

Metody numeryczne

12. Minimalizacja: funkcje wielu zmiennych

Daleko od minimum

P. F. Góra

https://zfs.fais.uj.edu.pl/pawel_gora

13 stycznia 2026

Minimalizacja funkcji wielu — niekiedy **bardzo** wielu — zmiennych jest jednym z podstawowych zastosowań metod numerycznych. Zarazem może to być problem o wielkim koszcie numerycznym. Dlatego, w zależności od okoliczności, stosuje się **dwie różne strategie**:

- Jeśli jesteśmy **daleko od poszukiwanego minimum**, szybko podążamy w jego kierunku, nie starając się jednak znaleźć dokładnego położenia minimum. Jest to strategia szczególnie często stosowana w problemach o dużej wymiarowości.
- Jeśli jesteśmy **blisko poszukiwanego minimum** i zależy nam na jego dokładnym zlokalizowaniu, staramy się o taką dokładną lokalizację, wiedząc, że może ona być kosztowna nawet w problemach kilkuwymiarowych.

W całym wykładzie rozważamy funkcje $f : \mathbb{R}^N \rightarrow \mathbb{R}$, odpowiednio wiele razy różniczkowalne.

Podstawową zasadą algorytmów minimalizacyjnych jest, że **idziemy zawsze “w dół”, w stronę malejących wartości funkcji**. To sprawia, że minimalizacja jest numerycznie prostsza od ogólnego problemu rozwiązywania układów równań algebraicznych (formalnie, matematycznie, znalezienie minimum oznacza rozwiązanie równania $\nabla f = 0$), ale oznacza też, że przedstawione tu algorytmy znajdują jedynie **minimum lokalne**.

Daleko od minimum: metoda najszybszego spadku

Daleko od minimum staramy się wyłącznie o to, aby w kolejnych krokach naszej procedury wartości funkcji malały. Ponieważ gradient funkcji wielu zmiennych pokazuje kierunek jej najszybszego wzrostu, kierunek przeciwny pokazuje kierunek jej najszybszego spadku. A ponieważ zależy nam, aby funkcja malała tak szybko, jak to możliwe, podążamy w tym właśnie kierunku. Otrzymujemy w ten sposób **algorytm najszybszego spadku** (ang. *steepest descent* lub *gradient descent*):

Niech aktualne położenie wynosi \mathbf{x}_k , $f_k = f(\mathbf{x}_k)$.

1.

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k) \quad (1a)$$

2. $\gamma = \gamma_{\text{start}}$

3.

$$\tilde{\mathbf{x}} = \mathbf{x}_k + \gamma \mathbf{p}_k \quad (1b)$$

$$\tilde{f} = f(\tilde{\mathbf{x}}) \quad (1c)$$

4. Jeżeli $\tilde{f} < f_k$

- $\mathbf{x}_{k+1} = \tilde{\mathbf{x}}$, $f_{k+1} = \tilde{f}$
- $k = k + 1$
- goto 1

5. Jeżeli $\tilde{f} \geq f_k$

- opcjonalnie: zmniejsz γ , potem goto 3
- STOP

Ta metoda jest nieprecyzyjna, nie dostarcza kryterium czy jesteśmy blisko minimum i w zamyśle służy do szybkiego przemieszczenia się w okolicę minimum. Często — zwłaszcza gdy ciąg kolejnych wartości f_k maleje *coraz wolniej*, co interpretujemy w ten sposób, że krajobraz się “wyplaszcza” — jest to *jedyna* metoda, jakiej się w praktyce używa. Należy jednak pamiętać, że jej wyniki na ogół są nieprecyzyjne i jeżeli zależy nam na lepszym, bardziej dokładnym zlokalizowaniu minimum, powinniśmy użyć jakichś innych metod, przynajmniej w końcowej fazie minimalizacji, pod warunkiem, że nie będzie to zbyt kosztowne ze względu na wymiarowość problemu.

Współczynnik γ na ogół musi być dość mały, gdyż przy dużych wartościach gradientu ryzykujemy, że “przeskoczmy” poszukiwane minimum.

Z punktu widzenia Big Data...

W ciągu ostatnich kilkunastu lat rozmiary dostępnych zbiorów danych rosną *szybciej* niż rośnie prędkość procesorów (nawet po uwzględnieniu paralelizacji i obliczeń na GPU). Z tego punktu widzenia możliwości uczenia maszynowego są ograniczone raczej przez możliwości obliczeniowe niż przez dostępne zbiory danych. Tę klasę problemów zwyczajowo określa się jako *Big Data*.

Uczenie maszynowe bardzo często oznacza konieczność minimalizowania funkcji

$$Q(\mathbf{X}) = \frac{1}{N} \sum_{i=1}^N Q_i(\mathbf{X}; x_i, y_i) \quad (2)$$

gdzie

- \mathbf{X} jest wielowymiarową zmienną, po której minimalizujemy — wektorem estymatorów, których wartości chcemy znaleźć. Tego typu problemy pojawiają się w zagadnieniu najmniejszych kwadratów.
- $\forall i: Q_i \geq 0$,
- Q_i mają taką samą postać funkcyjną, różnią się tylko elementami x_i, y_i (elementami zbioru uczącego),
- $N \gg 1$ (N jest zazwyczaj **BARDZO** duże, rzędu kilkudziesięciu, niekiedy kilkuset tysięcy).

Problem (2) można rozwiązać za pomocą metody najszybszego spadku, którejś z bardziej precyzyjnych metod minimalizacji lub, jeżeli mowa o liniowym zagadnieniu najmniejszych kwadratów, za pomocą przybliżonego rozwiązywania nadokreślonych układów równań przy pomocy SVD. Jednak dla **bardzo dużych N** , czyli dla bardzo dużych zbiorów uczących, inne podejście może być bardziej efektywne.

Punktem wyjścia jest metoda najszybszego spadku: W każdym kroku poruszamy w kierunku minus gradientu funkcji $Q(\mathbf{X})$, teoretycznie aż do osiągnięcia minimum kierunkowego*, czyli minimum w kierunku aktualnego gradientu.

Jak jednak pamiętamy, **daleko od minimum nie minimalizujemy**, tylko poruszamy z arbitralnie dobranym krokiem w kierunku ujemnego gradientu.

*Patrz następny wykład

Dlatego zamiast minimalizować, w n -tym kroku iteracji przyjmujemy

$$\begin{aligned}\mathbf{X}_{n+1} &= \mathbf{X}_n - \gamma \cdot \nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n} \\ &= \mathbf{X}_n - \gamma \cdot \frac{1}{N} \sum_{i=1}^N \nabla_{\mathbf{X}} Q_i(\mathbf{X}; x_i, y_i)|_{\mathbf{X}_n} .\end{aligned}\tag{3}$$

$\gamma = \text{const}$ i jest nazywane *prędkością uczenia* (ang. *learning rate*).

Uwaga: Przyjęcie stałego kroku nie jest aż tak naiwne, jak by się to mogło wydawać. Jeśli rozpatrzmy układ równań różniczkowych typu

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}) \quad (4)$$

to najprostszą (zdecydowanie nie najlepszą, ale najprostszą i najszybszą) metodą jego numerycznego rozwiązywania jest *metoda Eulera*

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot \mathbf{f}(\mathbf{x}_n), \quad (5)$$

gdzie h jest stałym krokiem narastania zmiennej niezależnej x . Metoda najszybszego spadku (3) ze stałym krokiem γ ma taką samą postać, co metoda Eulera, po utożsamieniu $\mathbf{f} \equiv -\nabla Q$.

Kryterium stopu

Jakie kryterium stopu przyjąć dla algorytmu najszybszego spadku (3)?
Sensowne wydają się dwie opcje:

(i). Przesunięcie jest dostatecznie małe:

$$\begin{aligned} \|\mathbf{X}_{n+1} - \mathbf{X}_n\| &< \varepsilon_1 \\ \gamma \cdot \|\nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n}\| &< \varepsilon_1 \\ \|\nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n}\| &< \frac{\varepsilon_1}{\gamma} \end{aligned} \quad (6)$$

(ii). Zmiana wartości funkcji jest dostatecznie mała:

$$\begin{aligned} & \left| Q(\mathbf{X}_{n+1}) - Q(\mathbf{X}_n) \right| < \varepsilon_2 \\ & \left| Q(\mathbf{X}_n) + (\mathbf{X}_{n+1} - \mathbf{X}_n)^T \nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n} - Q(\mathbf{X}_n) \right| < \varepsilon_2 \quad (7) \end{aligned}$$

gdzie skorzystaliśmy z rozwinięcia Taylora do pierwszego rzędu. Ponieważ

$$\left| (\mathbf{X}_{n+1} - \mathbf{X}_n)^T \nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n} \right| \leq \|\mathbf{X}_{n+1} - \mathbf{X}_n\| \cdot \|\nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n}\| \quad (8)$$

to jeżeli wielkość po prawej stronie (8) jest mniejsza od ε_2 , to także zmiana wartości funkcji jest mniejsza. Z kolei

$$\|\mathbf{X}_{n+1} - \mathbf{X}_n\| \cdot \|\nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n}\| < \varepsilon_2 \quad (9)$$

jeśli przesunięcie jest dostatecznie małe lub gdy gradient jest dostatecznie mały. Stwierdziliśmy jednak, że gdy dostatecznie mały jest gradient, małe jest też przesunięcie.

Zatem jako kryterium stopu przyjmujemy

$$\left\| \nabla_{\mathbf{X}} Q(\mathbf{X})|_{\mathbf{X}_n} \right\| < \varepsilon. \quad (10)$$

Metodę najszybszego spadku zatrzymujemy, gdy gradient funkcji jest dostatecznie mały — funkcja “wyplaszcza się”. Przyjęcie takiego kryterium dodatkowo uwalnia nas od konieczności obliczania wartości funkcji w każdym kroku.

Stochastic Gradient Descent

Jeśli w (3) $N \gg 1$, czas obliczania sumy gradientów cząstkowych $\sum_i^N \nabla Q_i$ może być bardzo znaczny. Zarazem wyrażenie $\frac{1}{N} \sum_i^N \nabla Q_i$ ma postać średniego gradientu cząstkowego. Jeśli założymy, że **poszczególne elementy tej sumy nie odbiegają zbytnio od średniej**, możemy średni, kosztowny w wyliczaniu gradient, zastąpić *losowo wybranym* gradientem cząstkowym. Otrzymujemy zatem algorytm ***Stochastic Gradient Descent***:

- wybierz losowo $k \in \{1, 2, \dots, N\}$
- przyjmij

$$\mathbf{X}_{n+1} = \mathbf{X}_n - \gamma \nabla_{\mathbf{X}} Q_k(\mathbf{X}; x_k, y_k) |_{\mathbf{X}_n} \quad (11)$$

- **Dodatkowo akceptujemy powyższy krok tylko jeżeli $Q(\mathbf{X}_{n+1}) < Q(\mathbf{X}_n)$. Jeśli ta nierówność nie zachodzi, nie wykonujemy kroku, ale losujemy nowe k . Ten wariant nazwiemy *Stochastic Gradient Descent z ograniczeniem*.**

Iterację kończymy albo po wykonaniu z góry określonej liczby kroków, albo — częściej — gdy spełnione zostanie kryterium (10), gdzie zamiast pełnego gradientu bierzemy wylosowany gradient cząstkowy. Rodzi to jednak niebezpieczeństwo, że wylosowany gradient cząstkowy przypadkowo będzie mały, podczas gdy cały gradient — niekoniecznie. Bezpieczne byłoby kryterium, że zmiana funkcji jest dostatecznie mała:

$$\left| Q(\mathbf{X}_n) - Q(\mathbf{X}_{n+1}) \right| < \varepsilon, \quad (12)$$

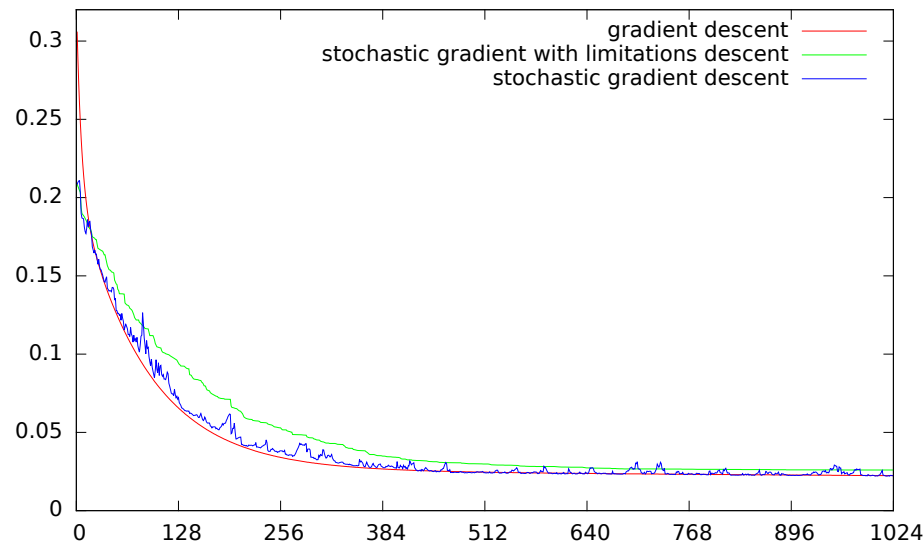
gdzie ε jest ustaloną tolerancją. Powoduje to jednak konieczność obliczania wartości Q w każdym kroku, co z kolei może być dużym obciążeniem, gdy liczba składników w (3) jest bardzo duża.

Przy zastosowaniu wariantu “z ograniczeniem” i tak musimy korzystać z kryterium (12).

Wartość *prędkości uczenia* γ oraz tolerancję ε dobieramy “eksperymentalnie”, to znaczy kierując się znajomością natury problemu i doświadczeniem uzyskanym przy rozwiązywaniu podobnych problemów.

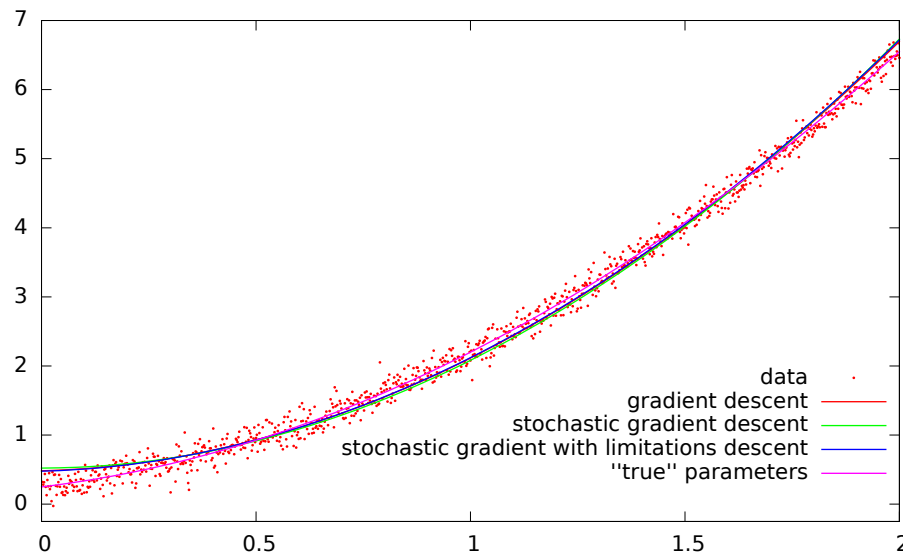
Okazuje się, że metoda *Stochastic Gradient Descent* działa *zdumiewająco dobrze dla bardzo dużych zbiorów uczących*. Co ciekawe, przyjęcie warunku, iż *wartość funkcji Q musi maleć po wykonaniu kroku* na ogół *pogarsza* wyniki: Od czasu do czasu można/trzeba wykonać krok “w złą stronę”.

Przykład



Wygenerowano $N = 1024$ punktów $y_i = ax_i^2 + bx_i + c + \xi_i$, gdzie $\{\xi_i\}_{i=1}^N$ są liczbami o rozkładzie normalnym, natomiast $x_i \in [0, 2]$. Rysunek przedstawia kolejne wartości funkcji $Q(a, b, c)$ uzyskane w metodzie najszybszego spadku (czerwony), w metodzie *Stochastic Gradient Descent* (niebieski) oraz *Stochastic Gradient Descent* z ograniczeniami (zielony). $\gamma = 1/128$. W przypadku *Stochastic Gradient Descent* wartość minimalizowanej funkcji niekiedy rośnie, ale po dostatecznie dużej liczbie iteracji wyniki tej metody

są nieco *lepsze*, niż dla metody z ograniczeniem. Co ważniejsze, wyniki są porównywalne z wynikami uzyskanymi w obliczeniowo droższej metodzie najszybszego spadku.



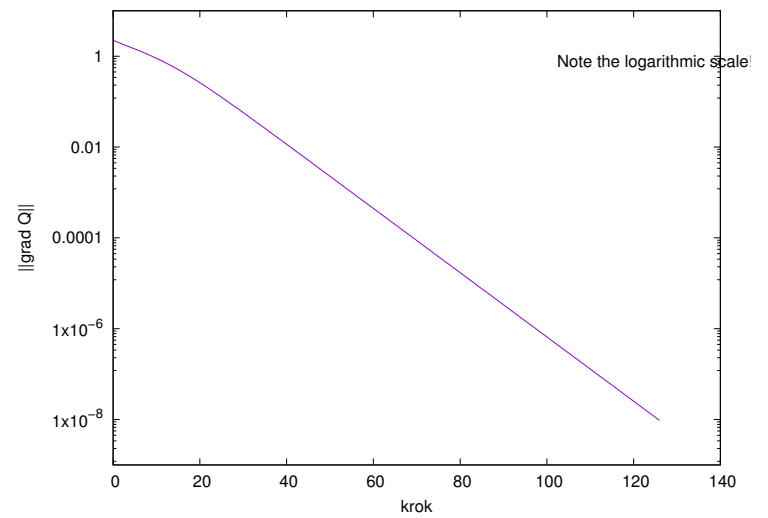
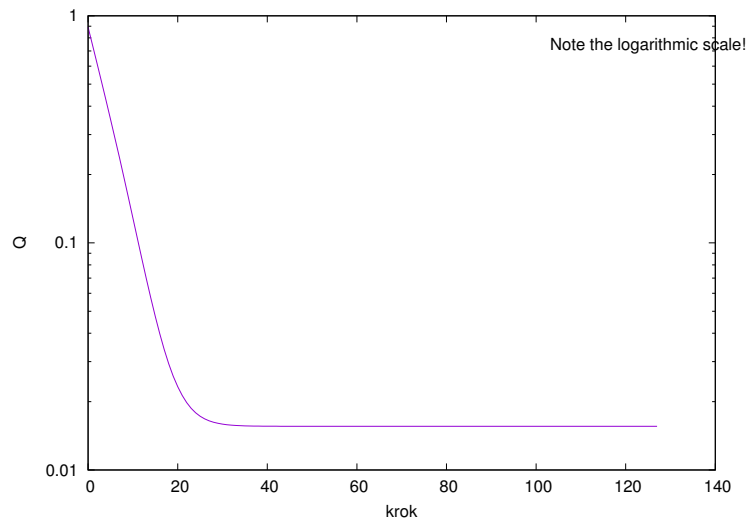
Dopasowane krzywe są niemalże nieodróżnialne w obszarze odpowiadającym danym wejściowym (zawartości zbioru uczącego).

Przykład

Wygenerowano $N = 1024$ punktów $y_i = (ae^{-x_i} + be^{x_i}) / (a + b) + \sigma\xi_i$, gdzie, jak wyżej, $\{\xi\}_{i=1}^N$ są liczbami o rozkładzie normalnym, natomiast $x_i \in [-2, 2]$. $\sigma = 0.125$. Zagadnienie dopasowania krzywej o zadanej postaci funkcyjnej do punktów danych jest przykładem *nieliowego*[†] zagadnienia najmniejszych kwadratów.

Przyjmując $\gamma = 1/32$, przeprowadzono minimalizację (a tak naprawdę procedurę podążania w stronę minimum) metodą najszybszego spadku.

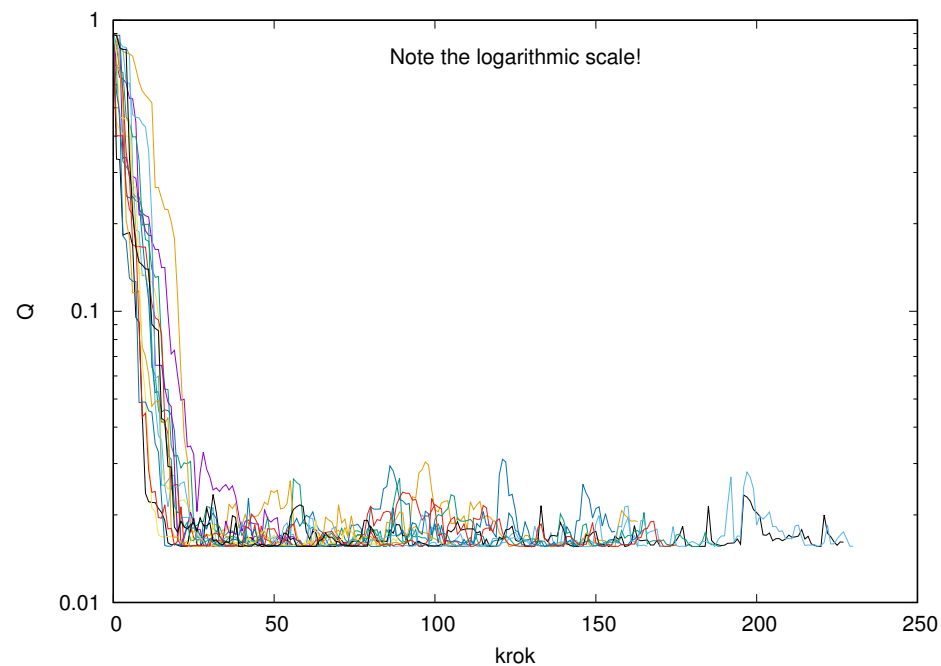
[†]Jest to słaba nieliniowość, model $y = \alpha e^{-x} + \beta e^x$, liniowy w parametrach, dałby praktycznie te same wyniki. 4



Stochastic Gradient Descent with Resetting

Aby uniknąć problemów wynikających, z jednej strony, z przypadkowego przedwczesnego zatrzymaniem procedury Stochastic Gradient Descent (SDG), z drugiej, z faktu, że trajektoria stochastyczna może zacząć *oddalać* się od minimum, powtarzamy SGD wielokrotnie, zatrzymując przebieg *albo* gdy (cząstkowy) gradient spadnie poniżej progu, *albo* gdy wykonamy ustaloną z góry, lecz losową liczbę kroków; okazuje się, że liczbę kroków należy losować z rozkładu Poissona. Po zatrzymaniu, proces *restartujemy* z tymi samymi warunkami początkowymi. Zapamiętujemy parametry końcowe osiągnięte w każdej trajektorii. Na koniec wybieramy parametry odpowiadające najmniejszej wartości Q .

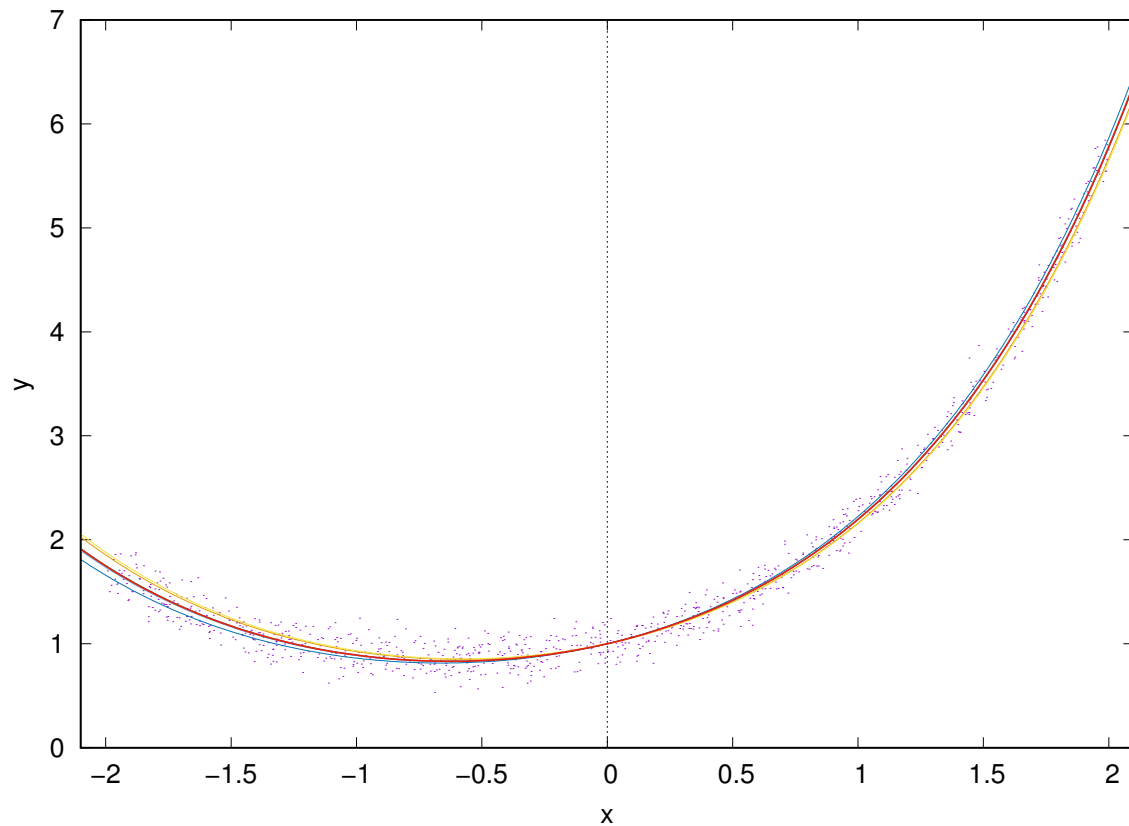
Dla opisanego wyżej problemu przykładowe trajektorie wyglądają tak:



Widać, że **początkowo** na każdej trajektorii wartości funkcji Q opadają, po czym mniej-więcej osiągnają plateau, od którego fluktuacyjnie się odchylają.

Podkreślam, że wartości Q są obliczane wzdłuż trajektorii tylko dla celów ilustracyjnych. Mając N punktów danych, zastosowanie **pełnej** metody najszyszego spadku wymaga w każdym kroku obliczenie N przyczynków do gradientu, N przyczynków do Q oraz wysumowania ich: jeden krok ma zatem złożoność $O(N)$, co może być kosztowne dla $N \gg 1$, a taka sytuacja zachodzi w przypadku *Big Data*. Przy zastosowaniu Stochastic Gradient Descent w każdym kroku obliczamy tylko jeden przyczynek, a wartość funkcji Q oblicza się tylko raz, po zakończeniu każdej trajektorii.

Co ciekawe, w omawianym przykładzie dopasowane krzywe *niezbyt* różnią się od siebie w obszarze, w którym dostępne są dane (ekstrapolacja poza ten obszar może spowodować znaczne różnice).



Przykład ten pokazuje, że metoda najszybszego spadku dobrze działa *daleko* od minimum, natomiast jeśli zależy nam na precyzyjnej lokalizacji minimum, metoda ta staje się nieefektywna.

Metody najszybszego spadku warto używać do pewnego zgrubnego zlokalizowania minimum. **BARDZO** często to wystarczy! Jeśli jednak zależy nam na precyzyjnej lokalizacji minimum, należy użyć innych, bardziej zaawansowanych metod.