Efikasna implementacija Nelder–Mead algoritma

Saša Singer

singer@math.hr

web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Pregled predavanja

- Metode direktnog pretraživanja (engl. Direct Search Methods, DSM) za bezuvjetnu optimizaciju.
- Najpoznatija metoda u DSM klasi: Nelder–Mead Search (NMS) ili pretraživanje simpleksima.
- Razne implementacije NMS algoritma, osnovna razlika: testovi zaustavljanja (odn. konvergencije).
- Analiza složenosti jedne iteracije NMS algoritma.
- Efikasna implementacija osnovnog algoritma.
- Posljedica: test zaustavljanja = moguće "usko grlo".
- Efikasni test zaustavljanja: relativni volumen simpleksa (implementacija, diskusija, primjeri).

Problem / Cilj

- Klasični problem bezuvjetne optimizacije:
  Locirati točku minimuma (ili maksimuma) x^* zadane funkcije $f : \mathbb{R}^n \to \mathbb{R}$.
- Rješenje: širok izbor metoda, ovisno o tome koliko informacija o funkciji f imamo na raspolaganju.
- Pretpostavimo da znamo (ili očekujemo) da:
  - f je neprekidna, ali nije glatka, ili
  - f nije ni neprekidna (!),

  barem u nekim točkama iz $\mathbb{R}^n$.
- Te informacije $\Longrightarrow$ fundamentalna ograničenja na izbor metode optimizacije.

Ograničenja na izbor metode

- Metoda optimizacije za nalaženje točke x^*
  - smije koristiti samo funkcijske vrijednosti $f(x)$ u pojedinim točkama $x \in \mathbb{R}^n$,
  - ne smije koristiti (računati ili procijeniti) derivacije od f, jer one ne moraju postojati u pojedinim točkama.
- Dakle, metoda se algoritamski svodi na
  - traži (ili nađi) točku s najmanjom (najvećom) vrijednošću funkcije (tj. samo usporedbe $f(x)$-ova).
- Takve metode obično zovemo
  - "Metode direktnog pretraživanja"

(engl. Direct Search Methods), ili, skraćeno, DSM.

Metode direktnog pretraživanja (DSM)

- Daleko najpoznatija i najčešće korištena DSM u praksi:
  - Nelder–Mead Search (skraćeno NMS), [original 1965., mnoge modifikacije kasnije].

  Osnova metode: pretraživanje simpleksima.
- Još neke metode sličnog "simplex search" tipa:
  - Subspace Simplex (skraćeno SUBPLEX), [Rowan, 1990.].
  - MultiDirectional Search (skraćeno MDS), [Torczon, 1989., Dennis and Torczon, 1991.].
- Zajednički naziv za metode ovog tipa je Simplex DSM.

DSM — nastavak

- Postoje i DSM drugačijeg tipa. Na primjer:
  - Alternating Directions Search (skraćeno ADS).
    Ova metoda radi pretraživanje po koordinatama, paralelno s koordinatnim osima, smjer po smjer.

Pregledni članci na temu DSM

- M. H. Wright: "Direct Search Methods: Once scorned, now respectable", Numerical Analysis 1995, Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis, D. F. Griffiths and G. A. Watson, eds., Addison Wesley Longman, Harlow, UK, 1996, pp. 191–208.

- M. J. D. Powell: "Direct search algortihms for optimization calculations", Acta Numerica 1998, Cambridge University Press, Cambridge, UK, 1998, pp. 287–336.

Nažalost, oba su relativno stara!

Iskustvo u primjeni NMS

- Usprkos popularnosti, postoji široko rasprostranjeno vjerovanje da NMS postaje vrlo neefikasan (tj. spor!) kad n raste, i to već za "umjerene" dimenzije prostora, poput $n \geq 10$.
- Podloga ili "dokaz" za to je, uglavnom, ogromno numeričko iskustvo.
- Nažalost, fali pravo matematičko opravdanje, jer nema dovoljno teorije.
- Posebno, fale rezultati o konvergenciji metode, tj.
  - "Kad konvergira i koliko brzo?"
- Po svemu sudeći, to je vrlo težak problem!

Što se stvarno zna o NMS?

- Ukratko, malo, i to samo za strogo konveksne funkcije.
  - Neki rezultati o konvergenciji postoje samo za male dimenzije (1 i 2) [Lagarias i dr., 1998.].
  - Zna se da NMS ne mora konvergirati prema točki optimuma [kontraprimjer, McKinnon, 1998.].
- Skoro ništa se ne zna o ponašanju NMS za prekidne funkcije (osim, naravno, da ne mora konvergirati).
- Baš takve funkcije se često pojavljuju u eksperimentalnoj matematici.

Problemi s neglatkim funkcijama

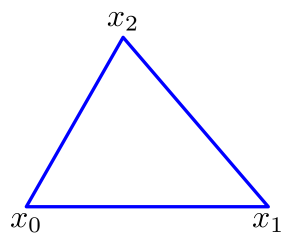
- Standardno se javljuju u praksi, posebno kad je računanje funkcijskih vrijednosti od f uvjetovano ili podložno raznim vrstama grešaka. Tipični primjeri:
  - eksperimentalne greške u mjerenim podacima,
  - greške zaokruživanja u aritmetici računala.
- Klasični numerički problemi s takvim funkcijama:
  - nelinearne aproksimacije izmjerenih podataka ili "data fitting" u nekoj normi (najmanji kvadrati, minimax, i sl.),
  - analiza stabilnosti numeričkih algoritama (na primjer, matričnih faktorizacija, poput LU-a).

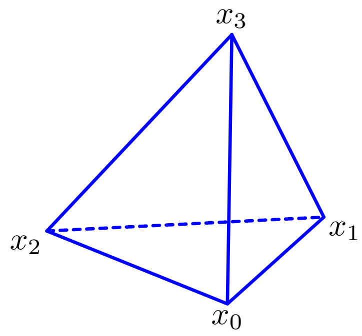
Početak cijele priče

- Za obje vrste problema godinama koristim NMS, a na drugom sam dobro "oprobao" njegovu neefikasnost.
- Problem: analiza stabilnosti LU i simetrične indefinitne faktorizacije obzirom na razne pivotne strategije,
- Posao: maksimizirati nekoliko raznih "mjera" stabilnosti (odnosno greške) s ciljem da se nađu najgori slučajevi (f je prekidna),
- Metode: NMS, SUBPLEX, MDS,
- Iskustvo: Među njima, NMS se pokazao kao (daleko) najsporiji algoritam.
- Terapija: Ostatak predavanja!

Simplex DSM — općenito

- Simpleks $S \subset \mathbb{R}^n$ se definira kao konveksna ljuska od $n+1$ točaka ili vrhova $x_0, \ldots, x_n \in \mathbb{R}^n$.
  Oznaka: $S = S(x_0, \ldots, x_n)$.





- Svi Simplex DSM algoritmi rade neke transformacije "radnog" simpleksa S na bazi funkcijskih vrijednosti u vrhovima simpleksa

$$f_j := f(x_j), \quad j = 0, \ldots, n,$$

  i vraćaju točku $x_{\text{final}} \in \mathbb{R}^n$ koja je izračunata aproksimacija za x^*.

Algoritam Simplex DSM

- Opći zapis algoritma u “kvazi–Pascalu”:

  INIT: konstruiraj inicijalni radni simpleks S_{init};
  repeat (ponavljaj sljedeće korake); {sljedeća iteracija}
    TERM: izračunaj informacije za test zaustavljanja;
    if test zaustavljanja nije ispunjen then
      TRANSF: transformiraj radni simpleks;
  until test zaustavljanja je ispunjen;
  $x_{\mathrm{final}} :=$ najbolji vrh u trenutnom simpleksu S.

- Za problem minimizacije, najbolji vrh je onaj s najmanjom vrijednošću funkcije.

Algoritam INIT

- Konstruira inicijalni simpleks $S_{\text{init}} = S(x_0, \ldots, x_n)$ oko ili blizu neke inicijalne točke x_{init} (obično ulaz), i računa funkcijske vrijednosti f_j u svim vrhovima.
- Najčešći izbor je $x_0 = x_{\text{init}}$ za "restart" algoritma (nastavi tamo gdje si stao).
- Obično je S_{init} pravokutan u x_0, prema koordinatnima osima, tj.

$$x_j := x_0 + h_j e_j, \quad j = 1, \ldots, n,$$

  gdje je h_j korak u smjeru jediničnog vektora $e_j \in \mathbb{R}^n$.
- Katkad je S_{init} pravilan simpleks (svi bridovi imaju istu duljinu).

Algoritam TRANSF (I)

- Algoritam TRANSF u unutarnjoj petlji određuje tip Simplex DSM algoritma.
  U nastavku promatramo samo NMS.
- U svim implementacijama NMS, TRANSF se sastoji iz sljedeća 3 koraka:

1. Odredi indekse h, s, l najgore, druge najgore i najbolje točke (vrha), respektivno, u radnom simpleksu

$$f_h = \max_j f_j, \quad f_s = \max_{j \neq h} f_j, \quad f_l = \min_{j \neq h} f_j.$$

(h = highest, s = second–highest, l = lowest or best).

Algoritam TRANSF (II)

2. Izračunaj centroid (težište) c najbolje strane (to je ona nasuprot najgorem vrhu x_h) u radnom simpleksu

$$c := \frac{1}{n} \sum_{\substack{j=0 \\ j \neq h}}^{n} x_j.$$

- U prvoj iteraciji, za S_{init}, centroid c_{init} moramo ovako izračunati.
- Međutim, iako c ovisi o n vrhova, to ne znači da ga i kasnije treba računati po ovoj formuli. To ovisi o tome koliko se vrhova mijenja iz iteracije u iteraciju.

Algoritam TRANSF (III)

3. Transformiraj simpleks, tj. izračunaj novi radni simpleks S iz prethodnog, nazovimo ga S'.
   - Prvo, probaj zamijeniti najgoru točku x_h boljom točkom x_{new}, koristeći elementarne transformacije

     reflect, expand, contract,

     (zrcaljenje, širenje, smanjenje) obzirom na najbolju stranu. Najbolja strana ostaje strana u novom S.
   - Ako to ne ide, onda primijeni transformaciju

     shrink

     (stiskanje) simpleksa prema najboljem vrhu x_l.

Parametri transformacija

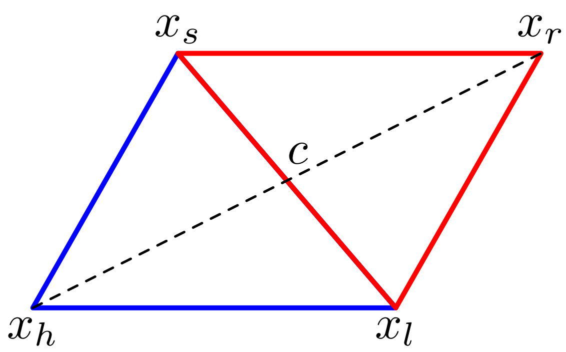
- Transformacije simpleksa $S' \to S$ u NMS određene su s 4 parametra (α, β, γ, δ, alternativno ρ, γ, χ, σ):
  - α za reflect,
  - β za contract,
  - γ za expand,
  - δ za shrink.
- Oni moraju zadovoljavati sljedeća ograničenja:

$$\alpha > 0, \quad 0 < \beta < 1, \quad \gamma > 1, \quad \gamma > \alpha, \quad 0 < \delta < 1.$$

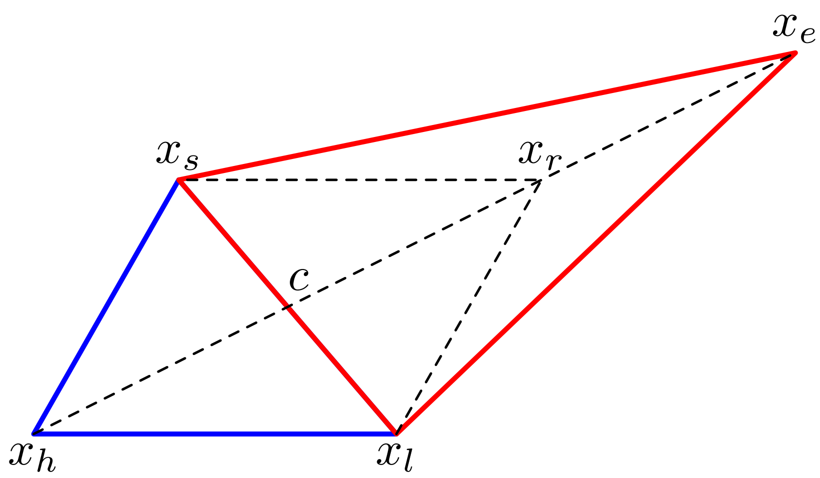
- Standardne vrijednosti, u većini implementacija, su:

$$\alpha = 1, \quad \beta = \frac{1}{2}, \quad \gamma = 2, \quad \delta = \frac{1}{2}.$$

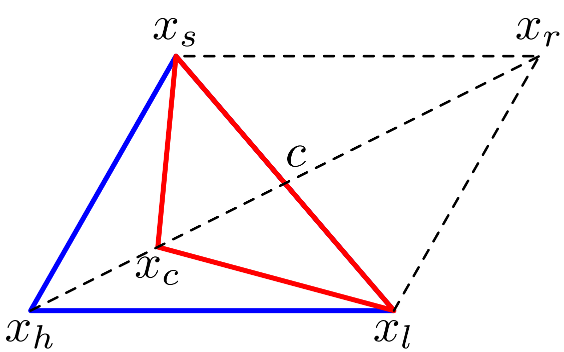
Grafički prikaz transformacija



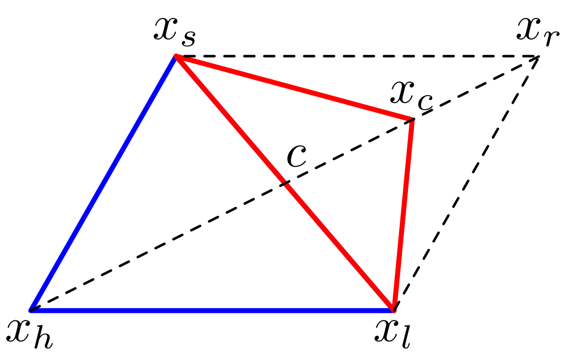
reflect



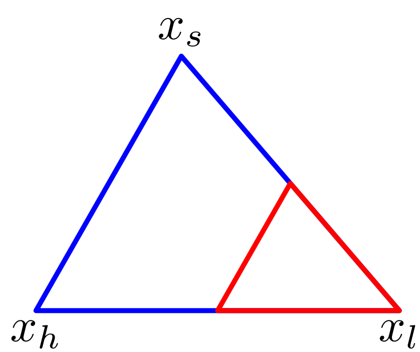
expand



inside contract



outside contract



shrink

Detalji koraka 3 (I)

{ Try to reflect the simplex }

$x_r := c + \alpha(c - x_h); \quad f_r := f(x_r);$

if $f_r < f_s$ then { Accept reflect }

$x_h := x_r; \quad f_h := f_r;$

if $f_r < f_l$ then { Try to expand }

$x_e := c + \gamma(x_r - c); \quad f_e := f(x_e);$

if $f_e < f_l$ then { Accept expand }

$x_h := x_e; \quad f_h := f_e;$

{ Note: this is "greedy expand" }

Detalji koraka 3 (II)

else { $f_r \geq f_s$. Reflect if it helps, and try to contract }

if $f_r < f_h$ then

$x_c := c + \beta(x_r - c);$ { Outside contract }

else

$x_c := c + \beta(x_h - c);$ { Inside contract }

$f_c := f(x_c);$

if $f_c < \min\{f_r, f_h\}$ then { Accept contract }

$x_h := x_c; \quad f_h := f_c;$

Detalji koraka 3 (III)

else { Shrink the simplex towards the best point }

for $j := 0$ to n do

if $j \neq l$ then

$$x_j := x_l + \delta(x_j - x_l); \quad f_j := f(x_j);$$

- Napomena o "greedy expand".

  Na početku, ako i reflect i expand daju bolju točku od najbolje do tada, tj. vrijedi

  $$f_r < f_l \quad \text{i} \quad f_e < f_l,$$

  imamo dvije mogućnosti.

Reflect ili expand?

- greedy expand (pohlepno širenje):
  - prihvaćamo expand, neovisno o odnosu f_r i f_e. Dakle, može se dogoditi da je $f_r < f_e$, tj. x_r je bolja od x_e, a ipak radimo expand, a ne reflect.
  - Ideja: Držimo simpleks što je moguće "većim", da izbjegnemo prerano zaustavljanje iteracija. To ima smisla za neglatke funkcije.
- greedy minimization (pohlepna minimizacija):
  - prihvaćamo bolju od točaka x_r, x_e u novi simpleks, tj. prihvaćamo expand ako i samo ako je $f_e < f_r < f_l$, a inače reflect.
- Za svaki pristup postoje primjeri kad je on bolji.

Algoritam TERM (I)

- Algoritam TERM računa logičku (boolean) vrijednost $term$ koja postaje istinita kad je vrijeme za prekid ili zaustavljanje iteracija.

- Sasvim općenito, $term$ se sastoji od 3 različita dijela

$$term := term_x \text{ or } term_f \text{ or } fail.$$

  Dogovor: Ako neki od ovih dijelova nije prisutan u implementaciji algoritma, smatramo da nije ispunjen, tj. “default” vrijednost tog dijela je laž.

Algoritam TERM (II)

- Značenja pojedinih dijelova su:
  - $term_x$ je "test zaustavljanja konvergencijom u domeni", koji postaje istinit kad radni simpleks S postane dovoljno malen u nekom smislu (neki ili svi vrhovi x_j su dovoljno bliski),
  - $term_f$ je "test zaustavljanja konvergencijom funkcijskih vrijednosti", koji postaje istinit kad (neke ili sve) funkcijske vrijednosti f_j postanu dovoljno bliske u nekom smislu,
  - $fail$ je test da "nema konvergencije na vrijeme".
- Uočiti odmah da $fail$ test mora biti prisutan u svakom numeričkom algoritmu (bez obzira na konvergenciju).

Algoritam TERM (III)

- Postoji mnogo načina za definiciju $term_x$ i $term_f$ testova, i tu je najveća razlika među raznim implementacijama NMS-a (vidjeti kasnije).
- Međutim, neovisno o točnoj definiciji $term$, treba primijetiti dvije jednostavne činjenice:
  - Bez $term_x$ testa, NMS algoritam, očito, ne radi za prekidne funkcije. Dodatno, ako želimo iole razumnu aproksimaciju za x^*, onda $term_x$ test postaje nužan i za neprekidne funkcije.
  - $term_f$ test je, stvarno, samo zaštita (osiguranje) za "skoro ravne" (flat) funkcije.

Efikasnost NMS — općenito

- Da bismo nešto rekli o sveukupnoj efikasnosti NMS-a, trebali bismo neku teoriju konvergencije koja daje procjenu broja iteracija za postizanje neke željene točnosti u testovima zaustavljanja.
- Toga nema, pa se ograničavamo na skromniji cilj:
  - analizu efikasnosti jedne iteracije NMS algoritma.
- To je sasvim dovoljno za otkrivanje potencijalnih "uskih grla" u algoritmu.
- Također, vrlo dobro objašnjava zašto se neke implementacije NMS-a (katkad) izvršavaju "bolno" sporo (problem s početka priče).

Kako mjerimo efikasnost?

- Ukratko: tako da bude praktično! Dakle,

  Definicija

  > Složenost algoritma ALG je broj "flopova" (aritmetičkih "floating point" operacija računala) potrebnih za izvršavanje algoritma za zadani ulaz.
  >
  > Oznaka: $T_{\text{alg}}(\text{input})$.

- Po ovoj definiciji, složenost treba izraziti u terminu ulaznih podataka. Za bilo koji DSM algoritam, to je funkcija f.

Složenost računanja funkcije (I)

- U praksi se ulazna funkcija f uvijek zadaje kao posebni algoritam F, koji računa $f(x)$ za zadani $x \in \mathbb{R}^n$.
- Taj algoritam F ima svoju složenost T_f, koja ovisi o ulaznoj točki x.
- Po definiciji, trebali bismo pisati $T_f = T_f(x)$, kad govorimo o složenosti.
- Naravno, T_f implicitno ovisi o n.
- Flopsi potrebni za računanje $f(x)$ stvarno rade na koordinatama $x(i)$, $i = 1, \ldots, n$, a ne na cijelom x.
- Dakle, $T_f = T_f(x, n)$.

Složenost računanja funkcije (II)

- Da bismo dobili jednostavne i korisne rezultate o složenosti, pretpostavljamo da
  - T_f (uglavnom) ne ovisi o x, a
  - bitno ovisi samo n,

  tako da za složenost vrijedi $T_f = T_f(n)$.
- Ova pretpostavka se lako provjerava i sigurno vrijedi u mnogim praktičnim primjenama.
- U teoriji, možemo ju interpretirati i u probabilističkom smislu, kao prosječnu složenost po svim x.
- Posljedica: rezultate o složenosti izražavamo u terminima n i $T_f(n)$.

Složenost jedne iteracije

- U nastavku koristimo standardne asimptotske oznake (o, O, Θ, Ω, ω) zato da "sakrijemo" nepotrebne detalje.
- Jedna cijela iteracija u Simplex DSM sastoji se od TERM i TRANSF, pa je ukupna složenost jedne iteracije:

$$T_{\text{iter}}(n) = T_{\text{transf}}(n) + T_{\text{term}}(n).$$

- Pošto je TRANSF efektivni (korisni) dio unutarnje petlje, želimo provesti što više vremena u TRANSF radeći koristan posao, a ne gubiti previše vremena u TERM.
- To je motivacija za sljedeću definiciju.

Definicija efkasnosti

Definicija

Efikasnost jedne iteracije Simplex DSM algoritma je

$$E_{\text{iter}}(n) := \frac{T_{\text{transf}}(n)}{T_{\text{iter}}(n)} = 1 - \frac{T_{\text{term}}(n)}{T_{\text{iter}}(n)}.$$

Algoritam je efikasan ako relacija

$$E_{\text{iter}}(n) \approx 1$$

vrijedi za većinu iteracija.

Složenost jedne iteracije NMS

Teorem

Pretpostavimo da je svaki korak u NMS algoritmu TRANSF implementiran maksimalno efikasno. Za svaku iteraciju, osim prve, složenost od TRANSF je

$$T_{\mathrm{transf}}(n) = \begin{cases} \Theta(n) + \Theta(T_f(n)), & \text{bez shrink}, \\ \Theta(n^2) + \Theta(nT_f(n)), & \text{sa shrink}. \end{cases}$$

Složenost ovisi o tome da li iteracija sadrži shrink transformaciju ili ne.

Za prvu iteraciju vrijedi donja relacija, neovisno o transformacijama.

Efikasna implementacija NMS (I)

- Dokaz teorema je konstruktivan, tj. pokazuje kako se efikasno implementiraju sva 3 koraka u TRANSF. Gruba skica:
- Prvi korak (indeksi h, s, l) ide uspoređivanjem u $O(n)$ operacija. Sortiranje (kao u Matlabu) ne treba.
- Osim prvog centroida, svi ostali idu u $O(n)$ operacija

$$c = \begin{cases} c' + \dfrac{1}{n}(x_{h'} - x_h), & \text{bez shrink}, \\ x_{l'} + \delta(c' - x_{l'}) + \dfrac{1}{n}(x_{h'} - x_h), & \text{sa shrink}, \end{cases}$$

ovisno o shrink u prethodnoj transformaciji $S' \to S$.

Napomene uz složenost NMS

- Razumno je očekivati da $f(x)$ ovisi o svih n koordinata od x, tj. svaka koordinata $x(i)$ se koristi bar jednom kao operand za flop.
- Flopovi imaju najviše dva operanda, pa trebamo barem $n/2$ flopova za računanje $f(x)$.
- Dakle, $T_f(n)$ je barem linearan u n, ili $T_f(n) = \Omega(n)$, pa drugi član dominira u teoremu.
- Spore shrink transformacije su rijetke u praksi, pa prvu relaciju iz teorema

$$T_{\mathrm{transf}}(n) = \Theta(T_f(n))$$

treba uzeti za procjenu efikasnosti.

Efikasnost jedne iteracije NMS

- Za efikasnost onda vrijedi

$$E_{\text{iter}}(n) = 1 - \frac{T_{\text{term}}(n)}{T_{\text{term}}(n) + \Theta(T_f(n))}.$$

- Efikasnost jedne iteracije NMS ključno ovisi o $T_{\text{term}}(n)$, tj. koliko je brz test zaustavljanja obzirom na računanje vrijednosti funkcije.

- Dakle, otkrili smo potencijalno "usko grlo" u TERM.

- Napomena: Ovaj rezultat je specifičan za NMS, jer su njegove iteracije vrlo brze, ako nema shrink transformacija.

Zaključak o efikasnosti

- Ako za danu funkciju f vrijedi

$$T_{\mathrm{term}}(n) = \omega(T_f(n))$$

onda test zaustavljanja postaje “usko grlo” i NMS algoritam je neefikasan za taj f.

- Da bismo to izbjegli za sve funkcije f, mora vrijediti

$$T_{\mathrm{term}}(n) = o(n),$$

tj. test zaustavljanja mora biti sublinearan u n.
To je nužan i dovoljan uvjet da NMS algoritam bude efikasan za sve f (ako je sve ostalo efikasno).

Efikasnost testova zaustavljanja (I)

- Ostaje vidjeti koliko je ova opasnost realna u praksi.
- Pogledajmo kako se standardno implementiraju sva 3 dijela $term$ testa: $fail$, $term_f$ i $term_x$.
- $fail$ test samo provjerava broj iteracija ili računanja funkcijskih vrijednosti obzirom za unaprijed zadane maksimalne vrijednosti.

  Složenost je $\Theta(1)$ i on je uvijek efikasan.

Efikasnost testova zaustavljanja (II)

- Postoje dvije vrste $term_f$ testova u praksi.
  - Prvi tip koristi konstantni broj funkcijskih vrijednosti (obično 2 ili 3) za računanje testa. Složenost je $\Theta(1)$, što je efikasno. Primjer: amoeba

    $$term\_f := 2 \cdot \frac{|f_h - f_l|}{|f_h| + |f_l|} \le tol\_f,$$

    gdje je tol_f neka zadana relativna tolerancija.
  - Drugi tip koristi svih $n + 1$ funkcijskih vrijednosti. Složenost je $\Theta(n)$, što može biti neefikasno. Primjer: testovi na bazi standardne devijacije (NAG, IMSL).

Efikasnost testova zaustavljanja (III)

- $term_x$ test je stvarno "usko grlo" u praksi.

  Sve implementacije NMS algoritma koje smo probali mogu se podijeliti u tri grupe obzirom na $term_x$ test.

  1. Ovaj test treba $\Theta(n^2)$ flopova, što je sporo, katkad vrlo sporo. Svi testovi na bazi dijametra radnog simpleksa spadaju u ovu grupu. Tipični primjeri:

Efikasnost testova zaustavljanja (IV)

- fminsearch.m (Matlab)

$$term_x := \max_{j \neq l} \|x_j - x_l\|_\infty \leq tol_x,$$

gdje je tol_x zadana apsolutna tolerancija,

- nmsmax.m (N. J. Higham)

$$term_x := \frac{\max\limits_{j \neq l} \|x_j - x_l\|_1}{\max\{1, \|x_l\|_1\}} \leq tol_x,$$

gdje je tol_x zadana relativna tolerancija.

Efikasnost testova zaustavljanja (V)

2. Ovaj test treba $\Theta(n)$ flopova, što može biti neefikasno, ako je $T_f(n) = \Theta(n)$.

   Jedini primjer kojeg smo našli je

- simplx.f (T. H. Rowan)

$$term_x := \|x_h - x_l\|_2 \leq tol_x \cdot \|x_h^{(0)} - x_l^{(0)}\|_2,$$

gdje su $x_l^{(0)}$ i $x_h^{(0)}$ najbolja i najgora točka u inicijalnom simpleksu S_{init}, a tol_x je zadana relativna tolerancija.

Efikasnost testova zaustavljanja (VI)

3. Nema $term_x$ testa. To je, očito, efikasno, ali radi samo za (barem) neprekidne funkcije.

   Ovoj grupi pripadaju implementacije: IMSL, NAG, amoeba (iz Numerical Recipes).

- Zaključak: Niti jedna od ovih implementacija nije efikasna za sve funkcije f!

  To se posebno odnosi na prekidne funkcije.

- Pitanje: Može li se konstruirati $term_x$ test koji bi bio efikasan za sve funkcije?
- Začudo, odgovor je DA, i to vrlo jednostavno!

Volumen simpleksa

- NMS algorithm radi niz transformacija simpleksa i lako se vidi da se volumeni simpleksa ponašaju vrlo jednostavno u tim transformacijama.
- Volumen simpleksa $S = S(x_0, \ldots, x_n)$ definira se kao

$$V(S) := \frac{1}{n!} \cdot \sqrt{\Gamma(x_1 - x_0, \ldots, x_n - x_0)},$$

gdje Γ označava Grammovu determinantu.

Omjeri volumena u NMS

- Svaku od 5 elementarnih transformacija simpleksa iz koraka 3 možemo napisati u obliku $S := \mathrm{transform}(S')$.
- Za omjer volumena ovih simpleksa vrijedi

$$\frac{V(S)}{V(S')} = \begin{cases} \alpha, & \text{za transform} = \text{reflect}, \\ \beta, & \text{za transform} = \text{inside contract}, \\ \alpha \cdot \beta, & \text{za transform} = \text{outside contract}, \\ \alpha \cdot \gamma, & \text{za transform} = \text{expand}, \\ \delta^n, & \text{za transform} = \text{shrink}. \end{cases}$$

- Ako faktore izračunamo unaprijed, omjer volumena možemo "popraviti" jednim flopom u svakoj iteraciji.

Relativni volumen simpleksa

- Očiti test zaustavljanja relativnim volumenom je

$$term_v := V(S) \leq tol_v \cdot V(S_{\mathrm{init}}),$$

  gdje je tol_v zadana volumna tolerancija.

- Da bismo izbjegli probleme u aritmetici računala (underflow/overflow), ovaj test možemo napisati u "lineariziranom" obliku.
  - Uzmemo n-ti korijen!

Linearizirani relativni volumen

- Neka je $LV(S) := \sqrt[n]{V(S)}$ linearizirani volumen od S. Pripadni "linearizirani" test ima oblik

$$term_x := LV(S) \leq tol_x \cdot LV(S_{\rm init}),$$

  gdje je tol_x zadana relativna tolerancija.

- "Popravci" omjera lineariziranih volumena i dalje trebaju jedan flop po iteraciji, što je uvijek efikasno!
- Početni linearizirani volumen $LV(S_{\rm init})$ se lako računa za sve standardne izbore $S_{\rm init}$.
- Međutim, $LV(S_{\rm init})$ nam uopće ne treba, jer gledamo samo omjere. Dovoljno je uzeti $LV(S_{\rm init}) = 1$.

Modelni problem za testiranje

- Pivotni rast u Gaussovim eliminacijama (LU faktorizaciji) matrice $A \in \mathbb{R}^{m \times m}$, za zadani $m \in \mathbb{N}$,
  - uz neku izabranu pivotnu strategiju P.
- Najgori slučajevi za strategiju P su maksimalne vrijednosti za

$$f(A) = \text{faktor rasta } \rho_m(A) \text{ u LU–P faktorizaciji od } A.$$

Faktor rasta

- Faktor rasta $\rho_m(A)$ (za strategiju P) je

$$\rho_m(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|},$$

  gdje su $a_{ij}^{(k)}$ "međuelementi" generirani u procesu eliminacije.
- f je prekidna u nekim točkama iz $\mathbb{R}^n$, uz $n = m^2$.
- Za sve razumne strategije, T_f vrlo blago ovisi o A, i vrijedi

$$T_f(m) = \Theta(m^3) \quad \text{or} \quad T_f(n) = \Theta(n^{3/2}).$$

Algoritmi

- S1 (slow), nalaženje indeksa h, s, l, sortiranjem, trajanje $T_1(n) = O(n^2) = O(m^4)$,
- F1 (fast), nalaženje h, s, l, direktnim usporedbama, trajanje $T_1(n) = O(n) = O(m^2)$.

- A1 = S1 + Spori centroid + Highamov $term\_x$,
- A2 = F1 + Spori centroid + Highamov $term\_x$,
- A3 = F1 + Brzi centroid + Highamov $term\_x$,
- A4 = F1 + Brzi centroid + Rowanov $term\_x$,
- A5 = F1 + Brzi centroid + Volume $term\_x$.

Prosječno vrijeme (u 10^{-6} s)

- po iteraciji za različite algoritme.

m	A1	A2	A3	A4	A5
2	162	146	138	124	114
3	363	358	312	230	212
4	827	823	668	393	382
5	1684	1669	1297	603	552
6	3163	3132	2350	885	798
7	5535	5460	3978	1228	1103
8	9112	8986	6427	1717	1545

Prosječno vrijeme (u 10^{-6} s)

- po izvrednjavanju funkcije za različite algoritme.

m	A1	A2	A3	A4	A5
2	73	79	74	67	61
3	212	218	190	139	125
4	506	513	423	247	218
5	1121	1130	875	406	364
6	2231	2193	1646	618	555
7	4031	3972	2894	893	801
8	6568	6336	4532	1213	1089

Broj izvrednjavanja funkcija i iteracija

- za $m = 8$ za različite algoritme.

broj	A1	A2	A3	A4	A5
f_{eval}	157407	156107	156091	155359	156091
iter	112201	110073	110057	109759	110057
reflect	81985	80541	80525	80938	80525
expand	1410	1489	1489	1504	1489
contract	28609	27817	27817	27087	27817
shrink	197	226	226	230	226

Pivotni rast za algoritam A5

- za parcijalno, potpuno i “rook” pivotiranje.

m	Parcijalno piv.	Potpuno piv.	“Rook” piv.
2	1.9999999999166	1.9999999998933	1.9999999999213
3	3.9999999998406	1.9999999999269	2.9999735796058
4	7.9961169743494	3.9773044815897	4.4541180620941
5	15.1876649879346	3.9997865225794	5.8048103382428
6	24.3141518961118	3.9998583132353	6.5902048007850
7	39.2122482678208	4.1606104368340	6.7646261129981
8	59.7370345076401	4.2481239632305	7.5704815202052
9	71.9623989093906	4.2966455561396	8.8856503945245
10	106.3947268046369	4.3543040373695	9.1827525408074