Glava 1

NELINEARNO PROGRAMIRANJE

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

$$\begin{aligned} f_i(x_1, \ldots, x_n) &\geq 0, \quad i = 1, \ldots, k_1, \\ g_i(x_1, \ldots, x_n) &\leq 0, \quad i = 1, \ldots, k_2, \\ h_i(x_1, \ldots, x_n) &= 0, \quad i = 1, \ldots, k_3, \\ k_1 + k_2 + k_3 &= m, \\ x_j &\geq 0, \quad j = 1, \ldots, n. \end{aligned} \tag{1.1}$$

1.1 Bezuslovna optimizacija

1.1.1 Negradijentni metodi

Jednodimenzionalna negradijentna optimizacija

Zadata je ciljna funkcija $Q(x)$ koja zavisi od jednog parametra x. Problem je naći lokalni maksimum (minimum) te funkcije, uz uslov nametnut upravljačkom parametru: $a \leq x \leq b$.

Skeniranje sa konstantnim korakom

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

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

Korak 2. Staviti $Q_m = Q(a)$, i u slučaju maksimuma i minimuma.

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

Korak 4. $X = X + \Delta$

Korak 5. Ako je $X > b$ za izlaz iz algoritma uzeti Q_m i vrednost parametra X_m za koji je optimalna vrednost dostignuta; inače preći na sledeći korak.

Korak 6. Izračunati $Q_1 = Q(X)$. Staviti $Q_m = Q_1$, $X_m = X$, ako je $Q_1 > Q_m$, u slučaju maksimuma, tj. ako je $Q_1 < Q_m$ u slučaju minimuma. Preći na korak 4.

Implementacija u programskom paketu *Mathematica* :

```
skk[Q_, pr_, a_, b_, del_] :=
  Block[{q0, qm, izb, lista = {}, x, xm},
    izb = Input["1 za minimum, 2 za maksimum"];
    xm = x = a;
    qm = N[Q /. pr[[1]] -> x];
    AppendTo[lista, {xm, qm}];
    Do[
      q0 = Q /. pr[[1]] -> x;
      If[(izb == 1 && N[q0] < N[qm]) ||
            (izb == 2 && N[q0] > N[qm]),
        xm = x;
        qm = N[q0];
        AppendTo[lista, {xm, qm}]
        ],
      {x, a, b, del}
      ];
    Return[{xm, qm, lista}]]
```

Skeniranje sa promenljivim korakom

Za prvo skeniranje se uzima relativno veliki konstantni korak $\Delta^{(1)}$ i grubo se likalizuje tačka ekstremuma $x_m^{(1)}$. Zatim se izdvaja podoblast $x_m^{(1)} \pm \Delta^{(1)}$ i u njoj se izvrši skeniranje sa manjim korakom $\Delta^{(2)}$. Korak se smanjuje, sve dok se ne postigne zadata tačnost Δ_{min}.

Implementacija u programskom paketu *Mathematica* :

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

```
  ];
{xm, qm, lista}
]
```

Zadatak 1. Metodom skeniranja sa konstantnim korakom naći minimum i maksimum funkcije $f(x) = \sin(x + 3)$ na intervalu $[-2, 2]$.

Zadatak 2. Metodom skeniranja sa promenljivim korakom naći minimum i maksimum funkcije $f(x) = x^2 - 5x + 8$ na intervalu $[-4, 4]$, sa početnim korakom 0.1 i tačnošću 0.02.

Zadatak 3. Koristeći metodu skeniranja sa promenljivim korakom izračunati lokalni minimum funkcije $f(x) = x(x^2 - 1)\sin 3x$, na intervalu $[0, 1]$. Prestati sa izračunavanjima kada se postigne tačnost 10^{-8}.

Jednodimenzionalni simpleks metod

Ovaj metod se može koristiti kako u slučaju kada su zadate granice upravljačkog parametra $a \leq x \leq b$, tako i kada jedna od granica (ili obe) nije zadata. Ovaj metod se koristi i kad ciljna funkcija zavisi od više upravljačkih parametara.

Varijanta I

Pretraživanje počinje sa zadatom veličinom početnog koraka, koji se neprekidno smanjuje i menja smer. Ova varijanta se koristi kada je zadata barem jedna od granica pretraživanja. Skeniranje započinje sa relativno velikim korakom $\Delta^{(0)}$, a nastavlja se sve dok se ne dobije neuspešan rezultat. U tom slučaju se menja smer skeniranja i uzima se korak manje dužine $\Delta^{(1)}$. Preporučuje se da taj korak bude $\Delta^{(k+1)} = -\Delta^{(k)}/4$. Rezultat je neuspešan ako, recimo tražimo minimum funkcije cilja a u nekom od koraka dobijemo veću vrednost funkcije cilja nego što smo je imali u prethodnom koraku, tj. ne menjamo smer skeniranja sve dok funkcija, u tom smeru opada, ukoliko tražimo minimum. U slučaju maksimuma imamo obrnutu situaciju, ne menjamo smer skeniranja sve dok funkcija raste. Iterativni proces se nastavlja sve dok se ne dostigne željena tačnost $\Delta^{(k)} \leq \Delta_{min}$.

Implementacija u programskom paketu *Mathematica* :

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

```
        xm = N[xt]; qm = N[q1];
        AppendTo[lista, {xm, qm}],
        del = N[-del/4];
        ];
      ];
    it += 1;
    xt = N[xt + del]
    ];
  Return[{N[xm], N[qm], lista}];
  ]
```

Varijanta II

Pretraživanje kod ove varijante započinje od izabrane početne tačke x_0, sa zadatom veličinom početnog koraka koji se uvećava sve dok se ne dođe u oblast ekstremne tačke. Tada počinjemo sa smanjivanjem koraka.

Ova varijanta simpleksa je pogodna kada oblast pretraživanja nije zadata.

Implementacija u programskom paketu *Mathematica* :

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

Zadatak 1. Jednodimenzionalnim simpleks metodom naći minimum i maksimum funkcije $\sin(x-1) + x^2\sqrt{|x-1|}$ na intervalu $[-4, 4]$. Neka početni korak skeniranja bude 0.1. Izračunavanje prekinuti kada se dostigne tačnost od 0.02.

Zadatak 2. Jednodimenzionalnim simpleks metodom naći minimum i maksimum funkcije x^2 na intervalu $[-1, 1]$. Neka početni korak skeniranja bude 0.1. Izračunavanje prekinuti kada se dostigne tačnost od 0.01.

Zadatak 3. Jednodimenzionalnim simpleks metodom naći minimum i maksimum funkcije $\sin y$ na intervalu $[-\pi, \pi]$. Neka početni korak skeniranja bude 0.1. Izračunavanje prekinuti kada se dostigne tačnost od 0.001.

Metod dihotomije

Ovaj metod predstavlja posebno skeniranje unutar intervala $[a, b]$, pri čemu se u svakoj iteraciji oblast smanjuje dva ili četiri puta. Najpre se izračunava vrednost funkcije u pet tačaka, koristeći korak skeniranja $\Delta = (b - a)/4$. Od pet dobijenih vrednosti ciljne funkcije u datim tačkama bira se najbolja i označava sa qm, a odgovarajuća vrednost sa xm.

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

```
      del = (d - c)/4;
      x2 = c + del; x3 = d - del; x1 = c + 2*del;
      qa = N[q /. pr[[1]] -> c]; qb = N[q /. pr[[1]] -> d];
      q1 = N[q /. pr[[1]] -> x1]; q2 = N[q /. pr[[1]] -> x2];
      q3 = N[q /. pr[[1]] -> x3];];
    If[izb == 2, qm = Max[qa, qb, q1, q2, q3],
      qm = Min[qa, qb, q1, q2, q3]]];
  Return[{N[xmax], N[qmax], Lista}]]
```

Metod zlatnog preseka

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

```
    Return[{xm, qm}]]
```

Metod Davies - Swann - Campey (DSC)

```
Dsk[q_, pr_List, x01_, del0_, delmin_] :=
  Block[{x0 = x01, q0, q1, q2, q3, x1, x2, x3, del1, dx = 10, xkmax, qmax, xm,
      xa, xb, qm, qa, qb, d = del0, it = 0, izb, Lista = {}},
    izb = Input["1 za minimum, 2 za maksimum "];
    (*Korak 15*)
    While[Abs[dx] >= delmin && it < 100,
      (*Korak 2*)
      q0 = N[q /. pr[[1]] -> x0];
      Lista = Append[Lista, {x0, q0}];
      (*Korak 3*)
      x1 = x0 + d; q1 = N[q /. pr[[1]] -> x1];
      (*Korak 5*)
      If[(izb == 2 && q0 > q1) || (izb == 1 && q0 < q1),
        d = -d; x2 = x0 + d; q2 = N[q /. pr[[1]] -> x2];
        (*Korak 6*)
        If[(izb == 2 && q0 > q2) || (izb == 1 && q0 < q2),
          xa = x2; xm = x0; xb = x1,
          q1 = q2; x1 = x2
          ];
        ];
      (*Korak 4*)
      If[(izb == 2 && q1 > q0) || (izb == 1 && q1 < q0),
        del1 = 2*d;
        x2 = x1 + del1; q2 = N[q /. pr[[1]] -> x2];
        (*Koraci 7, 8, 9*)
        While[((izb == 2 && q2 > q1) || (izb == 1 && q2 < q1)) && (it < 100),
          x0 = x1; q0 = q1;
          x1 = x2; q1 = q2;
          d = del1; del1 = 2*d;
          x2 = x1 + del1; q2 = N[q /. pr[[1]] -> x2];
          it += 1
          ];
        (*Korak 10*)
        x3 = x2 - del1/2; q3 = N[q /. pr[[1]] -> x3];
        (*Korak 11*)
        If[(izb == 2 && q3 > q1) || (izb == 1 && q3 < q1),
          x0 = x1; q0 = q1; x1 = x3; q1 = q3,
          x2 = x3; q2 = q3];
        (*Korak 12*)
        dx = d/2;
        xm = x1; xa = x0; xb = x2;
        ];
      (*Korak 13*)
```

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

Jednodimenzionalni Powelov metod

```
Powel[q_, pr_List, x01_, del0_, delmin_] :=
  Block[{x0 = x01, q0, q1, q2, x1, x2, in = 0, xkmax,
      qmax, qxt, xt, xkmin, a, b,
      delt = 10, p, it, d = del0, izb, Lista = {}},
    izb = Input["1 za minimum, 2 za maksimum"];
    (*Korak 2.*)
    q0 = N[q /. pr[[1]] -> x0];
    Lista = Append[Lista, {x0, q0}];
    (*Korak 3.*)
    x1 = x0 + d; q1 = N[q /. pr[[1]] -> x1];
    If[(izb == 2 && q1 <= q0 && in == 0) || (izb == 1 && q1 >= q0 && in == 0),
      d = -d; x1 = x0 + d; q1 = N[q /. pr[[1]] -> x1];
      in = 1
      ];
    If[(izb == 2 && q1 <= q0 && in == 1) || (izb == 1 && q1 >= q0 && in == 1),
      Print["Tacnost= ", d];
      Return[{x0, q0, Lista}]
      ];
    x2 = x1 + 2*d; q2 = N[q /. pr[[1]] -> x2];
    it = 2;
    While[Abs[delt] >= delmin,
      (*Korak 7.*)
      a = (x1^2 - x2^2)*q0 + (x2^2 - x0^2)*q1 + (x0^2 - x1^2)*q2;
      b = (x1 - x2)*q0 + (x2 - x0)*q1 + (x0 - x1)*q2;
      xkmax = a/(2*b); qmax = N[q /. pr[[1]] -> xkmax];
      Lista = Append[Lista, {xkmax, qmax}];
      (*Korak 8.*)
      If[izb == 1, qxt = Min[q0, q1, q2], qxt = Max[q0, q1, q2]];
```

```
      Which[qxt == q0, xt = x0,
            qxt == q1, xt = x1,
            qxt == q21, xt = x2
        ];
      (*Korak 10.*)
      delt = Abs[xkmax - xt];
      If[izb == 2, qxt = Min[q0, q1, q2],
                   qxt = Max[q0, q1, q2]];
      Which[qxt == q0, x0 = xkmax; q0 = qmax,
            qxt == q1, x1 = xkmax; q1 = qmax,
            qxt == q2, x2 = xkmax; q2 = qmax
        ];
      (*Ocuvanje redosleda x0 < x1 < x2*)
      If[x1 < x0,
        p = x0; x0 = x1; x1 = p; p = q0; q0 = q1; q1 = p
        ];
      If[x2 < x0,
        p = x0; x0 = x2; x2 = p; p = q0; q0 = q2; q2 = p
        ];
      If[x2 < x1,
        p = x1; x1 = x2; x2 = p; p = q1; q1 = q2; q2 = p
        ];
      it += 1;
      If[it > 100, Print["Tacnost nije dostignuta "];
        Return[{xkmax, N[q /. pr[[1]] -> xkmax], Lista}]
        ]
      ];
    Print["tacnost= ", delt];
    Return[{xkmax, N[q /. pr[[1]] -> xkmax], Lista}];
    ]
```

DSC-Powelov metod

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

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

```
    it += 1;
    If[it > 100, Print["Tacnost = ", delt];
      Return[{xkmax, N[q /. pr[[1]] -> xkmax], Lista}]
      ]
    ];
  Print["tacnost= ", delt];
  Return[{xkmax, N[q /. pr[[1]] -> xkmax], Lista}];
  ]
```

Metoda parabole

Algoritam:

(1) Specificirati pravilo zaustavljanja, tj. zadati pozitivan broj ε u odnosu na koji se algoritam prekida kada je $|f(x^*) - y(x^*)| \leq \varepsilon$.

(2) Odredite proizvoljne tačke $x_1 < x_2 < x_3$ da budu ispunjeni uslovi

$$f(x_1) > f(x_2) \text{ i } f(x_2) < f(x_3)$$

(3) Rešiti sistem jednačina:

$$\begin{aligned} a + x_1 b + x_1^2 c &= f(x_1) \\ a + x_2 b + x_2^2 c &= f(x_2) \\ a + x_3 b + x_3^2 c &= f(x_3) \end{aligned} \tag{1.2}$$

(4) Izračunati:

$$x^* = -\frac{b}{2c}$$

a zatim i $y(x^*) = a + bx^* + cx^{*2}$ i $f(x^*)$.

(5) Ako je $|f(x^*) - y(x^*)| \leq \varepsilon$ proces se zaustavlja; Ako je $|f(x^*) - y(x^*)| > \varepsilon$ označiti sa x_2 onu od četiri tačke u kojoj funkcija f ima najmanju vrednost, a sa x_1 i x_3 njoj dve susedne tačke (levo i desno od nje redom). Sa ovako određenim novim tačkama vratiti se na korak (3).

Zadatak 1. Koristeći metodu parabole, izračunati lokalni **minimum** i maksimum funkcije: $f(x) = 2x^4 - 3x$ na intervalu $[0, 2]$. Prestati sa iteracijama kada se postigne tačnost od 10^{-5}.

Rešenje:

```
In[1]:= f[x_]:=2x^4-3x In[1]:= y[x_]:=a+b*x+c*x^2 In[2]:= tabela={}
Out[2]= {} In[3]:= {x1,x2,x3}={0,1,2} Out[3]= {0,1,2} In[4]:=
sistem={a+x1*b+x1^2*c==f[x1],
                  a+x2*b+x2^2*c==f[x2],
                  a+x3*b+x3^2*c==f[x3]}
```

```
In[5]:= res=Solve[sistem] Out[5]= {{a->0,b->-15,c->14}} In[6]:=
xt=-b/(2c) In[7]:= xt=xt/.res[[1]] Out[7]= 15/28 In[8]:= N[xt]
Out[9]= 0.535714 In[10]:= greska[x_]:=Abs[f[x]-y[x]] In[11]:=
gr=greska[xt]/.res[[1]] Out[11]= 791505/307328 In[12]:= N[gr]
Out[12]= 2.57544 In[13]:= gr<10^-7 Out[13]= False In[14]:=
AppendTo[tabela,{N[x1],N[x2],N[x3],N[xt],N[gr]}] Out[14]=
{{0.,1.,2.,0.535714,2.57544}} In[15]:=
lista1={{x1,f[x1]},{x2,f[x2]},{x3,f[x3]},{xt,f[xt]}} In[16]:=
lista2=Map[N,lista1,{2}] Out[16]=
{{0.,0.},{1.,-1.},{2.,26.},{0.535714,-1.44242}} In[17]:=
x1=lista1[[2,1]] Out[17]= 1 In[18]:= x2=lista1[[4,1]] Out[18]= 15/28
In[19]:= x3=lista1[[3,1]] Out[19]= 2
```

Iteracija	x_1	x_2	x_3	$\tilde{x}$	$\lvert f(\tilde{x}) - y(\tilde{x})\rvert$
1	0	1	2	0.53571	2.57544
2	0	0.53571	1	0.63716	0.10192
3	0.53571	0.63716	1	0.69358	0.01564
4	0.63716	0.69358	1	0.70903	0.00196
5	0.69358	0.70903	1	0.71680	0.00032
6	0.70903	0.71680	1	0.71932	0.00005
7	0.71680	0.71932	1	0.72045	$7.3 \cdot 10^{-6}$
8	0.71932	0.72045	1	0.72085	$1.1 \cdot 10^{-6}$
9	0.72045	0.72085	1	0.72102	$2.0 \cdot 10^{-7}$
10	0.72085	0.72102	1	0.72108	$2.6 \cdot 10^{-7}$

Zadatak 2. Koristeći metodu parabole, izračunati lokalni minimum i **maksimum** funkcije: $f(x) = x(x^2 - 1)\sin 3x$ na intervalu $[0, 1]$. Prestati sa iteracijama kada se postigne tačnost od 10^{-6}.

Zadatak 3. Koristeći metodu parabole, izračunati lokalni minimum i **maksimum** funkcije: $f(x) = x^6 - 2x^2 + x$ na intervalu $[0, 1]$. Prestati sa iteracijama kada se postigne tačnost od 10^{-6}.

Višedimenzionalna negradijentna optimizacija

Metoda Hooke-Jeeves

Ideja ove metode je u tome da se u svakoj aproksimaciji x^k optimalne tačke x^* odrede smerovi u kojima grafik funkcije ima "brda" i "doline" i zatim se "doline" slede do nove aproksimacije x^{k+1}. Ovo se postiže tako što se iz tačke x^k prave "koraci" d^k duž svake koordinatne ose:

$$x^k \pm d_i^k, \quad i = 1, 2, \ldots, n$$

gde su "koraci":

$$d_1^k = \begin{bmatrix} \delta \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad d_2^k = \begin{bmatrix} 0 \\ \delta \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \cdots, \qquad d_n^k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \delta \end{bmatrix}$$

a δ je neki fiksiran pozitivan broj. Ovde indeks k označava redni broj iteracije, dok i označava

koordinatnu osu duž koje se pravi "korak". Korak je uspešan ako se radi o "dolini" (tj. ako se vrednost funkcije smanjuje), a neuspešan ako smo naleteli na "brdo".

Metodu počinjemo sa nekom proizvoljnom tačkom

$$x_0 = x_B^0$$

koju zovemo "bazična tačka". Kasnije, x_B^1, x_B^2, ..., itd. Bazične tačke aproksimiraju tačku optimuma x^*. Označimo sa t_i^k "tekuću" promenljivu koja ima zadatak da prati aproksimacije tokom računanja. Ovde k znači da se radi o aproksimaciji između $(k-1)$-ve i k-te bazične tačke, dok i označava koordinatnu osu.

Na početku računanja stavljamo:

$$t_0^1 = x_B^0$$

Prvo pretražujemo pozitivnu osu x_1, tj. pravimo korak

$$t_0^1 + d_1^0$$

i računamo $f(t_0^1 + d_1^0)$. Ako je $f(t_0^1 + d_1^0) \geq f(t_0^1)$ ovaj korak nas je odveo do "brda" i ovo pretraživanje proglašavamo neuspešnim. Sada pretražujemo negativnu osu

$$t_0^1 - d_1^0$$

Ako je ponovo $f(t_0^1 - d_1^0) \geq f(t_0^1)$ proglašavamo:

$$t_1^1 = t_0^1$$

i počinjemo sa pretraživanjem duž druge ose. Prvo koračamo u pozitivnom smeru:

$$t_1^1 + d_2^0$$

Pretpostavimo da je $f(t_1^1 - d_2^0) < f(t_1^1)$. U ovom slučaju korak nas je doveo u "dolinu", pretraživanje je bilo uspešno. Označimo

$$t_2^1 = t_1^1 + d_2^0$$

Sada ne pretražujemo negativnu osu, nego iz t_2^1 odmah pretražujemo treću osu. To pretraživanje ide prvo u pozitivnom smeru:

$$t_2^1 + d_3^0$$

Ukoliko je ovaj korak uspešan, tj. ako je $f(t_2^1 - d_3^0) < f(t_2^1)$ označimo

$$t_3^1 = t_2^1 + d_3^0$$

i nastavimo sa koracima duž četvrte ose, itd.

Treba imati u vidu dve stvari:

1. Ako je korak duž neke pozitivne koordinatne ose uspešan, tada se ne pravi korak duž negativnog dela te ose.

2. Treba uvek odrediti svih n tačaka t_1^k, t_2^k, ..., t_n^k između dve bazične tačke $x_B^(k-1)$ i x_B^k.

Tako u prvoj iteraciji, nakon pretraživanja svih n koordinatnih osa, završavamo sa novom bazičnom tačkom

$$x_B^1 = t_n^1$$

Pretraživanje koordinatnih osa sada se može nastaviti iz tačke x_B^1. Međutim, prvo se mora nova bazična tačka pokušati pomaknuti u smeru $x_B^1 - x_B^0$, npr. u tačku

$$t_0^2 = x_B^1 + (x_B^1 - x_B^0)$$

Budući da je $f(x_B^1) < f(x_B^0)$, postoji mogućnost da se "dolina" nastavlja spuštanjem u smeru $(x_B^1 - x_B^0)$. **Prvi uspešan korak iz t_0^2 proglašavamo novom bazičnom tačkom x_B^2 iz koje se ponovo počinju praviti koraci duž koordinatnih osa. Ako su svi koraci iz t_0^2 bezuspešni, tada se vraćamo u bazičnu tačku x_B^1 i iz nje pretražujemo koordinatne ose.** Ako su u nekoj bazičnoj tački x_B^k svi koraci dužine δ bezuspešni, tada se dužina δ smanjuje (recimo na polovinu ili desetinu) i koraci se ponavljaju sa novom manjom dužinom. Ako su ponovo svi koraci bezuspešni, ponovo se dužina koraka smanjuje, sve dok dužina koraka ne postane manja od nekog unapred zadatog broja $\varepsilon > 0$. Kada se ova tačnost postigne, algoritam se završava i poslednja bazična tačka je željna aproksimacija optimalne tačke.

Zadatak 1. Metodom Hooke-Jeeves odrediti lokalni minimum funkcije:

$$f(x) = f(x_1, x_2, x_3) = x_1^4 + x_1^3 - x_1 + x_2^4 - x_2^2 + x_2 + x_3^2 - x_3 + x_1 x_2 x_3$$

Sa iteracijama prekinuti kada se postigne tačnost 10^{-3}.

```
In[1]:=
f[vec_]:=vec[[1]]^4+vec[[1]]^3-vec[[1]]+vec[[2]]^4-vec[[2]]^2+vec[[2]]+
                 vec[[3]]^2-vec[[3]]+vec[[1]]*vec[[2]]*vec[[3]]
In[2]:= xk={0,0,0} Out[2]= {0,0,0} In[3]:= tk=xk Out[3]= {0,0,0}
In[4]:= lista={} Out[4]= {} In[5]:= delta=0.1 Out[5]= 0.1 In[6]:=
AppendTo[lista,Append[xk,delta]] Out[6]= {{0,0,0,0.1}} In[7]:=
dk=Table[If[i==j,delta,0],{i,3},{j,3}] Out[7]=
{{0.1,0,0},{0,0.1,0},{0,0,0.1}} In[8]:=
If[f[tk+dk[[1]]]<f[tk],tk=tk+dk[[1]],If[f[tk-dk[[1]]]<f[tk],tk=tk-dk[[1]]]]
Out[8]= {0.1,0,0} In[9]:=
If[f[tk+dk[[2]]]<f[tk],tk=tk+dk[[2]],If[f[tk-dk[[2]]]<f[tk],tk=tk-dk[[2]]]]
Out[9]= {0.1,-0.1,0} In[10]:=
If[f[tk+dk[[3]]]<f[tk],tk=tk+dk[[3]],If[f[tk-dk[[3]]]<f[tk],tk=tk-dk[[3]]]]
Out[10]= {0.1,-0.1,0.1} In[11]:=
If[xk!=tk,AppendTo[lista,Append[tk,delta]],Print["Promeni
korak!!!"]] Out[11]= {{0,0,0,0.1},{0.1,-0.1,0.1,0.1}} In[12]:=
xk1=tk Out[12]= {0.1,-0.1,0.1} In[13]:= tk1=xk1+(xk1-xk) Out[13]=
{0.2,-0.2,0.2} In[14]:=
If[f[tk1]<f[xk1],Print["dolina"];tk=tk1,Print["brdo"]]

         dolina
Out[14]= {0.2,-0.2,0.2} In[15]:=
If[f[tk+dk[[1]]]<f[tk],Print["uspeh"];tk=tk+dk[[1]],
         If[f[tk-dk[[1]]]<f[tk],Print["uspeh"];tk=tk-dk[[1]],Print["neuspeh"]]]

         uspeh
Out[15]= {0.3,-0.2,0.2} In[16]:=
If[f[tk+dk[[2]]]<f[tk],Print["uspeh"];tk=tk+dk[[2]],
         If[f[tk-dk[[2]]]<f[tk],Print["uspeh"];tk=tk-dk[[2]],Print["neuspeh"]]]

In[17]:= If[f[tk+dk[[3]]]<f[tk],Print["uspeh"];tk=tk+dk[[3]],
         If[f[tk-dk[[3]]]<f[tk],Print["uspeh"];tk=tk-dk[[3]],Print["neuspeh"]]]
In[18]:= xk=tk Out[18]= {0.3,-0.2,0.2} In[19]:=
AppendTo[lista,Append[xk,delta]] Out[19]=
```

```
{{0,0,0,0.1},{0.1,-0.1,0.1,0.1},{0.3,-0.2,0.2,0.1}}

Krajnji rezultat: {0,         0,         0,         0.1}, {0.1,
-0.1,      0.1,      0.1}, {0.3,      -0.2,      0.2,      0.1},
{0.4,      -0.3,      0.3,      0.1}, {0.5,      -0.5,      0.4,
0.1}, {0.5,      -0.6,      0.5,      0.1}, {0.5,      -0.8,
0.6,      0.1}, {0.5,      -0.9,      0.7,      0.1}, {0.6,
-0.9,      0.8,      0.1}, {0.55,     -0.95,     0.75,     0.05},
{0.575,    -0.95,     0.775,    0.025}, {0.575,    -0.95,     0.775,
0.0125}, {0.56875,  -0.9375,   0.76875,  0.00625}, {0.571875,
-0.940625, 0.76875,  0.003125}, {0.570313, -0.939063, 0.767188,
0.0015625}, {0.570703, -0.939453, 0.767578, 0.00039063}, {0.570703,
-0.939453, 0.767969, 0.00039063}}
```

Zadatak 2. Metodom Hooke-Jeeves odrediti lokalni minimum funkcije:

$$F(x) = \max\{|4x_1^3 + 3x_1^2 + x_2x_3 - 1|, |4x_2^3 - 2x_2 + x_1x_3 + 1|, |2x_3 + x_1x_2 - 1|\}$$

Prestati sa iteracijama kada se postigne tačnost 10^{-4}

```
In[61]:= f[x_]:=Max[Abs[4x[[1]]^3+3x[[1]]^2+x[[2]]*x[[3]]-1],
                    Abs[4x[[2]]^3-2x[[2]]+x[[1]]*x[[3]]+1],
                    Abs[2x[[3]]+x[[1]]*x[[2]]-1]]
In[62]:= xk={0.1,0.1,0.1} Out[62]= {0.1,0.1,0.1} In[63]:= tk=xk
Out[63]= {0.1,0.1,0.1} In[64]:= lista={} Out[64]= {} In[170]:=
delta=1.9*10^-4 Out[170]= 0.00019 In[66]:=
AppendTo[lista,Join[xk,{f[xk],delta}]] Out[66]=
{{0.1,0.1,0.1,0.956,0.1}} In[171]:=
dk=Table[If[i\[Equal]j,delta,0],{i,3},{j,3}] Out[171]=
{{0.00019,0,0},{0,0.00019,0},{0,0,0.00019}} In[193]:=
If[f[tk+dk[[1]]]<f[tk],tk=tk+dk[[1]],If[f[tk-dk[[1]]]<f[tk],tk=tk-dk[[1]]]]
In[194]:=
If[f[tk+dk[[2]]]<f[tk],tk=tk+dk[[2]],If[f[tk-dk[[2]]]<f[tk],tk=tk-dk[[2]]]]
In[195]:=
If[f[tk+dk[[3]]]<f[tk],tk=tk+dk[[3]],If[f[tk-dk[[3]]]<f[tk],tk=tk-dk[[3]]]]
In[196]:=
If[xk!=tk,AppendTo[lista,Join[tk,{f[tk],delta}]],Print["Promeni
korak!!!"]]

          Promeni korak!!!
In[186]:= xk1=tk Out[186]= {0.325,0.41899,0.17538} In[187]:=
tk1=xk1+(xk1-xk) Out[187]= {0.32481,0.4188,0.17538} In[188]:=
If[f[tk1]<f[xk1],Print["dolina"];tk=tk1,Print["brdo"]]

          dolina
Out[188]= {0.32481,0.4188,0.17538} In[189]:=
If[f[tk+dk[[1]]]<f[tk],Print["uspeh"];tk=tk+dk[[1]],
          If[f[tk-dk[[1]]]<f[tk],Print["uspeh"];tk=tk-dk[[1]],Print["neuspeh"]]]
In[190]:= If[f[tk+dk[[2]]]<f[tk],Print["uspeh"];tk=tk+dk[[2]],
```

```
          If[f[tk-dk[[2]]]<f[tk],Print["uspeh"];tk=tk-dk[[2]],Print["neuspeh"]]]
In[148]:= If[f[tk+dk[[3]]]<f[tk],Print["uspeh"];tk=tk+dk[[3]],
          If[f[tk-dk[[3]]]<f[tk],Print["uspeh"];tk=tk-dk[[3]],Print["neuspeh"]]]
In[191]:= xk=tk Out[191]= {0.32481,0.41899,0.17538} In[192]:=
AppendTo[lista,Join[xk,{f[xk],delta}]] Out[192]= {{0.1,     0.1,
0.1,     0.956,    0.1},
           {0.2,     0.2,     0.2,     0.808,    0.1},
           {0.3,     0.4,     0.3,     0.546,    0.1},
           {0.3,     0.4,     0.2,     0.542,    0.1},
           {0.4,     0.4,     0.2,     0.536,    0.1},
           {0.35,    0.4,     0.2,     0.526,    0.05},
           {0.325,   0.4,     0.175,   0.52,     0.025},
           {0.35,    0.4,     0.175,   0.51725,  0.025},
           {0.3375,  0.4125,  0.175,   0.51482,  0.0125},
           {0.325,   0.425,   0.175,   0.513937, 0.0125},
           {0.325,   0.4188,  0.175,   0.51389,  0.0062},
           {0.32519, 0.41899, 0.17519, 0.513369, 0.00019},
           {0.32519, 0.41918, 0.17538, 0.513291, 0.00019},
           {0.325,   0.41899, 0.17538, 0.513238, 0.00019},
           {0.32481, 0.41899, 0.17538, 0.513204, 0.00019}}
```

Powellova metoda

Ova metoda se može koristiti za izračunavanje lokalnog minimuma proizvoljne funkcije f sa n promenljivih, međutim metodu ćemo objasniti u slučaju kada je f kvadratna funkcija

Definicija 1. Kažemo da je funkcija f kvadratna ako se može predstaviti u obliku:

$$f(x) = \frac{1}{2}(x, Ax) + (b, x) + a \tag{1.3}$$

gde je A simetrična pozitivno definitna matrica.

Definicija 2. Za dva vektora, različita od nula vektora, u i v kažemo da su konjugovani vektori (konjugovani smerovi) matrice A ako važi: $(u, Av) = 0$

Napomena: Svaka simetrična, kvadratna, pozitivno definitna matrica A reda $n \times n$ ima n konjugovanih vektora, i to su npr. sopstveni vektori matrice A.

Optimalna tačka x^* kvadratne funkcije f se može zapisati u obliku:

$$x^* = x^0 + \sum_{i=1}^{n} \theta_i u^i$$

gde je x^0 neka fiksirana polazna tačka, a θ_i, $i = 1, \dots n$ su neki nepoznati brojevi.

Da bi se ovi brojevi odredili, zamenimo x sa x^* u (2.3). Tada je:

$$f(x^*) = \frac{1}{2}\left(x^0 + \sum_{i=1}^{n} \theta_i u^i, A\left[x^0 + \sum_{j=1}^{n} \theta_j u^j\right]\right) + \left(b, x^0 + \sum_{i=1}^{n} \theta_i u_i\right) + a =$$
$$= f(x^0) + \sum_{i=1}^{n} \left[\frac{1}{2}\theta_i^2 (u^i, Au^i) + \theta_i (u^i, Ax^0 + b)\right]$$

Ovo povlači sledeći zaključak:

Minimalnu tačku kvadratne funkcije f sa n promenljivih i sa pozitivno definitnom matricom A možemo odrediti posle najviše n jednodimenzionalnih pretraživanja duž konjugovanih smerova, tj. problem određivanja optimalne tačke x^* svodi se na rešavanje n jednodimenzionalnih problema koji daju željene $\theta_i, i = 1, \ldots, n$.

Treba pronaći brojeve θ_i koji minimiziraju funkcije:

$$\frac{1}{2}\theta_i^2(u^i, Au^i) + \theta_i(u^i, Ax^0 + b)$$

Posle izjednačavanja izvoda sa nulom dobijamo:

$$\theta_i = -\frac{(u^i, Ax^0 + b)}{(u^i, Au^i)}, \quad i = 1, \ldots, n$$

Zadatak 1. Metodom Powell-a odrediti minimum funkcija:
1. $f(x_1, x_2) = x_1^2 + x_1x_2 + \frac{1}{2}x_2^2 + x_1 + 10x_2$.
2. $f(x_1, x_2, x_3) = x_1^4 + x_1^3 - x_1 + x_2^4 - x_2^2 + x_2 + x_3^2 - x_3 + x_1x_2x_3$.

1.1.2 Gradijentna optimizacija

Cauchyeva metoda najstrmijeg pada

Algoritam:

(1) Specificirajte početnu aproksimaciju x^0 i pravilo zaustavljanja, npr. broj $0 < K < 1$. Stavite $k = 0$.

(2) Izračunajte $f(x^k)$ i gradijent $\nabla f(x^k)$ u tački x^k.

(3) Normalizujte gradijent, tj. izračunajte:

$$u^k = \frac{[\nabla f(x^k)]^T}{||[\nabla f(x^k)]^T||}$$

(4) Rešite problem pretraživanja funkcije f iz tačke x^k u smeru $-u^k$, tj. rešite

$$\min_{\sigma>0} f(x^k - \sigma u^k)$$

Označite optimalno rešenje sa σ_k.

(5) Izračunajte novu tačku $x^{k+1} = x^k - \sigma_k u^k$.

(6) Ako je $\sigma_k \leq K\sigma_0$, proces se zaustavlja; metoda daje približno optimalno rešenje $x^* = x^{k+1}$. Ako je $\sigma_k > K\sigma_0$, nastavite sa računom: vratite se u korak (2) sa x^{k+1}.

Zadatak 1. Primeniti Cauchyevu metodu na funkciju:

$$f(x) = x_1^4 + x_1^3 - x_1 + x_2^4 - x_2^2 + x_2 + x_3^2 - x_3 + x_1x_2x_3$$

Iteracije započeti iz koordinatnog početka, prestati kada bude ispunjeno pravilo zaustavljanja sa vrednosti $K = 0,1$. Za normiranje gradijenta koristiti Čebiševljevu normu.

Rešenje:

```
In[1]:= f[x_]:=x[[1]]^4+x[[1]]^3-x[[1]]+x[[2]]^4+-x[[2]]^2+x[[2]]+
               x[[3]]^2-x[[3]]+x[[1]]*x[[2]]*x[[3]]
In[4]:= norma[x_]:=Max[Map[Abs,x]] In[35]:=
grad[f_,lpr_]:={D[f,lpr[[1]]],D[f,lpr[[2]]],D[f,lpr[[3]]]} In[28]:=
x={x1,x2,x3} Out[28]= {x1,x2,x3} In[29]:= xk={0,0,0} Out[29]=
{0,0,0} In[32]:= list={} Out[32]= {} In[56]:= sigma0=smin[[1]]
Out[56]= 0.719893 In[153]:= f[xk] Out[153]= -1.91177 In[154]:=
AppendTo[list,Append[xk,f[xk]]] In[155]:=
grtac=grad[f[x],x]/.{x[[1]]->xk[[1]],x[[2]]->xk[[2]],x[[3]]->xk[[3]]}
Out[155]= {-0.00162158,-0.00347062,-0.0000782755} In[156]:=
uk=(1/norma[grtac])*grtac Out[156]= {-0.467231,-1.,-0.0225538}
In[157]:= fs=f[xk-s*uk] In[159]:= smin=zlatni[fs,{s},-10,10,0.01]
Out[159]= {0.0017307,-1.91176} In[160]:= xk=xk-smin[[1]]*uk
Out[160]= {0.571482,-0.938212,0.7682} In[161]:= smin[[1]] <=
0.01*sigma0 Out[161]= True

{0,         0,        0,         0}, {0.719893, -0.719893, 0.719893,
-1.62252}, {0.529686, -0.902248, 0.729655, -1.90194}, {0.561867,
-0.938905, 0.732349, -1.91051}, {0.569956, -0.933311, 0.748742,
-1.91131}, {0.567066, -0.940642, 0.754836, -1.91157}, {0.568229,
-0.939197, 0.756567, -1.91164}, {0.571092, -0.937224, 0.763898,
-1.91173}, {0.570356, -0.938955, 0.764625, -1.91176}, {0.57032,
-0.939713, 0.766356, -1.91177}, {0.571576, -0.938212, 0.768087,
-1.91176}, {0.570655, -0.939942, 0.768099, -1.91177}, {0.57149,
-0.938212, 0.76819,  -1.91176}, {0.570673, -0.939942, 0.768161,
-1.91177}}
```

Modifikacija Cauchyeve metode

U Cauchyeve-oj metodi, najviše vremena se gubi na jednodimenzionalno pretraživanje funkcije f, da bi se našla optimalna dužina koraka σ_k. U mnogim slučajevima metoda će konvergirati i kada se dužina koraka fiksira, tj. ako je σ_k u svakoj iteraciji isto. Ovako modifikovana metoda je veoma jednostavna i aproksimacije optimalnog rešenja računaju se po pravilu:

$$x^{k+1} = x^k - \sigma[\nabla f(x_k)]^T, \quad , k = 0, 1, 2, \ldots \tag{1.4}$$

Jedan od uslova pod kojima niz $\{x^k\}$ konvergira tački globalnog minimuma funkcije f je da f bude dva puta neprekidno diferencijabilna strogo konveksna funkcija. Korak σ bi trebalo da bude dovoljno mali, npr.

$$0 < \sigma < \frac{2}{\lambda_M}$$

gde je $\lambda_M > 0$ najveća sopstvena vrednost Hesseove matrice funkcije f.

Primer: Koristeći modifikovanu metodu najstrmijeg pada, odrediti globalni minimum funkcije:

$$f(x) = 4x_1^2 - \sin(x_1 + x_2) + 2x_2^2$$

Hesseova matrica:

$$\nabla^2 f(x) = H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

```
In[9]:= hesse[q_,prom_]:=Block[{n,i,j,hes={},dqdx},
           n=Length[prom];
           Do[dqdx={};
              Do[AppendTo[dqdx,D[q,prom[[i]],prom[[j]]]],{j,n}];
              AppendTo[hes,dqdx],
              {i,n}];
           Return[hes]]
In[5]:= f[x_]:=4x[[1]]^2-Sin[x[[1]]+x[[2]]]+2x[[2]]^2 In[11]:=
mat=hesse[f[{x1,x2}],{x1,x2}] Out[11]=
{{8+Sin[x1+x2],Sin[x1+x2]},{Sin[x1+x2],4+Sin[x1+x2]}} In[12]:=
Eigenvalues[mat] In[13]:= norma[x_]:=Max[Map[Abs,x]] In[33]:=
grad[f_,lpr_]:={D[f,lpr[[1]]],D[f,lpr[[2]]]} In[47]:= x={x1,x2}
Out[47]= {x1,x2} In[91]:= xk={0,0} Out[91]= {0,0} In[92]:= list={}
Out[92]= {} In[93]:= sigma=2/13 In[130]:= f[xk]; In[131]:=
grtac=grad[f[x],x]/.{x[[1]]->xk[[1]],x[[2]]->xk[[2]]}; In[132]:=
AppendTo[list,Join[Append[N[xk],N[f[xk]]],N[grtac]]] Out[132]=
{{0.,0.,0.,-1.,-1.},
{0.153846,0.153846,-0.160848,0.277734,-0.33765}, {0.111118,
0.205792,-0.177542,-0.0612603,-0.127033},
{0.120542,0.225336,-0.179349,0.0235618,-0.0394341},
{0.116918,0.231403,-0.179546,-0.00460668,-0.0143363},
{0.117626,0.233608,-0.179568,0.00206178,-0.00451527},
{0.117309,0.234303,-0.17957,-0.000345861,-0.00160671},
{0.117362,0.23455,-0.17957,0.000183316,-0.000514462},
{0.117334,0.234629,-0.179571,-0.0000247419,-0.000180308},
{0.117338,0.234657,-0.179571,0.0000165854,-0.0000584736}} In[129]:=
xk=xk-sigma*grtac;
```

Newtonova metoda

Da bi se Newtonova metoda mogla koristiti, potrebno je da funkcija f bude bar dva puta neprekidno diferencijabilna. U svakoj iteraciji se ne računa samo gradijent $\nabla f(x^k)$ nego i Hesseova matrica $\nabla^2 f(x^k)$. Nova aproksimacija x^{k+1} određuje se po formuli:

$$x^{k+1} = x^k - [\nabla^2 f(x^k)]^{-1} [\nabla f(x^k)]^T, \quad k = 0, 1, 2, \ldots$$

U konkretnoj primeni metode, pogotovu kada je n veliko, izbegava se traženje inverzije Hesseove matrice, već se pribegava sledećem:

$$[\nabla^2 f(x^k)] x^{k+1} = [\nabla^2 f(x^k)] x^k - [\nabla f(x^k)]^T, \quad k = 0, 1, 2, \ldots$$

Primer: Treba odrediti tačku lokalnog minimuma za funkciju

$$f(x) = f(x_1, x_2, x_3) = x_1^4 + x_1^3 - x_1 + x_2^4 - x_2^2 + x_2 + x_3^2 - x_3 + x_1 x_2 x_3$$

```
In[141]:= grad[f_,lpr_]:={D[f,lpr[[1]]],D[f,lpr[[2]]],D[f,lpr[[3]]]}
In[133]:= hesse[q_,prom_,x0_]:=Block[{n,i,j,hes={},dqdx},
          n=Length[prom];
          Do[dqdx={};
          Do[AppendTo[dqdx,D[q,prom[[i]],prom[[j]]]],{j,n}];
          AppendTo[hes,dqdx],
          {i,n}];
          Do[hes=hes/.prom[[i]]\[Rule]x0[[i]],{i,n}];
          Return[hes]]
In[147]:=
f[x_]:=x[[1]]^4+x[[1]]^3-x[[1]]+x[[2]]^4+-x[[2]]^2+x[[2]]+x[[3]]^3-x[[3]]+
                x[[1]]*x[[2]]*x[[3]]
In[148]:= norma[x_]:=Max[Map[Abs,x]] In[149]:= x={x1,x2,x3}
Out[149]= {x1,x2,x3} In[150]:= xk={1,-1,1} Out[150]= {1,-1,1}
In[151]:= list={} Out[151]= {} In[178]:= f[xk]; In[182]:=
AppendTo[list,Append[N[xk],N[f[xk]]]] Out[182]= {{1.,-1.,1.,-1.},
{0.706897,-0.948276,0.775862,-1.97176},
{0.586008,-0.937031,0.72041,-2.05308},
{0.564045,-0.935328,0.713598,-2.05489},
{0.563351,-0.935271,0.713416,-2.05489},
{0.56335,-0.935271,0.713415,-2.05489},
{0.56335,-0.935271,0.713415,-2.05489}} In[180]:=
grtac=grad[f[x],x]/.{x[[1]]->xk[[1]],x[[2]]->xk[[2]],x[[3]]->xk[[3]]};
In[181]:= xk=xk-Inverse[hesse[f[x],x,xk]].grtac;
```

Metode promenljive metrike - DFP metoda

Objedinjuju najbolja svojstva Cauchyeve metode najstrmijeg pada i Newtonove metode. Osnovna ideja ovih metoda je da se započne sa Cauchyevom metodom najstrmijeg pada (što daje velike početne "skokove"):

$$x^{k+1} = x^k - \sigma_k[\nabla f(x^k)]^T$$

i završi se modifikovanom Newtonovom metodom (ovo će garantovati kvadratno završavanje):

$$x^{k+1} = x^k - \sigma_k[\nabla^2 f(x_k)]^{-1}[\nabla f(x^k)]^T$$

Umesto inverzije Hesseove matrice koristiće se neka njena aproksimacija koju ćemo označiti sa H_k. Tada modifikovana Newtonova metoda u svakoj iteraciji izgleda ovako:

$$x^{k+1} = x^k - \sigma_k H_k[\nabla f(x^k)]^T$$

Na samom početku metode možemo specificirati da je $H_0 = I$, tj. za H_0 stavljamo jediničnu matricu. U tom slučaju dobijamo:

$$x^1 = x^0 - \sigma_0[\nabla f(x^0)]^T$$

Ovde pretpostavljamo da σ_k ima uobičajno značenje, tj. σ_k je optimalna dužina koraka u jednodimenzionalnom pretraživanju funkcije f iz tačke x^k u smeru:

$$d^k = -H_k[\nabla f(x^k)]^T$$

Za matricu H_k treba postaviti zahtev da u svakom koraku bude tako upotpunjena da je H_k približno jednaka inverznoj Hesseovoj matrici funkcije f.

Postoju mnogo metoda promenljive metrike i one se razlikuju bas u pravilu upotpunjivanja matrice H.

Algoritam DFP metode:

(i) Izaberite početnu aproksimaciju i početnu pozitivno definitnu matricu H_0. (Popularan izbor za matricu H_0 je jedinična matrica.) Izračunajte gradijent funkcije f u tački x_0, tj.

$$g_0 = [\nabla f(x_0)]^T$$

Specificirajte pravilo zaustavljanja, $K < 1$. Stavite $k = 0$.

(ii) Izračunajte

$$d^k = -\frac{H_k g^k}{||H_k g^k||}.$$

Vektor d_k određuje smer pretraživanja funkcije f iz tačke x_k.

(iii) Izvršite jednodimenzionalno pretraživanje funkcije f u smeru d_k, tj. izračunajte $\sigma_k > 0$ koje minimizira funkciju:

$$F(\sigma) = f(x^k + \sigma d^k)$$

(iv) Izračunajte

$$z^k = \sigma_k d^k$$

i novu aproksimaciju

$$x^{k+1} = x^k + z^k$$

Ako je $\sigma_k \leq K\sigma_0$ proces se zaustavlja; metoda daje približno optimalno rešenje $x* = x^{k+1}$. Ako je $\sigma_k > K\sigma_0$ nastavite sa algoritmom.

(v) Izračunajte redom:

$$\begin{aligned} g^{k+1} &= [\nabla f(x^{k+1})]^T \\ w^k &= g^{k+1} - g^k \\ A_k &= \frac{1}{(z^k, w^k)}(z^k)(z^k)^T \\ B_k &= -\frac{1}{(w^k, H_k w^k)} H_k w^k (w^k)^T H_k \\ H_{k+1} &= H_k + A_k + B_k \end{aligned}$$

Vratite se na korak (ii) sa novim k=k+1.

Zadatak 1. DFP metodom odrediti minimum funkcije

$$f(x_1, x_2) = x_1^2 - 2x_1x_2 + 2x_2^2 - x_1 - 2x_2$$

Zadatak 2. DFP metodom odrediti minimum funkcije

$$f(x) = f(x_1, x_2, x_3) = x_1^4 + x_1^3 - x_1 + x_2^4 - x_2^2 + x_2 + x_3^2 - x_3 + x_1x_2x_3$$

```
In[1]:= hesse[q_,prom_,x0_]:=Block[{n,i,j,hes={},dqdx},
        n=Length[prom];
        Do[dqdx={};
          Do[AppendTo[dqdx,D[q,prom[[i]],prom[[j]]]],{j,n}];
          AppendTo[hes,dqdx],
          {i,n}];
        Do[hes=hes/.prom[[i]]\[Rule]x0[[i]],{i,n}];
        Return[hes]]
In[2]:= grad[f_,lpr_]:={D[f,lpr[[1]]],D[f,lpr[[2]]]} In[3]:=
f[x_]:=x[[1]]^2-2x[[1]]*x[[2]]+2x[[2]]^2-x[[1]]-2x[[2]] In[4]:=
x={x1,x2} Out[4]= {x1,x2} In[6]:= list={} Out[6]= {} In[7]:=
Hk=Table[If[i\[Equal]j,1,0],{i,2},{j,2}] Out[7]= {{1,0},{0,1}}
In[8]:= xk={1,1.5} Out[8]= {1,1.5} In[9]:= grad[f[x],x] Out[9]=
{-1+2 x1-2 x2,-2-2 x1+4 x2} In[10]:=
gk=grad[f[x],x]/.{x[[1]]\[Rule]xk[[1]],x[[2]]\[Rule]xk[[2]]}
Out[10]= {-2.,2.} In[11]:= dk=-Hk.gk Out[11]= {2.,-2.} In[12]:=
fs=f[xk+s*dk] In[13]:= sigma=zlatni[fs,{s},0,100,0.001] In[14]:=
zk=sigma[[1]]*dk In[15]:= xk=xk+zk In[16]:=
AppendTo[list,Append[xk,f[xk]]] In[17]:=
gk1=grad[f[x],x]/.{x[[1]]\[Rule]xk[[1]],x[[2]]\[Rule]xk[[2]]}
In[18]:= wk=gk1-gk In[19]:=
vvT[v_]:=Table[v[[i]]*v[[j]],{i,2},{j,2}] In[20]:=
Ak=(1/zk.wk)*vvT[zk] In[21]:= Bk=-(1/wk.(Hk.wk))*Hk.vvT[wk].Hk
In[22]:= Hk=Hk+Ak+Bk
```

1.2 Razni zadaci

1. Koristeći metodu parabole, izračunati lokalni minimum i maksimum funkcije:

   $$f(x) = x(x^2 - 1)\sin 3x \qquad \text{na intervalu} \quad [1, 3].$$

   Prestati sa iteracijama kada se postigne tačnost od 10^{-6}.

2. Koristeći metodu parabole, izračunati lokalni minimum i maksimum funkcije:

   $$f(x) = x^6 - 2x^2 + x \qquad \text{na intervalu} \quad [0, 1].$$

   Prestati sa iteracijama kada se postigne tačnost od 10^{-6}

3. Metodom Hooke-Jeeves odrediti lokalni minimum funkcije:

   $$f(x_1, x_2, x_3) = (x_1 - 2)^2 + (x_2 - 5)^2 + (x_3 + 2)^4$$

   Za startnu tačku odabrati $(4, -2, 3)$. Prestati sa iteracijama kada se postigne tačnost od 10^{-5} ili kada broj iteracija pređe 7.

4. Koristeći Newtonov-u metodu odrediti tačku lokalnog minimuma za funkciju $f(x_1, x_2, x_3) = x_1^4 + x_1^3 - x_1 + x_2^4 - x_2^2 + x_2 + x_3^2 - x_3 + x_1x_2x_3$. Iteracije započeti iz tačke $(1, -2, 2)$. Prekinuti sa iteracijama kada najveća komponenta gradijenta $[\nabla f(x_k)]^T$ po apsolutnoj vrednosti bude manja od 10^{-5}.

5. Koristeći metodu najstrmijeg pada odrediti minimum funkcije

$$f(x, y) = x^2 + y^2 - 2x + 4y + 1$$

Za startnu tačku uzeti $(2, 1)$. Prekinuti sa iteracijama kada norma gradijenta bude manja od 10^{-5}. Opisati algoritam i svaku iteraciju. Za izračunavanje vrednosti optimalne tačke koristiti programski paket *Mathematica*.

6. Koristeći Newtonov-u metodu odrediti tačku lokalnog minimuma za funkciju $f(x, y) = \ln(1 + x^2) + y^2$. Iteracije započeti iz tačke $(\frac{1}{2}, 2)$. Prekinuti sa iteracijama kada najveća komponenta gradijenta $[\nabla f(x_k)]^T$ po apsolutnoj vrednosti bude manja od 10^{-5}. Za izračunavanje vrednosti optimalne tačke koristiti programski paket *Mathematica*.

7. Napravite 5 iteracija metodom najstrmijeg pada za funkciju:

$$f(x, y, z) = -4x^2 + 4y + 4xy - 11y^2 + 12yz - 4z^2$$

s početnom aproksimacijom $(x_0, y_0, z_0) = (0, 0, 0)$. Kolika je udaljenost pete iteracije od rešenja problema.Za izračunavanja koristiti programski paket *Mathematica*.

8. Napravite 5 iteracija metodom najstrmijeg pada za funkciju:

$$f(x, y) = (x - 1)^2 + (xy - 2)^2$$

s početnom aproksimacijom $(x_0, y_0) = (0, -1)$. Da li se metoda približila tački minimuma. Za izračunavanja koristiti programski paket *Mathematica*.

9. Napraviti 5 iteracija Newton-ovom metodom za funkciju:

$$f(x, y) = x^4 - x^2 + 2xy + y^2$$

s početnom aproksimacijom $(x_0, y_0) = (0, 0)$. Da li se metoda približila tački minimuma. Za izračunavanja koristiti programski paket *Mathematica*.

10. Napraviti 5 iteracija Newton-ovom metodom za funkciju:

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

s početnom aproksimacijom $(x_0, y_0) = (1 - \sqrt{2}, 1 + \sqrt{2})$. Da li se metoda približila tački minimuma. Za izračunavanja koristiti programski paket *Mathematica*.