilagođena za rad s ograničenjima nejednakosti i eksplicitnim ograničenjima. Naime, tijekom rada osigurava se da sve točke zadovoljavaju ograničenja; stvori li se operacijama algoritma neka nova točka izvan dozvoljenog područja, algoritam ju iterativno pokušava vratiti u dozvoljeno područje.

Boxov postupak započinje na sličan način kao i simpleks: zadana je jedna početna točka uz pomoću koje se stvara početni skup točaka. Dodatno, pretpostavlja se da početna točka zadovoljava sva ograničenja. Ako takva točka nije dostupna, moguće ju je pronaći primjenom postupka traženja unutarnje točke opisanog na početku poglavlja. Nakon toga, svaka dodatna točka definira se uz pomoć izraza

$$x_{i,j} = x_{dj} + r_{ij}(x_{gj} - x_{dj}), \tag{6.26}$$

pri čemu i označava indeks nove točke, j redni broj dimenzije točke, x_{dj} donju granicu dimenzije j, x_{gj} gornju granicu dimenzije j, te r_{ij} slučajni broj iz intervala $[0, 1]$. Točke stvorene na ovaj način zadovoljit će sva eksplicitna ograničenja, no ne nužno i ograničenja nejednakosti. Ako za novodefiniranu točku ograničenja nejednakosti nisu zadovoljena, provodi se operacija slična kontrakciji u simpleksu: točka koja ne zadovoljava ograničenje nejednakosti približava se na polovicu udaljenosti do centroida skupa svih *do tada prihvaćenih* točaka. Navedeno se može opisati izrazom

$$\boldsymbol{x}_i = \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_c) \tag{6.27}$$

I nakon približavanja centroidu moguće je da točka još uvijek ne zadovoljava ograničenje nejednakosti. U tom slučaju, točka se jednostavno opetovano približava centroidu (na pola razmaka) dok god ne zadovolji ograničenja nejednakosti.

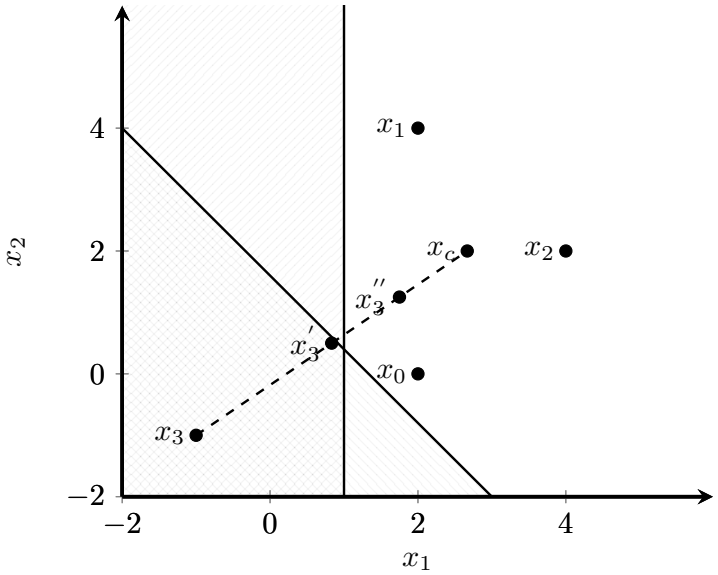
Primijetimo kako postupak popravljanja pretpostavlja da se i sam centroid do tada prihvaćenih točaka nalazi u dozvoljenom području, što će biti ispunjeno ako ograničenja nejednakosti zatvaraju *konveksan prostor*. Ovaj uvjet je svakako ispunjen kada dozvoljeno područje postoji i ako su sva ograničenja nejednakosti linearna. No ukoliko su zadana i nelinearna ograničenja, moguće je da se točka centroida ne nalazi u dozvoljenom području. U tom slučaju postoji velika vjerojatnost da algoritam zapne u beskonačnoj petlji, jer će točku koja ne zadovoljava ograničenja nejednakosti pokušavati bezuspješno približavati prema centroidu.

Dva primjera izgradnje početnog skupa točaka prikazana su na slici 6.8; granice dozvoljenog područja (bijelo) predstavljene su debljim linijama, a dosada prihvaćene točke su $\boldsymbol{x}_0$, $\boldsymbol{x}_1$ i $\boldsymbol{x}_2$. Nova slučajno stvorena točka $\boldsymbol{x}_3$ zadovoljava eksplicitna, ali ne i implicitna ograničenja. U slučaju linearnih ograničenja u gornjem primjeru, opetovano približavanje prema centroidu dovest će nakon dva koraka točku $\boldsymbol{x}_3^{''}$ u dozvoljeno područje. U donjem primjeru, međutim, niti centroid do sada prihvaćenih točaka ne leži u dozvoljenom području, pa postupak popravljanja točke $\boldsymbol{x}_3$ neće biti uspješan. Otkrije li se ovakvo ponašanje, moguće je ponoviti postupak stvaranja početnog skupa točaka ili započeti s drugom početnom točkom koja zadovoljava ograničenja. U općenitom slučaju, nekonveksnost dozvoljenog područja može uzrokovati nemogućnost primjene algoritma.

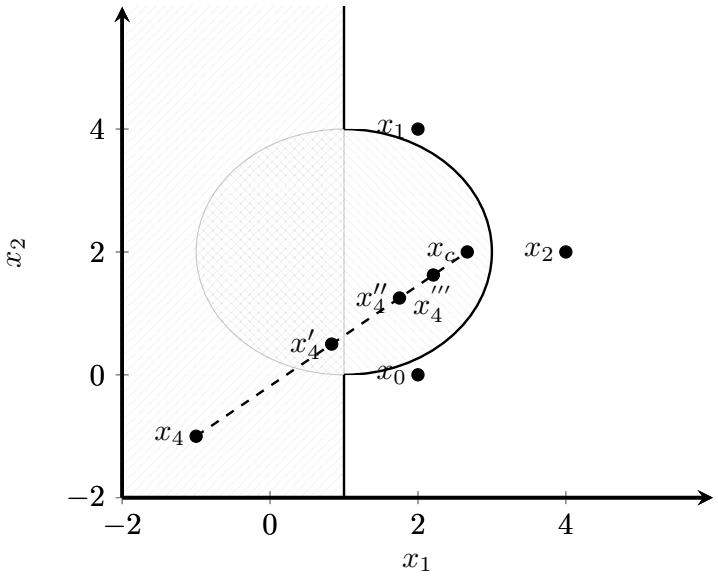
Nakon što je generiran početni skup točaka, započinje glavni dio algoritma koji se ponavlja do zadanog uvjeta zaustavljanja. U svakoj iteraciji određuje se najbolja točka iz skupa, $\boldsymbol{x}_l$, najgora točka $\boldsymbol{x}_h$, te računa centroid svih točaka osim najgore točke $\boldsymbol{x}_h$. Nakon toga, provodi se operacija refleksije:

$$\boldsymbol{x}_r = (1 + \alpha)\boldsymbol{x}_c - \alpha\boldsymbol{x}_h. \tag{6.28}$$

Početni dio iteracije identičan je postupku simpleksa po Nelderu i Meadu, uz razliku pri odabiru vrijednosti za faktor refleksije. Kako Boxov postupak ne koristi operaciju ekspanzije, vrijednost parametra α postavlja



(a) Prvi primjer - konveksno dozvoljeno područje



(b) Drugi primjer - nekonveksno dozvoljeno područje

Slika 6.8: Približavanje nove točke centroidu do tada prihvaćenih točaka

se obično na vrijednost koja je veća od 1, po autoru preporučeno na 1.3. Na taj način refleksija nadoknađuje nedostatak ekspanzije jer se točka $\boldsymbol{x}_h$ preslikava na malo veću udaljenost. Nakon što je izračunata točka refleksije $\boldsymbol{x}_r$, potrebno je provjeriti zadovoljava li nova točka sva ograničenja. Prvo se provjeravaju eksplicitna ograničenja; ako točka $\boldsymbol{x}_r$ ne zadovoljava eksplicitna ograničenja, postavlja se na najbližu granicu:

$$x_{ri} = x_{di} \text{ ako } x_{ri} < x_{di} \tag{6.29}$$

$$x_{ri} = x_{gi} \text{ ako } x_{ri} > x_{gi}. \tag{6.30}$$

Eksplicitna ograničenja je na ovaj način vrlo jednostavno zadovoljiti, za što nisu potrebne nikakve dodatne operacije. No, kod ovakvog popravljanja točke može doći do situacije gdje dosta točaka postaje međusobno linearno zavisno (odnosno, sve točke leže na granici istog ograničenja). Zbog toga se često točka refleksije ne postavlja izravno na samu granicu ograničenja, nego za mali slučajni iznos dodatno posmakne unutar dozvoljenog područja. Na taj način pokušava se spriječiti da skup točaka degenerira u podprostor (što je ilustrirano u nastavku).

Nakon što su zadovoljena sva eksplicitna ograničenja, prelazi se na ograničenja nejednakosti. U slučaju da neko ograničenje nije zadovoljeno, točka refleksije se približava prema centroidu na isti način kao tijekom početnog generiranja skupa točaka. Taj postupak se ponavlja dok god točka refleksije ne zadovolji sva ograničenja nejednakosti. Konačno, nakon što je dobivena točka koja zadovoljava sva ograničenja, provjerava se je li vrijednost funkcije cilja konačne točke $\boldsymbol{x}_r$ lošija od vrijednosti *druge* najgore točke u skupu točaka; drugim riječima, provjeravamo je li nova točka (dobivena refleksijom prethodno najgore točke) i dalje najgora u skupu. Ako jest, tada se točka refleksije $\boldsymbol{x}_r$ još jednom približava centroidu:

$$\boldsymbol{x}_r = \frac{1}{2}(\boldsymbol{x}_r + \boldsymbol{x}_c). \tag{6.31}$$

Ova posljednja operacija može se smatrati svojevrsnim nadomjestkom kontrakcije, budući da dodatno sažima skup točaka ako nije pronađen dobar smjer smanjenja funkcije cilja, ili ako je potrebno preciznije odrediti položaj lokalnog minimuma. Prethodno opisano ponavlja se na isti način, određivanjem novog centroida i provođenjem operacije refleksije za novu iteraciju.

Jedan od važnih parametara ovog algoritma je broj točaka koji se stvara u početnom skupu. Iako se u postupku po Nelderu i Meadu koristi skup od $n+1$ točke, u Boxovom postupku preporuča se koristiti više od toga, a uobičajeno korišteni broj točaka je $2n$. Veći broj točaka motiviran je postojanjem vjerojatnosti da radni skup točaka degenerira u podprostor, čime se gubi mogućnost pretraživanja čitavog prostora rješenja. Slika 6.9 prikazuje primjer ovakve situacije. Naime, u ovom primjeru može se vidjeti kako sve točke u skupu, nakon uzastopnih refleksija i pomicanja unutar dozvoljenog područja, leže na granici jednog ograničenja. U nastavku izvođenja algoritma skup točaka bi konvergirao prema točki $\boldsymbol{x}^{'}$, koja ne predstavlja pravi minimum problema s ograničenjima. Uz ove uvjete nije više moguće doseći točku $\boldsymbol{x}^{'}_{min}$, što je najbolje rješenje koje zadovoljava ograničenja. Problem je sličan kao i kod postupka simpleksa, jer jednom kada točke degeneriraju u podprostor, nije moguće pretražiti čitav prostor rješenja. Korištenje većeg broja točaka smanjuje mogućnost degeneracije u podprostor, no niti to ne jamči izbjegavanje ove pojave u potpunosti.

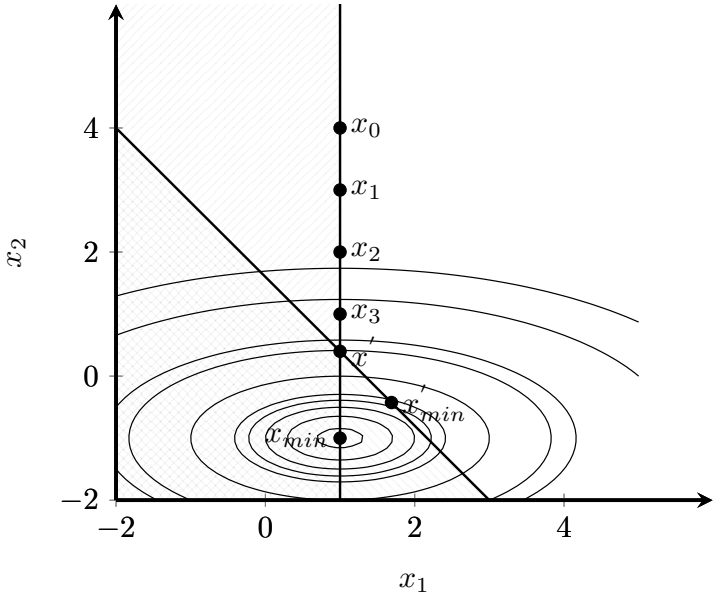
Programsko ostvarenje Boxovog postupka prikazano je u programu **??**; u retcima 77-84 definirana su implicitna ograničenja u obliku nejednakosti te eksplicitne granice. Odabrana funkcija cilja je sferna, a postupak se poziva s početnom točkom koja po pretpostavci zadovoljava zadana ograničenja. Kretanje centroida skupa točaka za prvih 20 koraka postupka prikazan je na slici 6.10. Iako je kretanje jasno vidljivo,

Algoritam 6.3: Algoritam Boxovog postupka

Function `Box(`f, x_0, α, xd, xg, g, ε`)`**:**

Podatci: f: funkcija koja se minimizira
a: početna granica unimodalng intervala
x_0: početna točka
α: faktor refleksije
xd: polje donjih granica pojedinih dimenzija
xg: polje gornjih granica pojedinih dimenzija
g: polje ograničenja nejednakosti
ε: preciznost

$x_c = x_0$;
$X[1] = x_0$;
za svaki $j = 1, 2, \ldots, 2n$ **čini**
  za svaki $i = 1, 2, \ldots, n$ **čini**
    $r = random(0, 1)$;
    $X[j]_i = xd_i + r(xg_i - xd_i)$;
  dok *postoji* g_i *za koji* $g_i(X[j]) < 0$ **čini**
    $X[j] = \frac{1}{2}(X[j] + x_c)$;
  $x_c = \frac{1}{2}\sum_{i=1}^{j} X[i]$;
ponavljaj
  $h = \arg\max_{i\in[1,n+1]}(f(X[i]))$;
  $h_2 = \arg\min_{i\in[1,n+1], i\neq h}(f(X[i]))$;
  $x_c = \frac{1}{n}\sum_{i\neq h} x_i.$;
  $x_r = (1 + \alpha)x_c - \alpha x_h$;
  za svaki $i = 1, 2, \ldots, n$ **čini**
    ako $x_{ri} < xd_i$ **onda**
      $x_{ri} = xd_i$;
    inače ako $x_{ri} > xg_i$ **onda**
      $x_{ri} = xg_i$;
  dok *postoji* g_i *za koji* $g_i(x_r) < 0$ **čini**
    $x_r = \frac{1}{2}(x_r + x_c)$;
  ako $f(x_r) > f(X[h_2])$ **onda**
    $x_r = \frac{1}{2}(x_r + x_c)$;
  $X[h] = x_r$;
dok ne bude $\sqrt{\frac{1}{2}\sum_{i=1}^{n}((f(x_i) - f(x_c))^2} < \varepsilon$;
vrati x_c ;



Slika 6.9: Primjer degeneracije dvodimenzijskog skupa točaka u podprostor

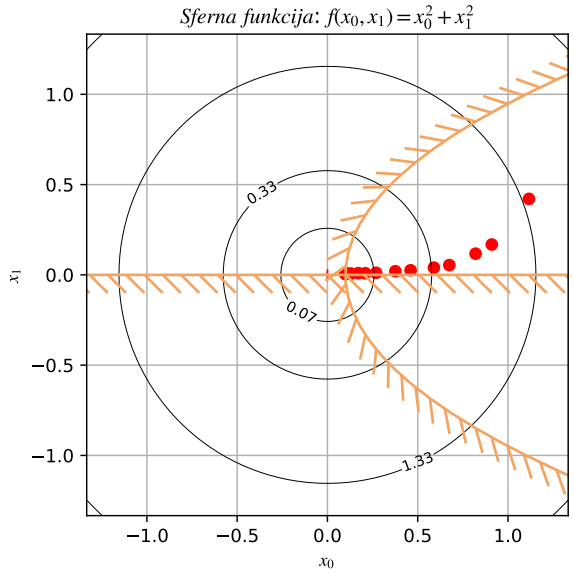
za ovaj postupak potrebno je relativno velik broj iteracija kako bi se približio optimumu optimizacijskog problema; primjerice, nakon 20 iteracija najbolja pronađena točka je $(0.100648, 0.002702)$, a nakon 100 iteracija dolazi se do točke $(0.100002, 0.000465)$. Naravno, postupak je u osnovi stohastičan (za razliku od prethodno opisane metode s nadomjesnom funkcijom cilja), pa se u svakom pokretanju očekuju manje varijacije rezultata.

6.3.1 Postupci optimiranja uz ograničenja u paketu `scipy.optimize`

Funkcija `minimize` u paketu `scipy.optimize` uključuje podršku za nekoliko postupaka rješavanja problema s ograničenjima. Jedan od dostupnih postupaka je SLSQP (engl. Sequential Least SQuares Programming), koji je programska implementacija slijednog kvadratnog programiranja (engl. SQP, Sequential Quadratic Programming). Postupak SQP temelji se na određivanju skupa aktivnih ograničenja u svakom koraku, odnosno onoga podskupa ograničenja na čijoj granici se trenutno nalazimo. Drugim riječima, aktivna ograničenja su ona čijim bi se uklanjanjem promijenila točka minimuma (dok to ne vrijedi za ograničenja koja nisu aktivna). U svakom koraku koristi se kvazi-Newtonov postupak za rješavanje problema uz zadani podskup ograničenja, te se u svakom sljedećem koraku ponovno određuje skup aktivnih ograničenja.

Prilikom uporabe, ograničenja se zadaju uz pomoć rječnika (`dictionary`) gdje se definira vrsta ograničenja (ključem 'eq' za jednakost odnosno 'ineq' za nejednakost) te funkcija ograničenja jednakosti ili nejednakosti. Ograničenja jednakosti su zadovoljena kada funkcija daje nulu, dok se ograničenja nejednakosti smatraju zadovoljenim kada funkcija daje nenegativnu vrijednost. Eksplicitna ograničenja zadaju se definiranjem granica varijabli kao niz parova, gdje svaki par specificira donju i gornju granicu neke varijable.

Pretpostavimo da želimo pronaći rješenje optimizacijskog problema s ograničenjima kao u primjeru



Slika 6.10: Prikaz optimizacije problema s ograničenjima Boxovim postupkom

6.25:

$$
\begin{aligned}
\min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) = x_0^2 + x_1^2 \\
& g_0(\boldsymbol{x}) : x_0 - 1 + x_1^2 \geq 0 \\
& g_1(\boldsymbol{x}) : x_1 \geq 0 \\
& h_0(\boldsymbol{x}) : x_1 - x_0 = 0
\end{aligned}
\tag{6.32}
$$

Rješavanje ovog problema metodom SLSQP uz pomoć funkcije `scipy.optimize.minimize` prikazano je u sljedećem programskom odsječku:

```
from scipy.optimize import minimize
import funkcije2d as f2d

# definicija ogranicenja nejednakosti
def g0(x):
    return (x[0]-0.1) + x[1]**2
def g1(x):
    return x[1]

# definicija ogranicenja jednakosti
def h0(x):
    return x[1] - x[0] - 0

# implicitna ogranicenja u obliku rjecnika
ogranicenja = [
    {'type': 'ineq', 'fun': g0},
    {'type': 'ineq', 'fun': g1},
    {'type': 'eq', 'fun': h0}
]

# eksplicitna ogranicenja
bounds = [(-3, 3), (-3, 3)]

```

```
x0 = [3, 1] # pocetna tocka
result = minimize(f2d.sferna, x0, method='SLSQP', bounds=bounds, constraints=ogranicenja)
print(f'Rezultat: f({result.x}) = {result.fun}')
```

Pokretanjem prethodnoga programa dobivamo sljedeće rješenje:

```
Rezultat: f([0.09160798 0.09160798]) = 0.01678404338119844
```

Osim samog položaja minimuma i vrijednosti funkcije cilja, rezultat koji vraća funkcija `minimize` uključuje i dijagnostičke detalje o procesu optimizacije, kao što je status konvergencije, broj evaluacija funkcije cilja te postignuti uvjet zaustavljanja.