

Градијентни метод (Gradient)

- Полазна тачка $\mathbf{x}^0 = (x_1^0, x_2^0, \dots, x_D^0)$
- Нумеричка апроксимација градијента

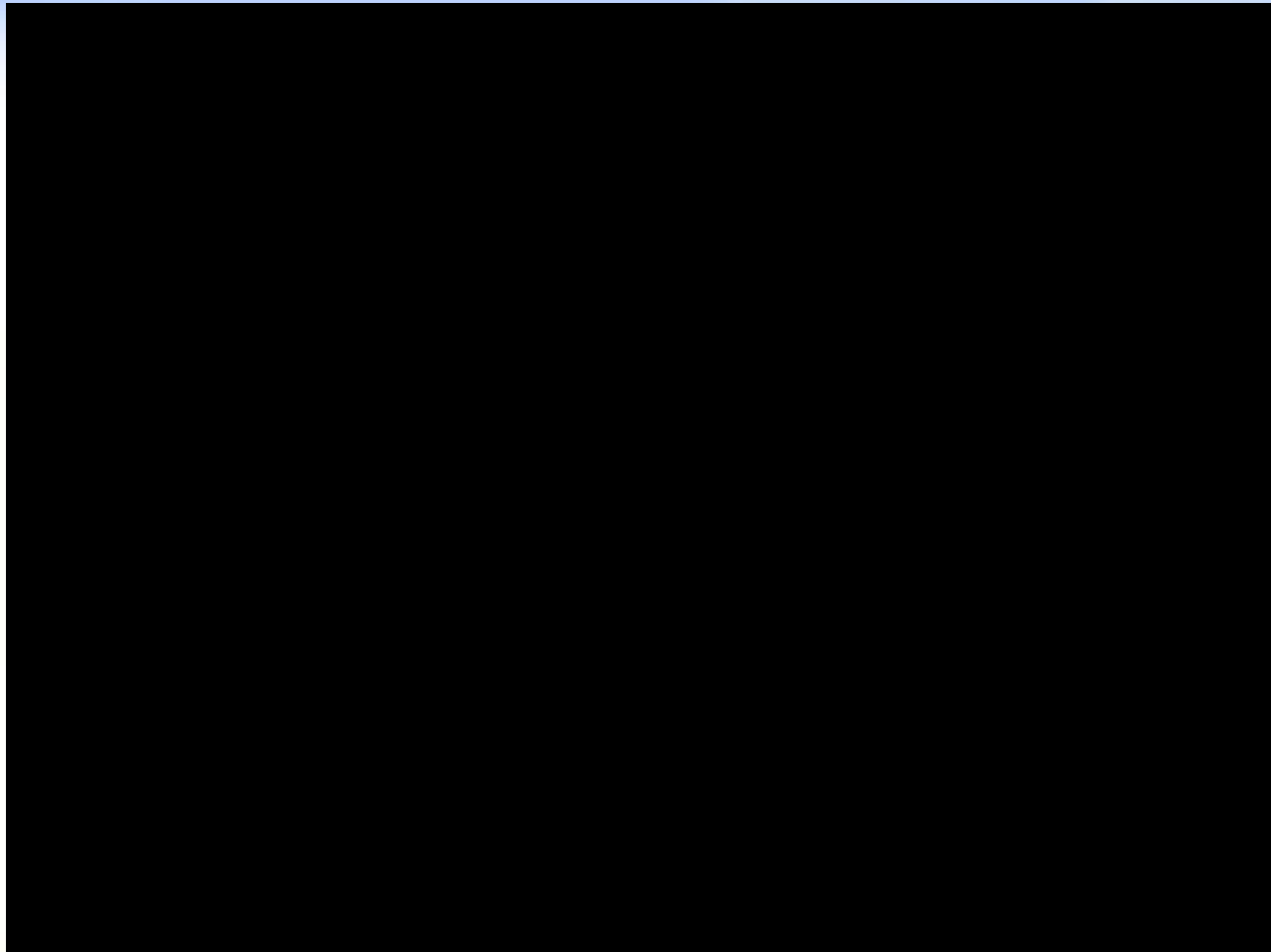
$$\text{grad}(f)|_{\mathbf{x}=\mathbf{x}^0} \approx \left(\frac{\Delta f}{\Delta x_1} \mathbf{i}_{x_1} \right) \Big|_{\mathbf{x}=\mathbf{x}^0} + \left(\frac{\Delta f}{\Delta x_2} \mathbf{i}_{x_2} \right) \Big|_{\mathbf{x}=\mathbf{x}^0} + \dots + \left(\frac{\Delta f}{\Delta x_D} \mathbf{i}_{x_D} \right) \Big|_{\mathbf{x}=\mathbf{x}^0}$$

- Наредна тачка

$$x_k^1 = x_k^0 - s \frac{\left(\frac{\Delta f}{\Delta x_k} \right) \Big|_{\mathbf{x}=\mathbf{x}^0}}{\|\text{grad}(f)\|_{\mathbf{x}=\mathbf{x}^0}}, \quad k = 1, 2, \dots, D$$

- s је дужина корака која се задаје

Илустрација (10 пута поновљен градијентни метод)



Предности и мане градијентног метода

- Локални оптимизациони алгоритам
- Изузетно брза конвергенција
- Додатно повећање брзине конвергенције:
уколико је добро процењен правац
повећати корак у следећој итерацији
- Проналази само најстрмији минимум
у околини полазног решења
(то није нужно и најдубљи минимум!)
- Ефикасан уколико знамо добро полазно решење
- У D -димензионом простору захтева
 $D+1$ итерација за процену градијента
- Корак за процену градијента Δx_i зависи од проблема!

Хесијан матрица

- Претпоставимо

$f(\mathbf{x}): R^D \rightarrow R$ и
постоји други извод

- Хесијан:

је матрица
израчуната у $\mathbf{x}_k$

$$H(f(\mathbf{x}_k)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_D} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_D} & \frac{\partial^2 f}{\partial x_2 \partial x_D} & \dots & \frac{\partial^2 f}{\partial x_D^2} \end{bmatrix}_{\mathbf{x}_k}$$

- Алгоритам за рачунање наредне итерације:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - H(f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$$

Пример коришћења Хесијана

- Функција $f(a,b) = a^2 + b^2$
- Полазно решење $\mathbf{x}_1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$
- Градијент $\nabla f(\mathbf{x}_1) = \left. \begin{bmatrix} 2a \\ 2b \end{bmatrix} \right|_{\mathbf{x}_1} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$
- Хесијан $H(f(\mathbf{x}_1)) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$
- Наредно решење $\mathbf{x}_2 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Предности и мане оптимизације засноване на Хесијану

- У претходном примеру даје решење у једном кораку за произвољну полазну тачку
- Уколико оптимизациона функција има непрекидан први и други извод, оптимизација заснована на Хесијану је ефикаснија од градијентне методе
- Које се решење добија уколико оптимизациона функција има више од једног минимума?
- Израчунавање двоструких парцијалних извода може да буде проблематично код практичних оптимизационих проблема
- ...или први и други извод не морају да постоје

Практична употреба Хесијана

- Оптимизациона функција са више променљивих

- Тејлоров развој

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H}(\mathbf{x}) \Delta\mathbf{x} + \dots$$

- Проблем је рачунање Хесијана
- Најпознатији је
Broyden–Fletcher–Goldfarb–Shanno
алгоритам

Broyden–Fletcher–Goldfarb–Shanno (BFGS) алгоритам

- $\mathbf{x}_0$ је полазна тачка, $\mathbf{B}_0$ је нулта ест. Хес., типично $\mathbf{I}$
- У $\mathbf{x}_k$ одреди се правац претраге $\mathbf{p}_k$

$$\mathbf{B}_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

- Промени се позиција $\mathbf{s}_k = \alpha_k \mathbf{p}_k$ $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$
- Нека је $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
- Естимација Хесијана у наредној тачки је

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k}$$

- Понавља се за наредну тачку у простору

Комплетно и парцијално решење

- Алгоритми који раде са комплетним решењем
 - потпуно дефинисани проблем
 - заустављање алгоритма даје до тада најбоље решење
 - примери: систематско претраживање, случајно претраживање, hill-climbing, градијентни метод...
- Алгоритми који раде са парцијалним решењем
 - апроксимативно решење (surrogate models, fitness fitting, fitness approximation, space mapping, Kriging, response surface methodology...)
 - заустављени алгоритам не мора да има корисно решење (случајно претраживање без памћења најбољег пронађеног...)
- Инжењерски употребљиви су алгоритми који дају најбоље решење до тада

Constrained vs. Unconstrained optimization

- Како алгоритам који решава оптимизационе проблеме без ограничења искористити за решавање опт. проблема са ограничењима?
- Најједноставнији приступ:
уколико ограничења нису задовољена додати “пенал” на оптимизациону функцију
$$F(\mathbf{x}) = f(\mathbf{x}) + f_p(g(\mathbf{x}))$$
где је $g(\mathbf{x})$ услов који није испуњен, а $f_p(g(\mathbf{x}))$ додатна функција (“пенал”) који се додаје

Укључивање ограничења у оптимизациону функцију

- Пример

$$\text{minimize } f(x_1, x_2) = (x_1 + 1)^2 + x_2^2 + 1$$

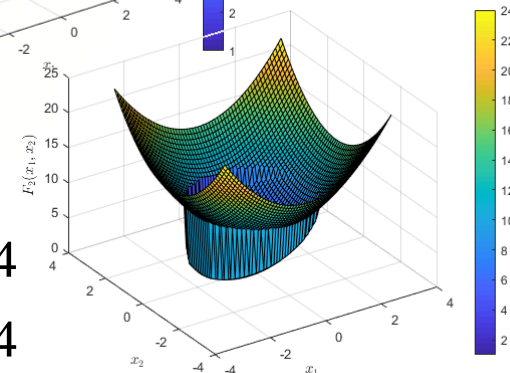
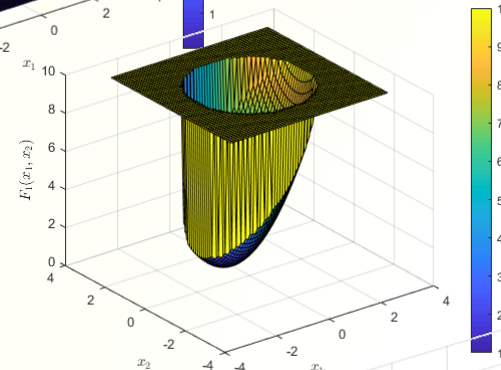
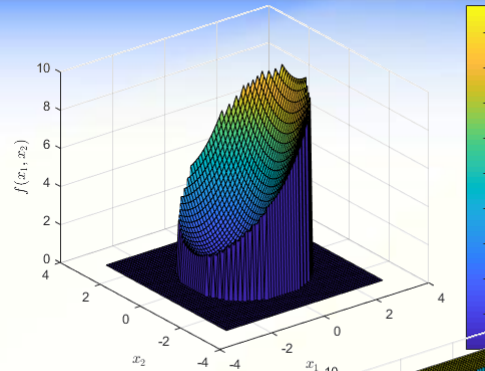
$$g(x_1, x_2) : x_1^2 + x_2^2 \leq 4$$

- Једноставно решење

$$\text{minimize } F_1(x_1, x_2) = f(x_1, x_2) + \begin{cases} 0, & x_1^2 + x_2^2 \leq 4 \\ 10, & x_1^2 + x_2^2 > 4 \end{cases}$$

- Боље (сложеније) решење

$$\text{minimize } F_2(x_1, x_2) = f(x_1, x_2) + \begin{cases} 0, & x_1^2 + x_2^2 \leq 4 \\ 10 + (x_1^2 + x_2^2 - 4), & x_1^2 + x_2^2 > 4 \end{cases}$$



Nelder-Mead

Симплекс Алгоритам

- Основне референце
 - J. A. Nelder, R. Mead “A Simplex Method for Function Minimization”, *The Computing Journal* 7, pp. 308-313, 1965.
 - Lagarias, J. C., J. A. Reeds, M. H. Wright, and P. E. Wright. “Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions.” *SIAM Journal of Optimization*. Vol. 9, Number 1, 1998, pp. 112–147.
 - Gao, F. and Han, L. “Implementing the Nelder-Mead simplex algorithm with adaptive parameters,” *Computational Optimization and Applications*, 51:1, 2012, pp. 259-277.
- Један од најчешће коришћених алгоритама за локалну NLP оптимизацију
- Често се референцира само под именом симплекс или Nelder-Mead
- Понекад се назива и “амоеба” алгоритам

Имплементације

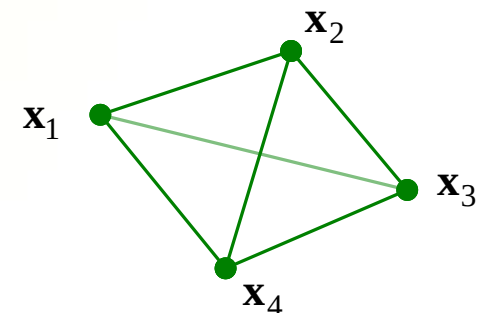
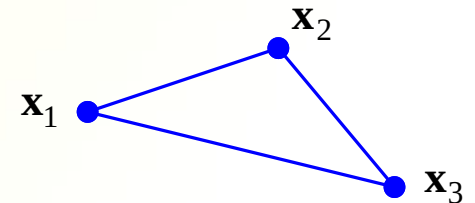
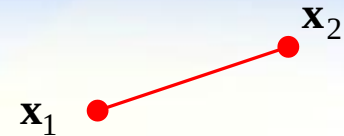
- MATLAB:
 - `fminsearch` uses the **Nelder-Mead** simplex algorithm as described in Lagarias et al.
- Mathematica:
 - The Nelder-Mead method is implemented as `NMinimize[f, vars, Method -> "NelderMead"]`.
- Python:
 - `scipy.optimize.minimize(fun, x0, args=(), method='Nelder-Mead', ...)`
- Numerical recipes in C: The art of scientific computing
 - void **amoeba**(float **p, float y[], int ndim, float ftol, float (*funk)(float []), int *nfunk)
[Multidimensional minimization of the function funk(x) based on Nelder, J.A., and Mead, R. 1965, Computer Journal, vol. 7, pp. 308-313]

Искључиво се користе вредности описне функције

- Алгоритам користи искључиво вредности описне функције
- Не рачуна изводе описне функције (ни експлицитно ни имплицитно)
- Спада у класу директних алгоритама (алгоритми који не користе изводе опт. ф.)
- Алгоритам се заснива на трансформацијама геометријске фигуре под именом симплекс

Симплекс фигура

- Симплекс фигура има ненулту “зап्रेмину”
- Састоји се од $D+1$ тачке у D димензионом оптимизационом простору
 - за 1-D проблем симплекс је дуж (“запремина” је дужина дужи)
 - за 2-D проблем симплекс је троугао (“запремина” је површина троугла)
 - за 3-D проблем симплекс је тетраедар итд. (“запремина” је запремина тетраедра)
 - ...
- Идеја: поступно се мења једна по једна тачка симплекса док се он не смањи тако да се све тачке симплекса налазе у епсилон околини решења



Параметри симплекс алгоритма

- У општем случају постоје четири коефицијента који се задају унапред
 - коефицијент рефлексije α ,
 - коефицијент контракције β ,
 - коефицијент експанзије γ и
 - коефицијент сажимања σ
- Коефицијент сажимања није експлицитно дефинисан у оригиналном раду у коме је алгоритам описан (усвојен је као константа)

Избор параметара симплекс алгоритма

- Ови параметри задовољавају следеће услове

$$\alpha > 0, \quad 0 < \beta < 1, \quad \gamma > 1, \quad 0 < \sigma < 1$$

- Готово универзалан избор вредности ових параметара

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

- Ове вредности установљене су још у оригиналном раду, а даља истраживања су потврдила да је овакав избор врло добар за велики број различитих проблема

Почетак симплекс алгоритма

- Алгоритам почиње формирањем $D+1$ тачке симплекса (обавезно ненулте запремине) у простору и израчунавањем оптимизационе функције у тим тачкама
- На почетку k -тог корака алгоритма, задато је $D+1$ тачака које заједно формирају симплекс
- Сваки корак алгоритма почиње сортирањем и обележавањем ових тачака $\mathbf{x}_1(k), \mathbf{x}_2(k), \dots, \mathbf{x}_{D+1}(k)$

$$f_i^{(k)} = f(\mathbf{x}_i^{(k)}) \quad f_1^{(k)} \leq f_2^{(k)} \leq \dots \leq f_{D+1}^{(k)}$$

Најбоље и најгоре тачке и функције у оквиру алгоритма

- У сваком кораку алгоритма формира се нови скуп тачака (симплекс) који је различит од симплекса у претходном кораку
- Циљ алгоритма је проналажење минимума функције f
 - тачка $\mathbf{x}_1(k)$ се назива најбоља тачка,
 - $\mathbf{x}_{D+1}(k)$ је најгора тачка, а
 - $\mathbf{x}_D(k)$ је друга најгора тачка
- Најбоља функција грешке је $f_1^{(k)}$, а најгора функција грешке је $f_{D+1}^{(k)}$

Један корак симплекс алгоритма

- Један корак алгоритма састоји се из следећих пет операција
 - (1) сортирање
 - (2) рефлексација
 - (3) експанзија
 - (4) контракција
 - (5) сажимање
- Само неке операције се извршавају у оквиру једног корака
- Која ће се операција (или операције) извршити зависи од текућег стања

1. Сортирање

(1) Сортирање: сортирати темена симплекса тако да је испуњен услов

$$f_1 \leq f_2 \leq \dots \leq f_{D+1}$$

Алгоритам за сортирање није критичан, јер симплекс по правилу има до највише неколико стотина тачака

2. Рефлексија

(2) **Рефлексија:** Израчунати центроид D најбољих тачака,

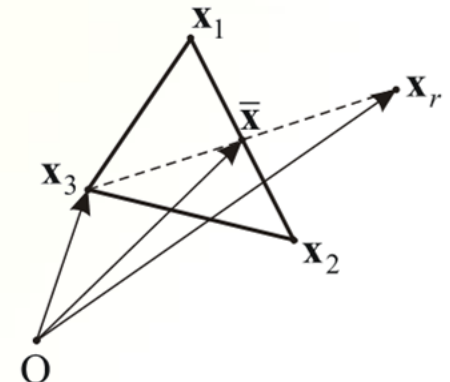
$$\bar{\mathbf{x}} = \frac{1}{D} \sum_{i=1}^D \mathbf{x}_i$$

и у односу на њега израчунати тачку рефлексије $\mathbf{x}_r$ према формули

$$\mathbf{x}_r = \bar{\mathbf{x}} + \alpha \cdot (\bar{\mathbf{x}} - \mathbf{x}_{D+1}) = (1 + \alpha) \cdot \bar{\mathbf{x}} - \alpha \cdot \mathbf{x}_{D+1}$$

Израчунати f_r

Уколико је $f_1 \leq f_r < f_D$ изоставити најгору тачку, а уместо ње уврстити $\mathbf{x}_r$ у симплекс и завршити корак



3. Експанзија

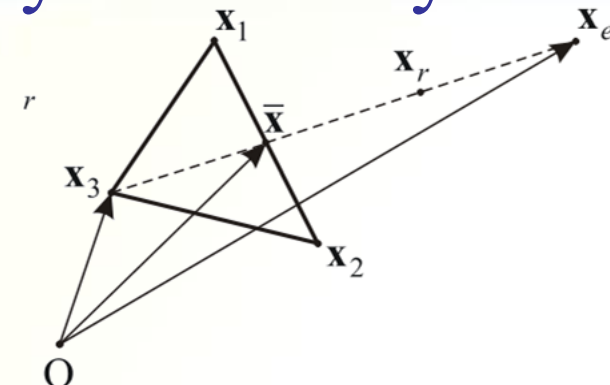
(3) **Експанзија:** ако је $f_r < f_1$ израчунати тачку експанзије $\mathbf{x}_e$ према формули

$$\mathbf{x}_e = \bar{\mathbf{x}} + \gamma \cdot (\mathbf{x}_r - \bar{\mathbf{x}}) = \gamma \cdot \mathbf{x}_r + (1 - \gamma) \cdot \bar{\mathbf{x}}$$

Израчунати f_e

Ако је $f_e < f_r$ изоставити најгору тачку и уместо ње уврстити $\mathbf{x}_e$ у симплекс и завршити корак

У супротном, ако је $f_e \geq f_r$ уврстити $\mathbf{x}_r$ и завршити корак



4. Контракција

(4) **Контракција:** Ако је $f_r \geq f_D$ доделити

$$\mathbf{x}_{D+1} = \begin{cases} \mathbf{x}_r, & f_r \leq f_{D+1} \\ \mathbf{x}_{D+1}, & f_r > f_{D+1} \end{cases}$$

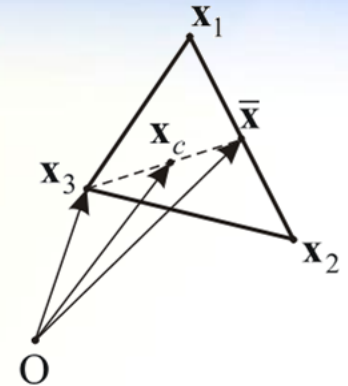
израчунати тачку контракције $\mathbf{x}_c$ као

$$\mathbf{x}_c = \bar{\mathbf{x}} + \beta \cdot (\mathbf{x}_{D+1} - \bar{\mathbf{x}}) = \beta \cdot \mathbf{x}_{D+1} + (1 - \beta) \cdot \bar{\mathbf{x}}$$

Израчунати f_c , уврстити $\mathbf{x}_c$ и завршити корак ако је $f_c \leq \min(f_{D+1}, f_r)$,

(тј. тачка контракције је боља од $\mathbf{x}_{D+1}$ и $\mathbf{x}_r$)

У супротном извршити сажимање



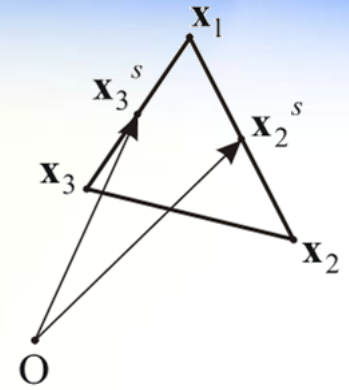
5. Сажимање

(5) **Сажимање:** заменити све тачке осим најбоље са

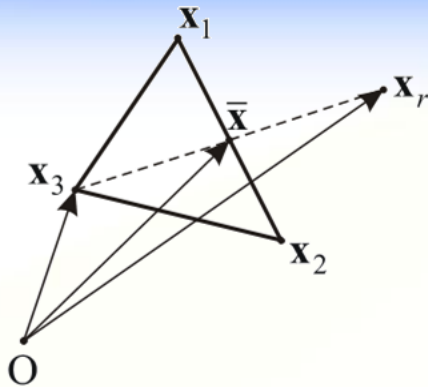
$$\mathbf{x}_i = \mathbf{x}_1 + \sigma \cdot (\mathbf{x}_i - \mathbf{x}_1), \quad i = 2, 3, \dots, D+1$$

израчунати функцију грешке у свим новим тачкама и завршити корак

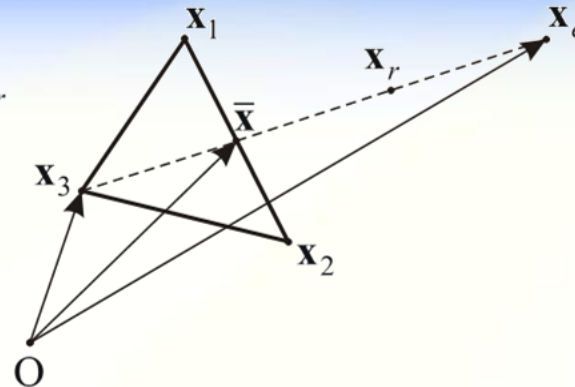
* По завршетку једног корака, а пре преласка у наредни корак потребно је извршити проверу да ли је алгоритам испунио услове за завршетак оптимизације



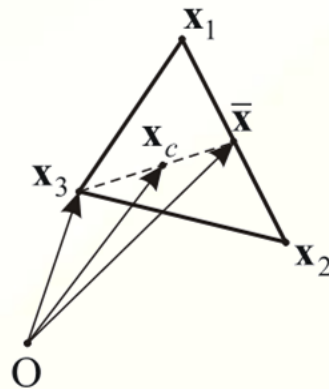
Графички приказ трансформација симплекса у 2-D простору



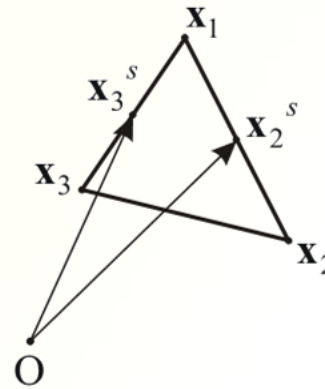
(a) Рефлексија



(б) Експанзија

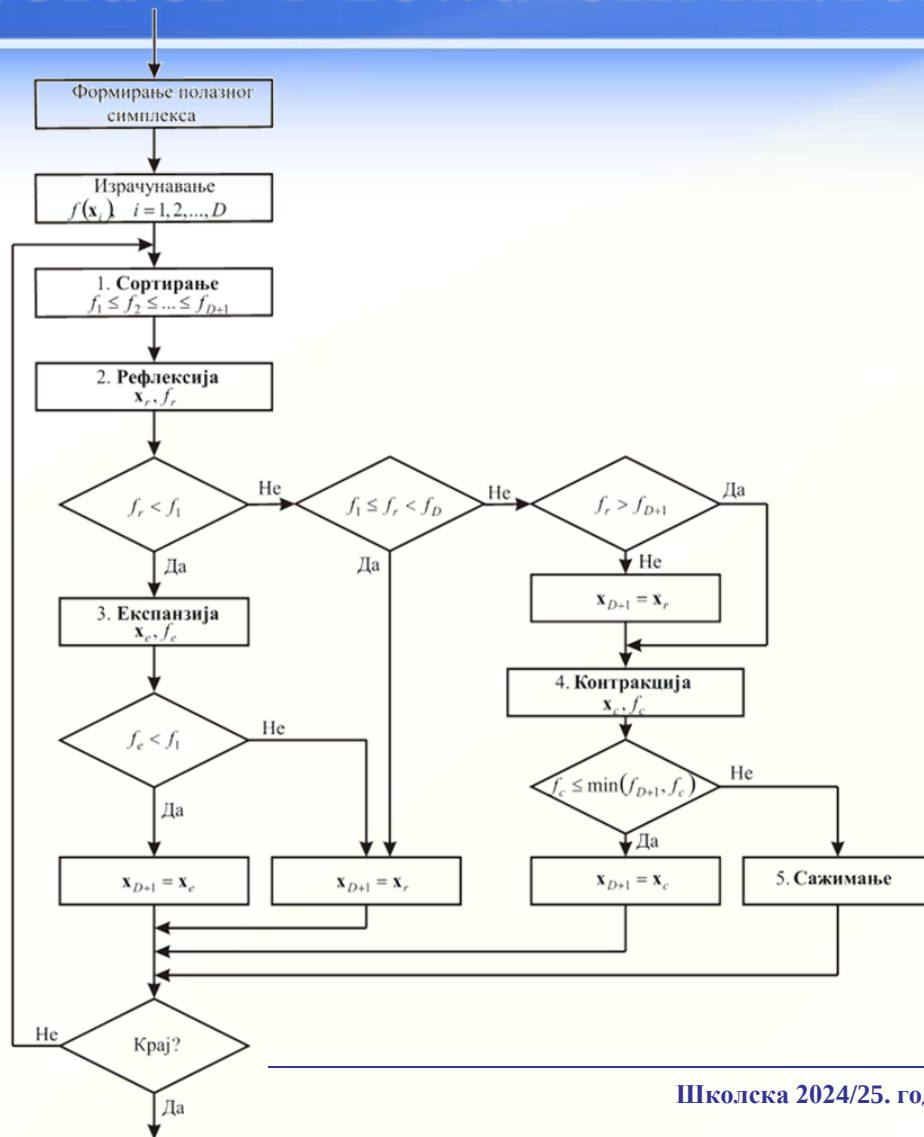


(в) Контракција



(г) Сажимање

Блок дијаграм Nelder-Mead симплекс алгоритма



Формирање полазног симплекса

- Типично је позната једна тачка $\mathbf{x}_0$ (почетно решење), осталих D је потребно формирати
- Најједноставнији приступ (хипер-правоугаоник):
 $\mathbf{x}_1 = \mathbf{x}_0 + (\Delta x_1, 0, \dots, 0)$
 $\mathbf{x}_2 = \mathbf{x}_0 + (0, \Delta x_2, \dots, 0)$
...
 $\mathbf{x}_D = \mathbf{x}_0 + (0, 0, \dots, \Delta x_D)$
- Проблеми:
 - Како изабрати Δx_k ?
 - Неке тачке могу изаћи из оптимизационог простора
- Провера да ли је тачка у оптимизационом простору и ако није промена знака Δx_k
- Могуће је користити и друге D димензионе фигуре (сфера, коцка, итд.)

Адаптивна промена параметара

- Параметри алгоритма могу се подешавати у односу на број димензија оптимизационог простора D
- Један могући приступ

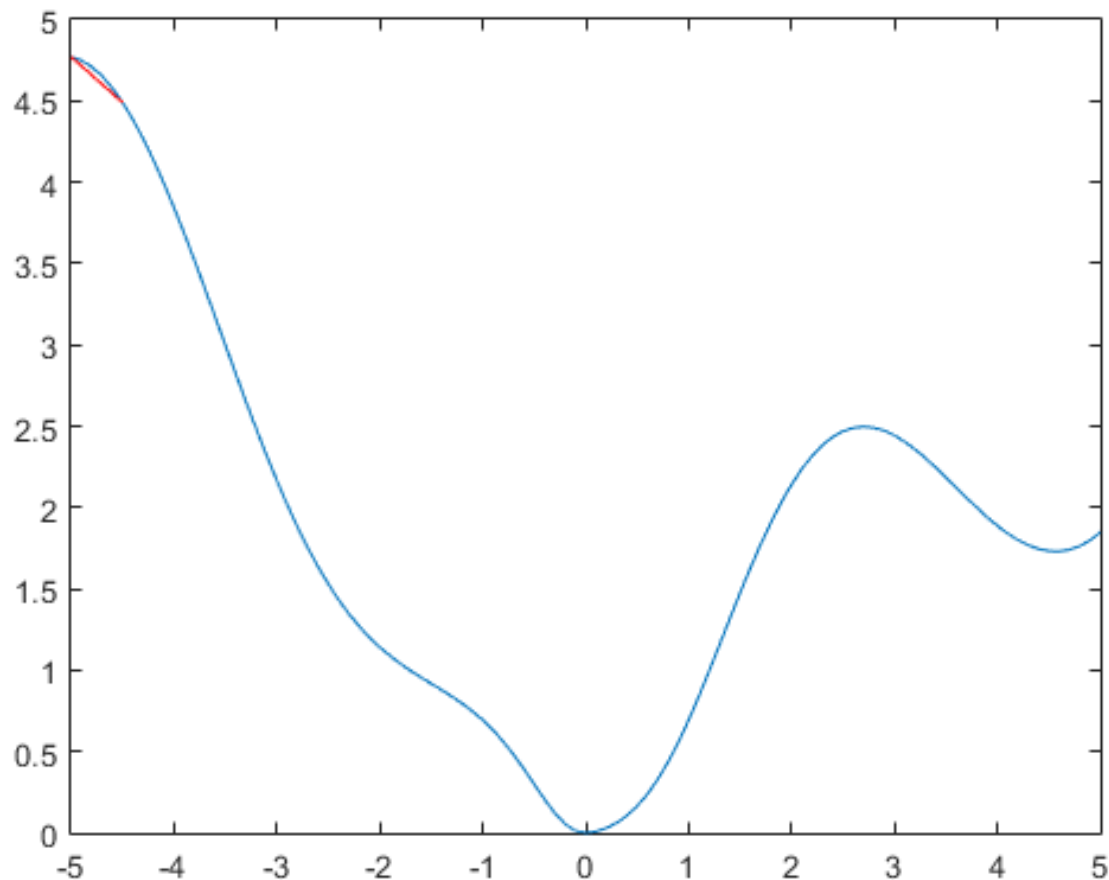
$$\alpha = 1, \quad \beta = \frac{3}{4} - \frac{1}{2D}, \quad \gamma = 1 + \frac{2}{D}, \quad \sigma = 1 - \frac{1}{D}$$

- Повољно у случајевима “великог” D (од ~ 10 до ~ 30)
- Python имплементација адаптира параметре у зависности од D
- Оптимизација параметара оптимизационог алгоритма (енглески: *metaheuristics*)

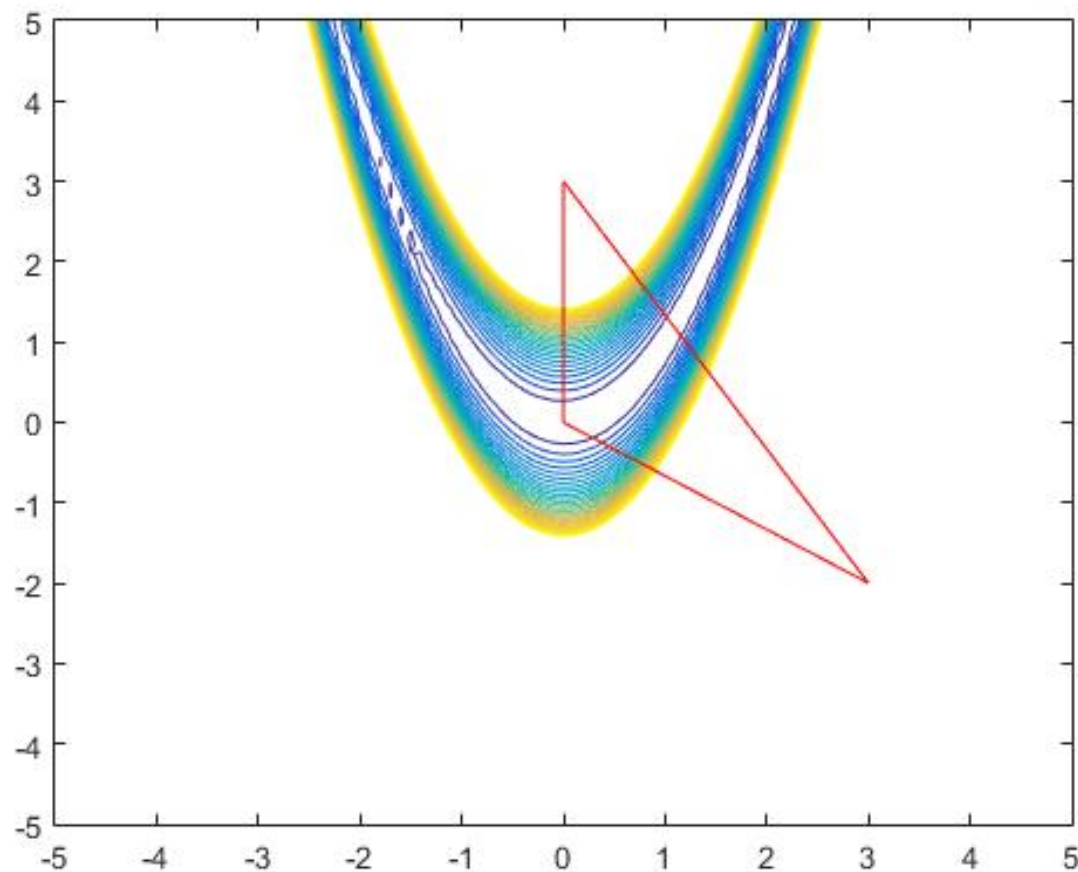
Колика је сложеност симплекс алгоритма?

- У пракси алгоритам типично конвергира за $O(D^2)$ итерација тј. $\sim D^2$ израчунавања оптимизационе функције
- У литератури постоје теоријска разматрања само за мањи број димензија оптимизационог простора, типично до 10
- У општем случају, није позната сложеност овог алгоритма

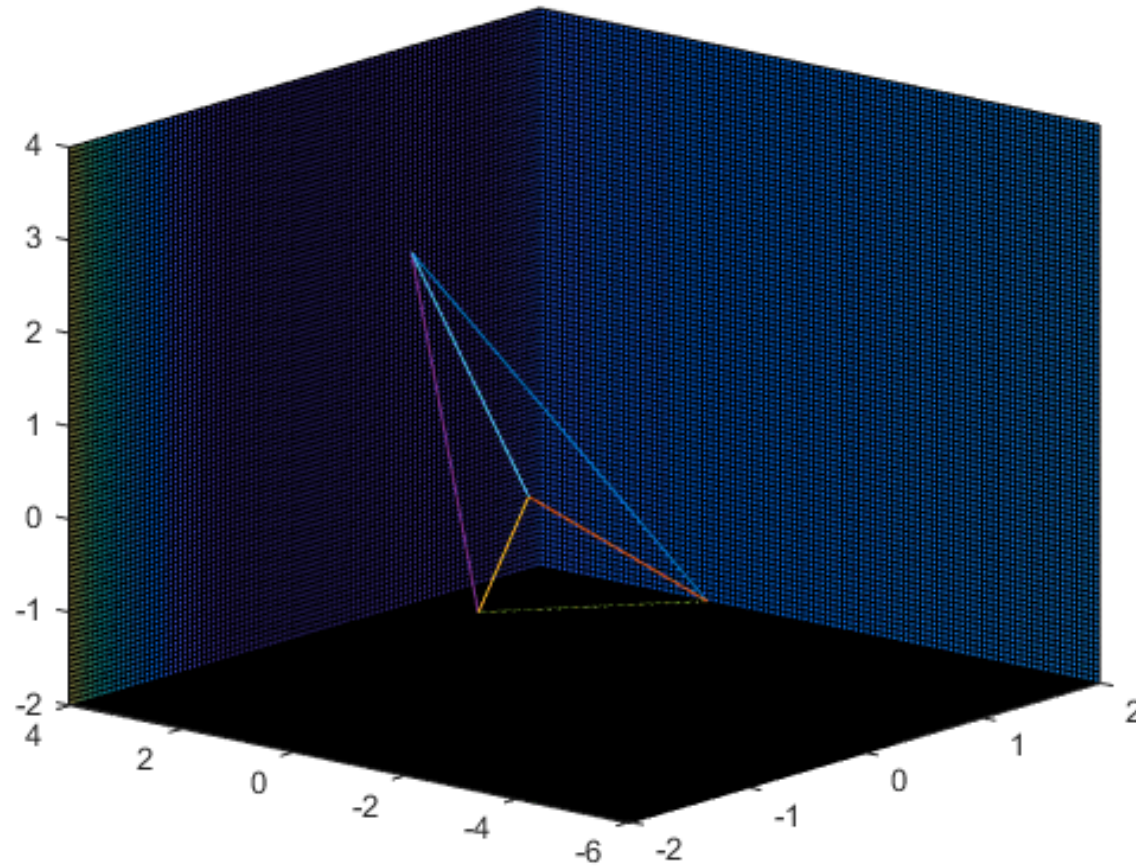
Илустрација рада симплекс алгоритма (1D)



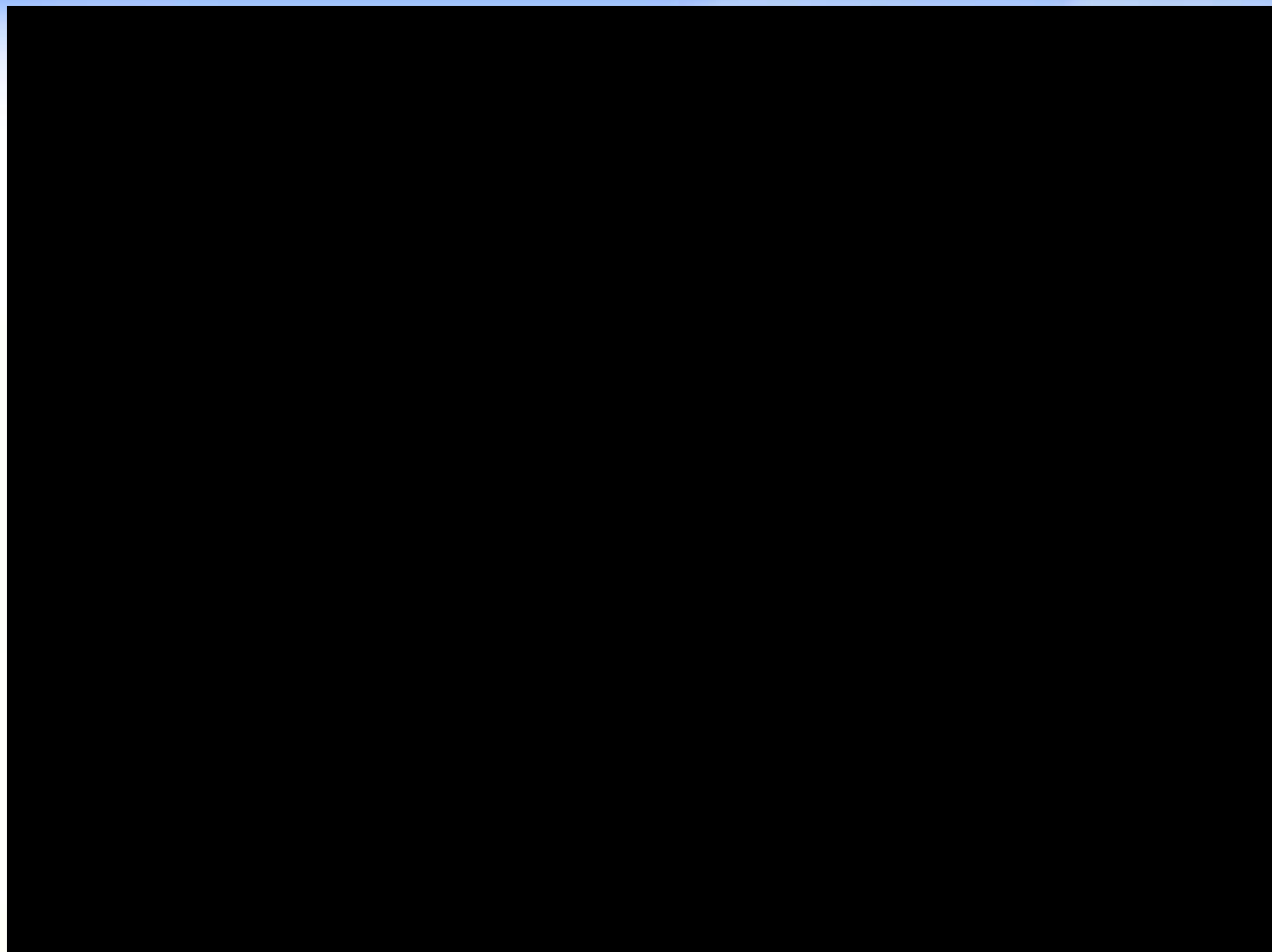
Илустрација (2D)



Илустрације (3D)



Ток симплекс алгоритма 10 пута

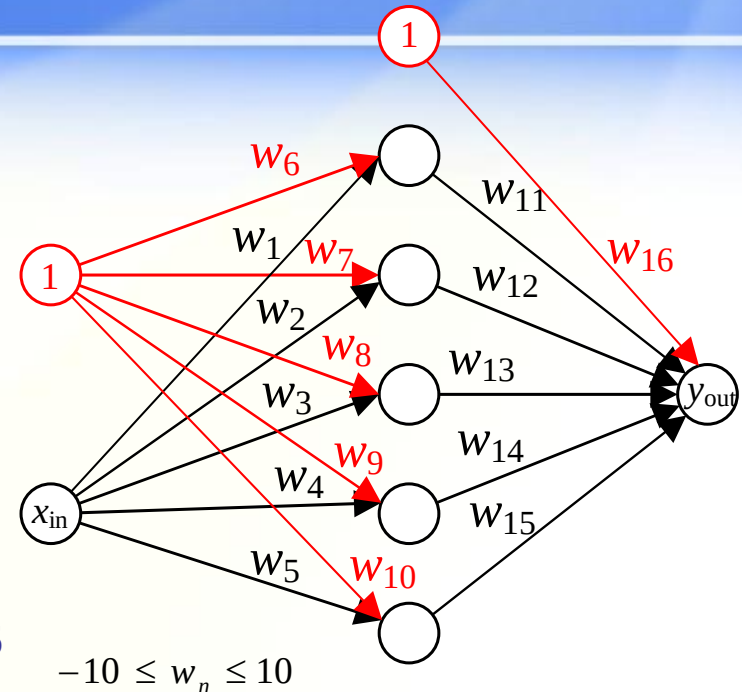


Практични закључци о симплекс алгоритму

- Спада у групу локалних оптимизационих метода
- Поседује способност да,
у одређеним ситуацијама,
прескочи из једног локалног минимума
у други уколико се тиме добија боље решење
- Ефекат “прескакања” се не дешава увек и зависи од
 - конкретног проблема и
 - полазног симплекса
- Ова особина даје предност симплекс алгоритму у односу на (све)
друге локалне оптимизационе алгоритме
- Постоје напори да се строго математички докаже због чега
симплекс алгоритам постиже врло добре резултате у пракси,
међутим генералног доказа нема
- Симплекс се може посматрати и као
генерализација метода половљења интервала

Задатак за вежбе (поставка)

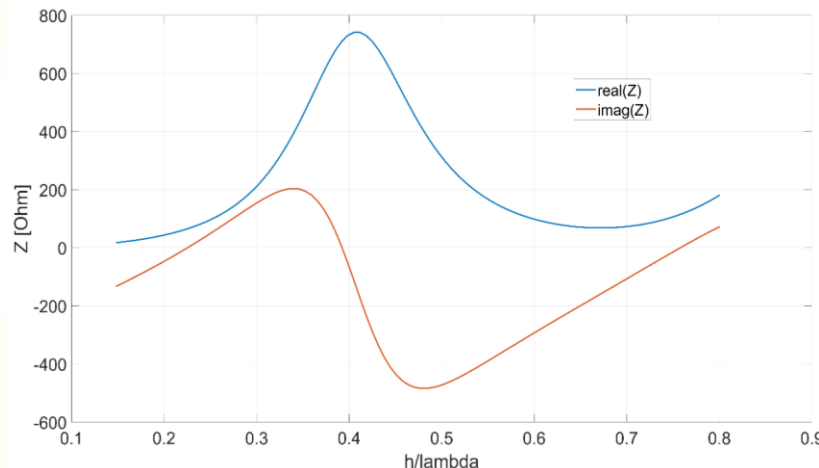
- На слици је приказана једноставна неурална мрежа са једним улазом, једним излазом и једним скривеним слојем са 5 чворова (неурона)
- Улаз мреже је x_{in}
излаз мреже је y_{out}
коефицијенти мреже су $w_1, w_2, \dots, w_{16}$
- Активациона функција сваког чвора скривеног слоја је дата изразом (тангенс хиперболички)
- Активациона функција излазног чвора је линеарна, $a(x) = x$



$$a(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Задатак за вежбе (поставка)

- Оптимизацијом је потребно пронаћи вредности тежинских коефицијената $\mathbf{w} = (w_1, w_2, \dots, w_{16})$ тако да мрежа представља апроксимацију функције $y_{\text{training}}(x)$ чији су одбирци у фајлу data.txt у формату x y
- Ова функција представља нормализовани модул импедансе танке цилиндричне антене у зависности од односа дужине њеног крака и таласне дужине



Задатак за вежбе (деталји)

- Вредност на улазу (x_k) и излазу чвора (y_k) је
$$x_k = w_k x_{in} + w_{k+5}, \quad k = 1, 2, \dots, 5$$
$$y_k = a(x_k)$$
- Вредност на излазу мреже дата је изразом
$$y_{out}(x_{in}) = w_{16} + \sum_{k=1}^5 w_{k+10} a(w_k x_{in} + w_{k+5})$$
- Оптимизациона функција је дата изразом (средња квадратна грешка, енг: mean square error, скраћено MSE)

$$f(\mathbf{w}) = \frac{1}{100} \sum_{k=1}^{100} (y_{out}(x_k, \mathbf{w}) - y_{training}(x_k))^2$$

$x_k, k = 1, \dots, 100$ су одбирци из фајла data.txt

Задатак (имплементација)

- Написати код који рачуна излаз задате мреже, на основу улаза
- Написати код који рачуна оптимизациону функцију за један избор тежинских коефицијената
- Користити једну од следећих ставки
 - C/C++ код за Nelder-Mead симплекс
<http://mtt.etf.rs/si/IOA/NMSimplex.zip>
 - Python функција
`scipy.optimize.minimize(..., method='nelder-mead', ...)`
 - или код неког другог класичног метода обрађеног на курсу
- Одредити коефицијенте $\mathbf{w} = (w_1, w_2, \dots, w_{16})$ тако да оптимизациона функција буде мања од $5 \cdot 10^{-4}$
- Дозвољени опсег за коефицијенте је $-10 \leq w_1, w_2, \dots, w_{16} \leq 10$
- У пратећи ASCII фајл записати
 - коефицијенте $\mathbf{w}$ са 15 цифара после децималне тачке у облику $[w_1, w_2, \dots, w_{16}]$,
 - написати вредност минималне пронађене оптимизационе функције
- Нацртати на истом графику $y_{\text{out}}(x), y_{\text{training}}(x), -1 \leq x \leq 1$