UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET
ODSJEK ZA RAČUNARSTVO I INFORMATIKU

Nelder-Meadov simpleks algoritam za minimizaciju funkcija više nezavisnih promjenjivih

SEMINARSKI RAD
PRVI CIKLUS STUDIJA

Student:
Kanita Kadušić

Nastavni ansambl:
Red. prof. dr. Željko Jurić, dipl. ing. el.
Šeila Bećirović, MoEE - ing. el.

Sarajevo, februar 2024.

SAŽETAK

Mnogi problemi u praksi se svode na problem minimizacije ili maksimizacije izvjesnih funkcija kriterija, što čini algoritme za optimizaciju izuzetno značajnim. Upravo se jedan od postojećih optimizacijskih algoritama obrađuje u ovom seminarskom radu, a riječ je o Nelder-Meadovom simpleks algoritmu.

Algoritam se razmatra s fokusom na minimizaciji, s obzirom da se problem maksimizacije uvijek može svesti na problem minimizacije. Nadalje, bitno je razlikovati minimizaciju bez ograničenja od minimizacije s ograničenjima, kao i pojam lokalnog minimuma od pojma globalnog minimuma. Zaključno, domen algoritma je minimizacija realnih funkcija više realnih i nezavisnih argumenata bez ograničenja s ciljem pronalaska lokalnih minimuma.

U uvodnom dijelu rada se uvodi pojam simpleksa i teorijski obrađuje algoritam, što prethodi prikazu izvornog kôda primjera implementacije u Julia jeziku, te navođenju i pojašnjenju testnih slučajeva. Slijedi opis raznih problema na koje se algoritam primjenjuje u praksi, nakon čega se rad zaokružuje diskusijom o performansama algoritma.

S obzirom na različite pristupe implementaciji algoritma, rad ima za cilj čitaocu na intuitivan i čitljiv način prikazati osnovnu strukturu algoritma, te moguća i poželjna poboljšanja i pristupe koji bi vodili efikasnijoj implementaciji.

ABSTRACT

Nowadays, many practical problems are reduced to the minimization problem or maximization of certain criterion functions, which makes optimization algorithms extremely important. This paper primarily deals with the Nelder-Mead simplex algorithm, one of the existing optimization algorithms.

The focus of the algorithm is on a minimization, given that a maximization problem can always be reduced to a minimization problem. Furthermore, it is important to distinguish unconstrained minimization from constrained minimization, as well as the concept of local minimum from the concept of global minimum. In conclusion, the focus of the algorithm is the minimization of real functions of multiple real and independent arguments without restrictions to find local minima.

In the introductory part of the paper, the term simplex is introduced and the algorithm is theoretically treated, which precedes the presentation of the source code of the implementation example in the Julia language, and the listing and clarification of test cases. Additionally, a description of various problems to which the algorithm is applied in practice is presented. Finally, the paper concludes with a discussion of the algorithm's performance.

Concerning different approaches to the implementation of the Nelder-Mead simplex algorithm, the paper aims to present the basic structure of the algorithm to the reader in an intuitive and readable way, as well as possible and desirable improvements and approaches that would lead to more efficient implementation.

SADRŽAJ

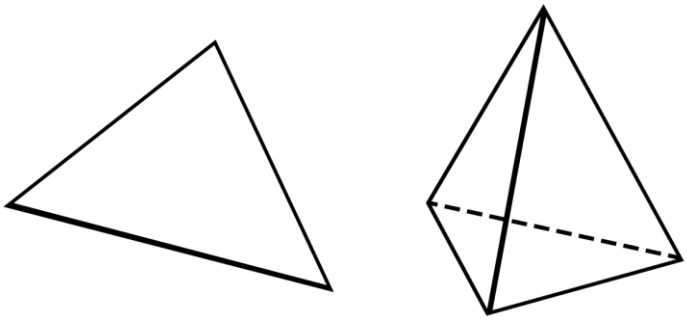
TEORIJSKI UVOD

Nelder-Meadov algoritam, poznat i pod nazivima **simpleks spust** i **ameba algoritam**, važi za jedan od najuspješnijih metoda direktnog pretraživanja u praksi [1]. Generalno, tehnike za višedimenzionalnu minimizaciju se mogu podijeliti u tri skupine: metodi direktnog pretraživanja, metodi pretraživanja po pravcima, te metodi regiona od povjerenja. Metodi direktnog pretraživanja upoređuju vrijednosti funkcije u više različitih tačaka, nakon čega, ovisno od rezultata poređenja, procjenjuju gdje bi se mogla nalaziti sljedeća tačka u kojoj bi trebalo nastaviti ispitivanje [2].

Algoritam ne zahtijeva nikakve dodatne pretpostavke o funkciji koju minimizira. Konkretno, informacije o izvodima funkcije nisu potrebne.

Simpleks

Nelder-Meadov algoritam koristi geometrijski koncept poznat pod nazivom **simpleks**, odnosno n-dimenzionalni objekat sa $n + 1$ vrhova. Na primjer, simpleks za dvodimenzionalnu funkciju predstavlja trougao, dok za trodimenzionalnu funkciju čini tetraedar.



Slika 1: Prikaz simpleksa za dvodimenzionalnu (lijevo) i trodimenzionalnu (desno) funkciju

Bitno je razmotriti pitanje **inicijalnog simpleksa**, kojem se može pristupiti na dva načina, ovisno od toga da li postoji ulazni podatak koji predstavlja tačku za koju se vjeruje da se nalazi u blizini minimuma. Ukoliko takav podatak ne postoji, vrhovi simpleksa se obično biraju tako da budu blizu jedan drugom, recimo da se razlikuju svaki od svakog u vrijednosti jedne od koordinata [2]. Ako se simpleks modelira kao matrica dimenzija $(n + 1) \times n$, dakle čiji redovi predstavljaju vrhove simpleksa u prostoru $\mathbb{R}^n$, inicijalni simpleks može imati sljedeći oblik (d je parametar koji određuje veličinu simpleksa, E je jedinična matrica dimenzija $n \times n$) [3]:

$$d \begin{bmatrix} \mathbf{0} \\ E \end{bmatrix}$$

U suprotnom, simpleks se konstruiše oko date tačke, najčešće tako da jedan vrh simpleksa usvoji njenu vrijednost, dok se ostali raspoređuju u odnosu na njega na sljedeći način ($\boldsymbol{v}_j$ su vektori položaja vrhova simpleksa, $\boldsymbol{e}_j$ su jedinični vektori, h_j su veličine koraka) [1]:

$$\boldsymbol{v}_j = \boldsymbol{v}_1 + h_j \boldsymbol{e}_j;\ j = 2, \dots, n + 1$$

Kao što se može i pretpostaviti, ne postoji takav odabir inicijalnog simpleksa koji rezultira optimalnim radom algoritma za proizvoljnu funkciju. U svakom slučaju, taj izbor je najbolje ostaviti korisniku metode koja implementira Nelder-Meadov algoritam, s pretpostavkom da on posjeduje neko znanje o funkciji koju minimizira, što može biti korisno pri konstrukciji inicijalnog simpleksa.

S obzirom da je za rad algoritma potrebno pronaći položaj **težišta** n-dimenzionalnog simpleksa, slijedi prikaz formule za tu svrhu:

$$\boldsymbol{c} = \frac{1}{n+1}\sum_{j=1}^{n+1} \boldsymbol{v}_j$$

Naravno, podrazumijeva se da je masa ravnomjerno raspoređena unutar cijelog simpleksa ili da je koncentrisana u vrhovima s jednakom masom [4].

Nelder-Meadov algoritam

Algoritam ćemo objasniti razmatrajući funkcije dvije nezavisne promjenjive (simpleks predstavlja trougao), dok analogni koraci vrijede i za funkcije više dimenzionalnosti. Postupak započinje konstrukcijom inicijalnog simpleksa na već opisani način, pri čemu se korisniku algoritma daje mogućnost da ponudi simpleks za koji vjeruje da je pogodno odabran.

U jednoj iteraciji algoritma se izvršavaju sljedeći koraci: sortiranje vrhova, računanje težišta, transformacija simpleksa te ispitivanje konvergencije. Transformacija simpleksa je moguća u vidu refleksije (engl. *reflect*), istezanja (engl. *expand*) i kontrakcije (engl. *contract*), čime se efektivno modifikuje jedan njegov vrh. Ukoliko nije podesno provesti navedene transformacije, vrši se skupljanje (engl. *shrink*) simpleksa, kojim se modifikuju skoro svi njegovi vrhovi. Nakon jedne uspješno završene transformacije, ostale se ne razmatraju.

Kostur iteracije možemo prikazati na sljedeći način:

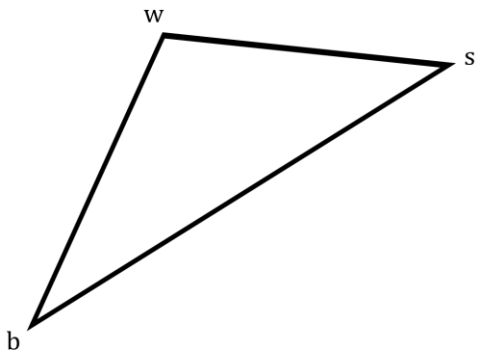
```
sort
calculate centroid
if ...
   reflect
else if ...
   expand
else if ...
   contract outside or shrink
else if ...
   contract inside or shrink
check convergence
```

I Sortiranje

Izračunajmo vrijednosti funkcije u tačkama koje odgovaraju vrhovima trougla (svim vrhovima formiranog simpleksa). Sortirajmo tačke po odgovarajućim vrijednostima funkcije, te ih označimo tako da vrijedi:

$$f(\boldsymbol{b}) < f(\boldsymbol{s}) < f(\boldsymbol{w})$$

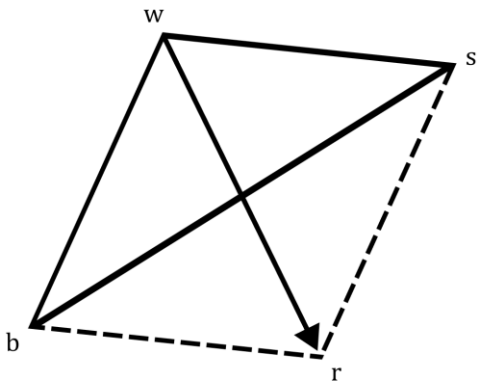
Primijetimo da je, u smislu pronalaska minimuma, tačka $\boldsymbol{b}$ najbolja (u njoj je vrijednost funkcije manja od vrijednosti funkcije u svim ostalim tačkama), a tačka $\boldsymbol{w}$ najgora tačka. Tačku $\boldsymbol{s}$ ćemo smatrati za drugom najgorom tačkom.



Slika 2: Nelder-Meadov algoritam: Sortiranje

II Refleksija

Nastojimo poboljšati postojeći trougao korekcijom najgore tačke $\boldsymbol{w}$, odnosno pronaći takvu tačku $\boldsymbol{r}$, koja će zajedno s tačkama $\boldsymbol{b}$ i $\boldsymbol{s}$ formirati trougao koji je bliži minimumu funkcije. Posmatrajmo vektor s početkom u tački $\boldsymbol{w}$ i krajem u težištu $\boldsymbol{c}$ trougla. Ako adekvatno povećamo njegov intenzitet, tražena tačka $\boldsymbol{r}$ će se nalaziti u tački koja odgovara kraju takvog vektora.



Slika 3: Nelder-Meadov algoritam: Refleksija

Ova transformacija se odvija na sljedeći način [1]:

$$\boldsymbol{r} = \boldsymbol{c} + \alpha(\boldsymbol{c} - \boldsymbol{w})$$

Parametar α je parametar refleksije, a u mnogim implementacijama je $\alpha = 1$.

Tačku $\boldsymbol{r}$ prihvatamo ukoliko je vrijednost funkcije u njoj manja od vrijednosti funkcije u tački $\boldsymbol{s}$, ali nije manja od vrijednosti funkcije u tački $\boldsymbol{b}$. Ispunjenjem navedenih uslova tačka $\boldsymbol{w}$ preuzima vrijednost tačke $\boldsymbol{r}$. U tom slučaju vrijedi:

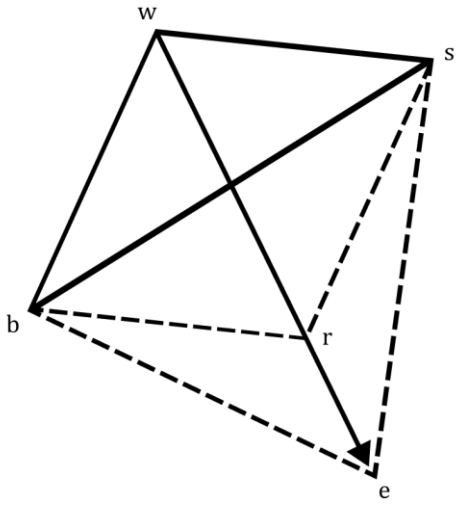
$$f(\boldsymbol{b}) \leq f(\boldsymbol{r}) < f(\boldsymbol{s}) < f(\boldsymbol{w})$$

Odgovarajući dio prikazanog kostura sada poprima oblik:

if $f(r) < f(s)$ **and** $f(r) \geq f(b)$
  $w \leftarrow r$

III Istezanje

Ukoliko se nalazimo na dobrom putu, nastojimo ga iskoristiti, te požuriti prema minimumu tako što ćemo povećati iskorak koji smo napravili refleksijom. U skladu s tim, ako je tačka $\boldsymbol{r}$ bolja ne samo od tačke $\boldsymbol{s}$, već i od tačke $\boldsymbol{b}$ (u smislu minimizacije), pomalo pohlepno ćemo okušati sreću time što ćemo dodatno povećati intenzitet posmatranog vektora, te njegov kraj označiti tačkom $\boldsymbol{e}$:



Slika 4: Nelder-Meadov algoritam: Istezanje

Ova transformacija se odvija na sljedeći način [1]:

$$\boldsymbol{e} = \boldsymbol{c} + \gamma(\boldsymbol{r} - \boldsymbol{c})$$

Parametar γ je parametar istezanja, a u mnogim implementacijama je $\gamma = 2$.

Tačku $\boldsymbol{e}$ prihvatamo ukoliko je vrijednost funkcije u njoj manja od vrijednosti funkcije u tački $\boldsymbol{r}$. Ispunjenjem navedenog uslova tačka $\boldsymbol{w}$ preuzima vrijednost tačke $\boldsymbol{e}$. U suprotnom prihvatamo tačku $\boldsymbol{r}$, te tačka $\boldsymbol{w}$ preuzima njenu vrijednost. U svakom slučaju vrijedi:

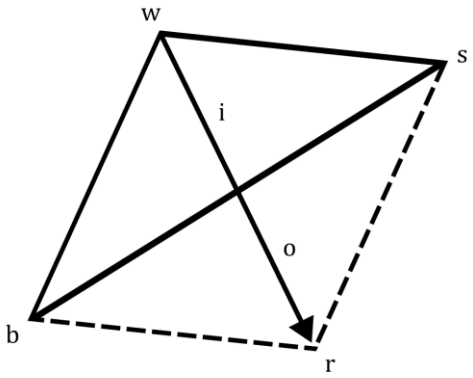
$$f(\boldsymbol{r}) < f(\boldsymbol{b}) < f(\boldsymbol{s}) < f(\boldsymbol{w})$$

Odgovarajući dio prikazanog kostura sada poprima oblik:

else if $f(r) < f(b)$
  if $f(e) < f(r)$
    $w \leftarrow e$
  else
    $w \leftarrow r$

IV Kontrakcija

Kako je skupljanje skupa operacija, poboljšati trougao ćemo još pokušati smanjivanjem iskoraka napravljenim refleksijom. Prvo ispitamo da li je tačka $\boldsymbol{r}$ bolja od najgore tačke $\boldsymbol{w}$. Ako jeste, vršimo vanjsku kontrakciju te novu tačku označimo s $\boldsymbol{o}$. U suprotnom, vršimo unutrašnju kontrakciju te novu tačku označimo s $\boldsymbol{i}$. Inače, kontrakcija je korisna kada prethodne dvije transformacije čine da iskačemo iz susjedstva minimuma.



Slika 5: Nelder-Meadov algoritam: Kontrakcija

Vanjska kontrakcija se odvija na sljedeći način [1]:

$$\boldsymbol{o} = \boldsymbol{c} + \beta(\boldsymbol{r} - \boldsymbol{c})$$

Parametar β je parametar kontrakcije, a u mnogim implementacijama je $\beta = \frac{1}{2}$.

Tačku $\boldsymbol{o}$ prihvatamo ukoliko je vrijednost funkcije u njoj manja ili jednaka vrijednosti funkcije u tački $\boldsymbol{r}$. Ispunjenjem navedenog uslova tačka $\boldsymbol{w}$ preuzima vrijednost tačke $\boldsymbol{o}$. U tom slučaju vrijedi:

$$f(\boldsymbol{b}) < f(\boldsymbol{s}) \leq f(\boldsymbol{o}) \leq f(\boldsymbol{r}) < f(\boldsymbol{w})$$

Unutrašnja kontrakcija se odvija na sljedeći način [1]:

$$\boldsymbol{i} = \boldsymbol{c} + \beta(\boldsymbol{w} - \boldsymbol{c})$$

Tačku $\boldsymbol{i}$ prihvatamo ukoliko je vrijednost funkcije u njoj manja od vrijednosti funkcije u tački $\boldsymbol{w}$. Ispunjenjem navedenog uslova tačka $\boldsymbol{w}$ preuzima vrijednost tačke $\boldsymbol{i}$. U tom slučaju vrijedi:

$$f(\boldsymbol{b}) < f(\boldsymbol{s}) < f(\boldsymbol{i}) < f(\boldsymbol{w}) \leq f(\boldsymbol{r})$$

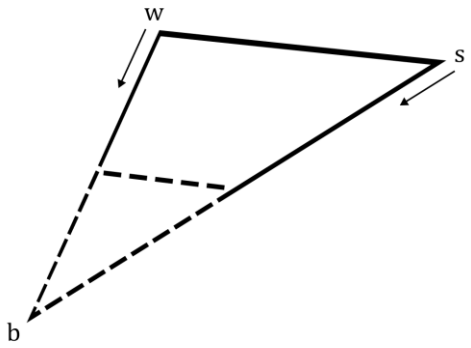
Ukoliko dođe do odbijanja bilo tačke $\boldsymbol{o}$, bilo tačke $\boldsymbol{i}$, nastupa transformacija skupljanja.

Odgovarajući dio prikazanog kostura sada poprima oblik:

```
else if f(r) ≥ f(s) and f(r) < f(w)
    if f(o) ≤ f(r)
        w ← o
    else
        shrink
else if f(r) ≥ f(w)
    if f(i) < f(w)
        w ← i
    else
        shrink
```

V Skupljanje

S obzirom da ni na jedan učinkovitiji način nismo uspjeli transformisati trougao, realiziramo njegovo skupljanje. Ta transformacija uvijek uspijeva, ali po cijenu povećanog broja operacija. Naime, skupljanjem se koriguju i tačka $\boldsymbol{w}$ i tačka $\boldsymbol{s}$, time što one klize ka najboljoj tački $\boldsymbol{b}$ (tačka $\boldsymbol{b}$ ostaje tu gdje jeste). U općem slučaju, vrši se korekcija n tačaka simpleksa, tj. samo tačka u kojoj je vrijednost funkcije najmanja ostaje netaknuta.



Slika 6: Nelder-Meadov algoritam: Skupljanje

Ova transformacija se odvija na sljedeći način [1]:

$$\boldsymbol{v}_j = \boldsymbol{b} + \delta(\boldsymbol{v}_j - \boldsymbol{b});\ j = 2, \dots, n+1$$

Parametar δ je parametar skupljanja, a u mnogim implementacijama je $\delta = \frac{1}{2}$.

Iako je skupljanje skupa operacija, ona je ujedno i rijetka. Štaviše, minimizacija funkcija modeliranih na osnovu problema iz prakse gotovo nikada ne dovodi do ove transformacije [1].

VI Ispitivanje konvergencije

Pokazuje se da je stroga analiza Nelder-Meadovog algoritma težak matematički problem, odnosno odgovor na pitanje konvergencije je dat samo u nekim specijalnim slučajevima.

Za metode direktnog pretraživanja, ispitivanje konvergencije se oslanja na provjeru da li se uglovi između susjednih ivica simpleksa, tokom svake iteracije, nalaze u opsegu od 0 do π, te se uvodi mjera dovoljnog smanjenja vrijednosti funkcije u njegovim vrhovima [5]. U općem slučaju, Nelder-Meadov algoritam ne zadovoljava ni jedno od navedenih svojstava [1].

Ostaje još da razmotrimo pod kojim uvjetima prekinuti rad algoritma. Za tu svrhu se koriste tri tipa testa:

- test konvergencije domena – postaje tačan kada se vrhovi simpleksa razlikuju za manje od definisane tolerancije
- test konvergencije vrijednosti funkcije – postaje tačan kada se vrijednosti funkcije u vrhovima simpleksa razlikuju za manje od definisane tolerancije
- test bez konvergencije – postaje tačan kada se dostigne definisani maksimalni broj iteracija

Algoritam terminira čim barem jedan od testova postane tačan, pri čemu je izlaz iz algoritma najbolja tačka $\boldsymbol{b}$ koja predstavlja lokaciju minimuma [1].

IMPLEMENTACIJA I TESTIRANJE

Implementacija u Julia jeziku

U priloženom kôdu se nalaze četiri funkcije. Ključna od njih je *NelderMead* funkcija, a logika funkcija *getSimplex*, *domainTestPassed* i *valueTestPassed* je s namjerom izdvojena u zasebne cjeline. Ostale upute i pojašnjenja su data u vidu komentara u Julia jeziku.

```
using LinearAlgebra

"""
 - `n` number of vertices
 - `d` size of simplex
 - `return` d * [0 ; E]
"""
function getSimplex(n, d = 1.0)
    [i - 1 == j ? d : 0.0 for i in 1 : n, j in 1 : n - 1]
end

# domain convergence test
# becomes true when the working simplex is sufficiently small i.e. all vertices are close enough
function domainTestPassed(simplex, tolerance)
    all(abs.(simplex[1, 1 : end] .- simplex[2 : end, 1 : end]) .<= tolerance)
end

# function-value convergence test
# becomes true when the values of the function in all vertices of the simplex are close enough
function valueTestPassed(f, simplex, tolerance)
    all(abs.(f(simplex[1, :]) .- f.(eachrow(simplex[2 : end, :]))) .<= tolerance)
end
```

```
"""
 - `f` function with dimension >= 2
 - `simplex` n /vertices/ x (n - 1) /function dimension/
 - `tolerance` non-negative
 - `maxIterations` non-negative
 - `return` [accuracy achieved, minimum]
"""
function NelderMead(f, simplex, tolerance = 1e-10, maxIterations = 500)
    if size(simplex, 1) < 3 || any(length(vertex) != size(simplex, 1) - 1 for vertex in eachrow(simplex))
        throw(ArgumentError("simplex"))
    end

    if tolerance < 0 || maxIterations < 0
        throw(ArgumentError("tolerance or maxIterations"))
    end

    # N - number of vertices of the simplex
    # D - dimension of the function
    N, D = size(simplex, 1), size(simplex, 2)

    # iBest - index of the best performing vertex
    # iWorst - index of the worst performing vertex
    # iSecondWorst - index of the second worst performing vertex
    iBest, iWorst, iSecondWorst = 1, N, N - 1

    # ALPHA - parameter for reflection
    # BETA - parameter for contraction
    # GAMMA - parameter for expansion
    # DELTA - parameter for shrinkage
    ALPHA, BETA, GAMMA, DELTA = 1.0, 0.5, 2.0, 0.5

    while maxIterations > 0
        # sort the vertices by corresponding function values in ascending orderplease
        # f(simplex[1, :]) < f(simplex[2, :]) < ... < f(simplex[N, :])
        for i in 2 : size(simplex, 1)
            j = i

            while j > 1 && f(simplex[j, :]) < f(simplex[j - 1, :])
                simplex[j, :], simplex[j - 1, :] = simplex[j - 1, :], simplex[j, :]
                j -= 1
            end
        end

        if domainTestPassed(simplex, tolerance) || valueTestPassed(f, simplex, tolerance)
            break
        end

        # calculate the centroid of the best side - the one opposite the worst vertex
        # the worst vertex is the last one after sorting
        centroid = sum(simplex[1 : N - 1, :], dims = 1) / (N - 1)
```

```
        # compute the reflection point
        reflection = [centroid[i] + ALPHA * (centroid[i] - simplex[iWorst, i]) for i in 1 : D]

        if f(reflection) < f(simplex[iSecondWorst, :])
            # the reflection point is better than the second worst point
            # check if it is better than the best point
            if f(reflection) < f(simplex[iBest, :])
                # the reflection point is better than all of the vertices
                # compute the expansion point
                expansion = [centroid[i] + GAMMA * (reflection[i] - centroid[i]) for i in 1 : D]

                # accept the expansion point if it is better than the reflection point
                # otherwise accept the reflection point
                if f(expansion) < f(reflection)
                    reflection .= expansion
                end
            end

            simplex[iWorst, :] .= reflection
        else
            # the reflection point is as good as or worse than the second worst point
            # check if it is better than the worst point
            isContracted = false

            if f(reflection) < f(simplex[iWorst, :])
                # the reflection point is better than the worst point
                # compute the outsideContraction point
                contraction = [centroid[i] + BETA * (reflection[i] - centroid[i]) for i in 1 : D]

                # accept the outsideContraction point if it is better or as good as the reflection point
                # otherwise perform a shrink transformation by computing n new vertices
                if f(contraction) <= f(reflection)
                    isContracted = true
                end
            else
                # the reflection point is worse than all of the points
                # compute the insideContraction point
                contraction = [centroid[i] + BETA * (simplex[iWorst, i] - centroid[i]) for i in 1 : D]

                # accept the insideContraction point if it is better than the worst point
                # otherwise perform a shrink transformation by computing n new vertices
                if f(contraction) < f(simplex[iWorst, :])
                    isContracted = true
                end
            end

            # perform contraction or shrinkage
            if isContracted
                simplex[iWorst, :] .= contraction
            else
                for i in 2 : N
                    simplex[i, :] .= simplex[iBest, :] .+ DELTA * (simplex[i, :] .- simplex[iBest, :])
                end
            end
        end

        maxIterations -= 1
    end

    return maxIterations != 0, simplex[iBest, :]
end
```

Testiranje na specifičnim funkcijama

U priloženom kôdu se nalaze dvije funkcije. Funkcija *generateTests* objedinjuje testne slučajeve, a pozivom funkcije *NelderMeadTest* se dobija pregledan ispis rezultata testiranja.

```
using Printf

function generateTests()
    # function name
    # function
    # point of minimum, value at minimum
    return [
        ("Paraboloid",
            x -> (x[1] + 5) ^ 2 + (x[2] + 2) ^ 2,
            [-5.0, -2.0], 0.0),
        ("Sphere",
            x -> sum(x .^ 2),
            [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.0),
        ("Rosenbrock",
            x -> sum(100 * (x[i + 1] - x[i] ^ 2) ^ 2 + (1 - x[i]) ^ 2 for i in 1 : length(x) - 1),
            [1.0, 1.0, 1.0, 1.0, 1.0], 0.0),
        ("Beale",
            x -> (1.5 - x[1] + x[1] * x[2]) ^ 2 + (2.25 - x[1] + x[1] * x[2] ^ 2) ^ 2
               + (2.625 - x[1] + x[1] * x[2] ^ 3) ^ 2,
            [3.0, 0.5], 0.0),
        ("Booth",
            x -> (x[1] + 2 * x[2] - 7) ^ 2 + (2 * x[1] + x[2] - 5) ^ 2,
            [1.0, 3.0], 0.0),
        ("Matyas",
            x -> 0.26 * (x[1] ^ 2 + x[2] ^ 2) - 0.48 * x[1] * x[2],
            [0.0, 0.0], 0.0),
        ("Easom",
            x -> -cos(x[1]) * cos(x[2]) * exp(- (x[1] - pi) ^ 2 - (x[2] - pi) ^ 2),
            [pi, pi], -1.0),
        ("McCormick",
            x -> sin(x[1] + x[2]) + (x[1] - x[2]) ^ 2 - 1.5 * x[1] + 2.5 * x[2] + 1,
            [-0.54719, -1.54719], -1.9133)
    ]
end

function NelderMeadTest()
    accurate = generateTests()

    for i in 1 : length(accurate)
        tested = NelderMead(accurate[i][2], getSimplex(length(accurate[i][3]) + 1), 1e-12)

        @printf("%15s%-10s", "FUNCTION", "")
        println(accurate[i][1])
        @printf("%15s%-10s", "ACCURATE", "")
        println(accurate[i][3], " = ", accurate[i][4])
        @printf("%15s%-10s", "TRUE/FALSE", "")
        println(tested[1])
        @printf("%15s%-10s", "TESTED", "")
        println(tested[2], " = ", accurate[i][2](tested[2]), "\n")
    end
end

NelderMeadTest()
```

Rezultati testiranja

Izabrane testne funkcije se koriste za procjenu karakteristika algoritama optimizacije [6]. Ipak, nisu imale mogućnost u potpunosti doći do izražaja, s obzirom da se cilj ovog seminarskog rada ne ogleda u poboljšanju performansi algoritma.

```
  FUNCTION          Paraboloid
  ACCURATE          [-5.0, -2.0] = 0.0
TRUE/FALSE          true
    TESTED          [-5.000000035118296, -1.9999995315518346] = 2.206769783397391e-13

  FUNCTION          Sphere
  ACCURATE          [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] = 0.0
TRUE/FALSE          true
    TESTED          [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] = 0.0

  FUNCTION          Rosenbrock
  ACCURATE          [1.0, 1.0, 1.0, 1.0, 1.0] = 0.0
TRUE/FALSE          true
    TESTED          [0.999999950008059, 0.9999999122665197, 0.9999998358646316,
                     0.9999996937712807, 0.9999993489446601] = 3.5632597182986863e-13

  FUNCTION          Beale
  ACCURATE          [3.0, 0.5] = 0.0
TRUE/FALSE          true
    TESTED          [2.999998199489788, 0.4999994729893313] = 6.685003270068765e-13

  FUNCTION          Booth
  ACCURATE          [1.0, 3.0] = 0.0
TRUE/FALSE          true
    TESTED          [0.999999507131972, 3.0000003646052837] = 4.4166123144995305e-13

  FUNCTION          Matyas
  ACCURATE          [0.0, 0.0] = 0.0
TRUE/FALSE          true
    TESTED          [0.0, 0.0] = 0.0

  FUNCTION          Easom
  ACCURATE          [π, π] = -1.0
TRUE/FALSE          true
    TESTED          [3.141592773569225, 3.141593228264658] = -0.9999999999994831

  FUNCTION          McCormick
  ACCURATE          [-0.54719, -1.54719] = -1.9133
TRUE/FALSE          true
    TESTED          [-0.5471976666287838, -1.5471971605544237] = -1.9132229549807476
```

Iz priloženog se vidi da su minimumi, s većom ili manjom preciznošću, pronađeni. Nije suvišno napomenuti da se parametrom *tolerance* u funkciji *NelderMead* ne može direktno utjecati na preciznost rezultata. On se koristi u testu konvergencije domena i testu konvergencije vrijednosti funkcije na način koji je već objašnjen.

PRIMJENE ALGORITMA U PRAKSI

Od sredine *XX* stoljeća, kada su se pojavili prvi metodi direktnog pretraživanja, te od 1965. godine, kada su John Nelder i Roger Mead predložili svoju modifikaciju algoritma, nije trebalo proći dugo vremena kako bi stekao popularnost. U to vrijeme, zbog konceptualne jednostavnosti i povoljnih memorijskih zahtjeva, bio je vrlo prikladan za rad u laboratorijama, a nedugo zatim, postao je i standardni dio nekoliko velikih softverskih biblioteka. Njegova reputacija je najviše dobila na značaju 80-ih godina, kada se algoritam prvi put pojavio u „Numeričkim receptima“ (engl. *Numerical Recipes*) i softverskom paketu Matlab [1].

Funkcija koja nam, na osnovu priloženih ulaza, vraća potrebne izlaze, ali za koju ne posjedujemo nikakve strukturne informacije, predstavlja *black-box* funkciju. Neuronska mreža se često navodi kao primjer takvog problema. U kontekstu ove vrste funkcija, razvijaju se *black-box* algoritmi. Upravo je Nelder-Meadov algoritam sastavni dio nekolicine *black-box* optimizacijskih algoritama, koji imaju brojne primjene u praksi, od kojih izdvajamo **planiranje ruta evakuacije** [7].

Primjena se ogleda i u kompjuterskoj viziji (engl. *computer vision*) koja je dio umjetne inteligencije, a bavi se treniranjem algoritama u cilju adekvatne **interpretacije okoline**. Pod tim se podrazumijeva prikupljanje više skupova podataka, nastalih uzorkovanjem iste scene ili objekta u različito vrijeme i iz različitih perspektiva. Neophodna transformacija kojom se raspoloživi podaci svode na jedinstven koordinatni sistem je u vezi upravo s Nelder-Meadovim algoritmom [8]. Dublja analiza ne ulazi u opseg ovog seminarskog rada, pa ćemo spomenuti samo da je zadatak pronaći korelaciju između slika dobijenih iz raspoloživih skupova podataka.

U holografskom snimanju, Nelder-Meadov algoritam se može koristiti za optimizaciju parametara koji utječu na kvalitetu rezultujućih holograma. Ugao snimanja, udaljenost između svjetlosnog izvora i senzora te karakteristike optičkih elemenata, parametri su koji se usklađuju kako bi se postigla optimalna jasnoća i kontrast uz smanjenje šumova. Ova optimizacija pomaže u stvaranju visokokvalitetnih holografskih slika, što je posebno važno u područjima kao što je **medicinska dijagnostika** [9].

U **robotici**, hodanje humanoidnih robota predstavlja složen zadatak jer rukovodi s gomilom parametara. Neki od njih su visina, dužina i frekvencija koraka uz vođenje računa o težištu robota. Istovremeno je potrebno obraditi još mnoštvo senzorima skupljenih informacija iz okoline kako bi se robot adekvatno prilagodio terenu i orijentisao u prostoru. Sve to utječe na stabilnost i brzinu, odnosno opću efikasnost kretanja. Podešavanje parametara je dugotrajan proces sklon greškama. Međutim, pokazalo se da prilagodba temeljena na Nelder-Meadovom algoritmu značajno doprinosi performansama [10].

Medicinska dijagnostička metoda koja se temelji na mjerenju magnetnog polja prouzrokovanog električnom aktivnošću mozga – magnetoencefalografija, za potrebe istraživanja koristi uređaje u obliku kacige. Pošto oni rade na niskim temperaturama, tokom procesa hlađenja može doći do promjene u položaju ili obliku senzora, što stvara potrebu za preciznom **kalibracijom**. Kako bi se postigao taj cilj, parametri senzora se određuju optimizacijskim postupkom temeljenim na Nelder-Meadovom algoritmu. Postupak podrazumijeva maksimizaciju koeficijenta korelacije između predviđenog i zabilježenog magnetnog polja, nakon čega se analitički procjenjuje osjetljivost senzora [11].

U kontekstu dizajna ekološki prihvatljivih zgrada, koristi se za optimizaciju arhitektonskih parametara, poput orijentacije zgrada, odabira materijala i raspodjele otvora, s ciljem postizanja **održivog dizajna**. Regulisanjem tih parametara, algoritam pomaže u stvaranju zgrada koje su ne samo energetski učinkovite, već i održive, koristeći obnovljive izvore energije, čime se smanjuje ukupni ekološki otisak [12].

Algoritam nalazi primjenu i u optimizaciji **rasporeda antena** u bežičnim komunikacijama kako bi se poboljšala pokrivenost signala i kapacitet mreže, te smanjila interferencija između baznih stanica. Prilagođava parametre kao što su visina i ugao nagiba antena kako bi se postigli izloženi ciljevi, što rezultuje efikasnijim korištenjem resursa i poboljšanim performansama komunikacije [13].

Već je rečeno da se mnogi problemi u praksi svode na problem optimizacije. Izloženi primjeri daju uvid u brojnost i raznolikost primjene Nelder-Meadovog algoritma. Očigledno je da se algoritam koristi u svrhu podešavanja i prilagođavanja raznoraznih parametara karakterističnih za posmatranu oblast primjene. To otvara čitav spektar disciplina koje bi mogle iskoristiti prednosti Nelder-Meadovog algoritma, recimo finansije, zrakoplovstvo ili razvoj igara.

ZAKLJUČAK I DISKUSIJA

Nelder-Meadov algoritam je algoritam za minimizaciju realnih funkcija više realnih i nezavisnih argumenata bez ograničenja. Koristi geometrijski koncept poznat pod nazivom simpleks. Osim vrijednosti funkcije u vrhovima simpleksa, ne zahtijeva nikakve pretpostavke o funkciji koju minimizira. Osnovu algoritma čine konstrukcija inicijalnog simpleksa i njegova transformacija, a izlaz je vrh simpleksa s najmanjom vrijednošću funkcije. Zaključno, oblasti primjene Nelder-Meadovog algoritma su brojne i raznovrsne.

Cilj rada se sastojao u prikazu osnovne strukture algoritma, pogodne za razmatranje mogućih i poželjnih poboljšanja i pristupa koji bi vodili efikasnijoj implementaciji. Recimo, jasno je da nije potrebno vrijednosti funkcije u svim vrhovima simpleksa računati u svakoj iteraciji (a ni pri njihovom sortiranju), što posebno dolazi do izražaja ako se ostavi postrani transformacija skupljanja simpleksa, za koju je već rečeno da se skoro nikada ne dešava u praksi. Time jedna iteracija algoritma zahtijeva računanje svega dvije nove tačke i odgovarajućih vrijednosti funkcije, od kojih samo jedna postaje novi vrh simpleksa. Ideja se sada sama nameće: sortiranje treba izvesti koristeći algoritam umetanja (engl. *insertion sort*), a za računanje težišta u tekućoj iteraciji je dovoljno od postojećeg težišta oduzeti najgoru tačku (nakon sortiranja) i dodati tačku prihvaćenu u prethodnoj iteraciji. Dakle, i sortiranje vrhova i računanje težišta moguće je obaviti u linearnom vremenu.

LITERATURA

[1] S. Singer i J. Nelder, „Nelder-Mead algorithm“, *Scholarpedia*, 4(7):2928, 2009. [Online]. Dostupno: http://www.scholarpedia.org/article/Nelder-Mead_algorithm. [Pristupano 10.02.2024.]

[2] Ž. Jurić, „Pristupi minimizaciji funkcija više realnih promjenljivih“, *Numerički algoritmi*, 1. izdanje, Štamparija Fojnica, 2018.

[3] „Nelder-Mead Method“, *Maths from Nothing*. [Online]. Dostupno: https://mathsfromnothing.au/nelder-mead-method/?i=1. [Pristupano 13.02.2024.]

[4] „Centroid“, *Wikipedia*, 05.01.2024. [Online]. Dostupno: https://en.wikipedia.org/wiki/Centroid. [Pristupano 12.02.2024.]

[5] C. Audet i J. E. Dennis, „Analysis of generalized pattern searches“, *SIAM Journal on Optimization*, 2003.

[6] „Test functions for optimization“, *Wikipedia*, 29.12.2023. [Online]. Dostupno: https://en.wikipedia.org/wiki/Test_functions_for_optimization. [Pristupano 17.02.2024.]

[7] S. Takenaga, Y. Ozaki i M. Onishi, „Practical initialization of the Nelder-Mead method for computationally expensive optimization problems“, *SpringerLink*, 2023. [Online]. Dostupno: https://link.springer.com/article/10.1007/s11590-022-01953-y. [Pristupano 13.02.2024.]

[8] M. Holia, „Image registration for recovering affine transformation using Nelder Mead Simplex method for optimization“, *ResearchGate*, 2009. [Online]. Dostupno: https://www.researchgate.net/publication/41890657_Image_registration_for_recovering_affine_transformation_using_Nelder_Mead_Simplex_method_for_optimization. [Pristupano 19.02.2024.]

[9] L. Yang, Z. Cai, R. Wang, S. Feng, Y. Dong, X.Shen, F. Meng i Y. Hu, „Optimization of band gap of photonic crystals fabricated by holographic lithography“, *IOPscience*, 2007. [Online]. Dostupno: https://iopscience.iop.org/article/10.1209/0295-5075/81/14001/meta. [Pristupano 19.02.2024.]

[10] J. D. Weingarten, G. A. D. Lopes, M. Buehler, R. E. Groff i D. E. Koditschek, „Automated gait adaptation for legged robots“, *IEEE Xplore*, 2004. [Online]. Dostupno: https://ieeexplore.ieee.org/abstract/document/1307381. [Pristupano 19.02.2024.]

[11] V. Vivaldi, S. Sommariva i A. Sorrentino, „A simplex method for the calibration of a MEG device“, *Sciendo*, 2019. [Online]. Dostupno: https://sciendo.com/article/10.2478/caim-2019-0005. [Pristupano 19.02.2024.]

[12] Z. Romani, A. Draoui i F. Allard, „Metamodeling the heating and cooling energy needs and simultaneous building envelope optimization for low energy building design in Morocco“, *ScienceDirect*, 2015. [Online]. Dostupno: https://www.sciencedirect.com/science/article/abs/pii/S0378778815003138. [Pristupano 19.02.2024.]

[13] M. H. Wright, „Optimization methods for base station placement in wireless applications“, *IEEE Xplore*, 2002. [Online]. Dostupno: https://ieeexplore.ieee.org/abstract/document/686601. [Pristupano 19.02.2024.]