# Metody Optymalizacji: Przeszukiwanie Liniowe.

Wojciech Kotłowski

Instytut Informatyki Politechniki Poznańskiej
email: imię.nazwisko@cs.put.poznan.pl

pok. 2 (CW) tel. (61)665-2936 konsultacje: piątek 15:10-16:40
Slajdy dostępne pod adresem: `http://www.cs.put.poznan.pl/wkotlowski/to/`

30.10.2018

# Spis treści

1 Przeszukiwanie liniowe dla dowolnych funkcji

2 Przeszukiwanie liniowe dla funkcji jednomodalnych

# Spis treści

1 Przeszukiwanie liniowe dla dowolnych funkcji

2 Przeszukiwanie liniowe dla funkcji jednomodalnych

# Opis problemu

Znajdź minimum (maksimum) ciągłej funkcji jednej zmiennej $f(x)$, gdzie $x \in \mathbb{R}$.

# Opis problemu

Znajdź minimum (maksimum) ciągłej funkcji jednej zmiennej $f(x)$, gdzie $x \in \mathbb{R}$.

- Nie zakładamy nic o różniczkowalności.
- Kształt funkcji nieznany (niekoniecznie wypukła).
- Zakładamy, że znamy dziedzinę funkcji $X = [a, b]$.

# Rozwiązanie – metoda „siatki” (*grid*)

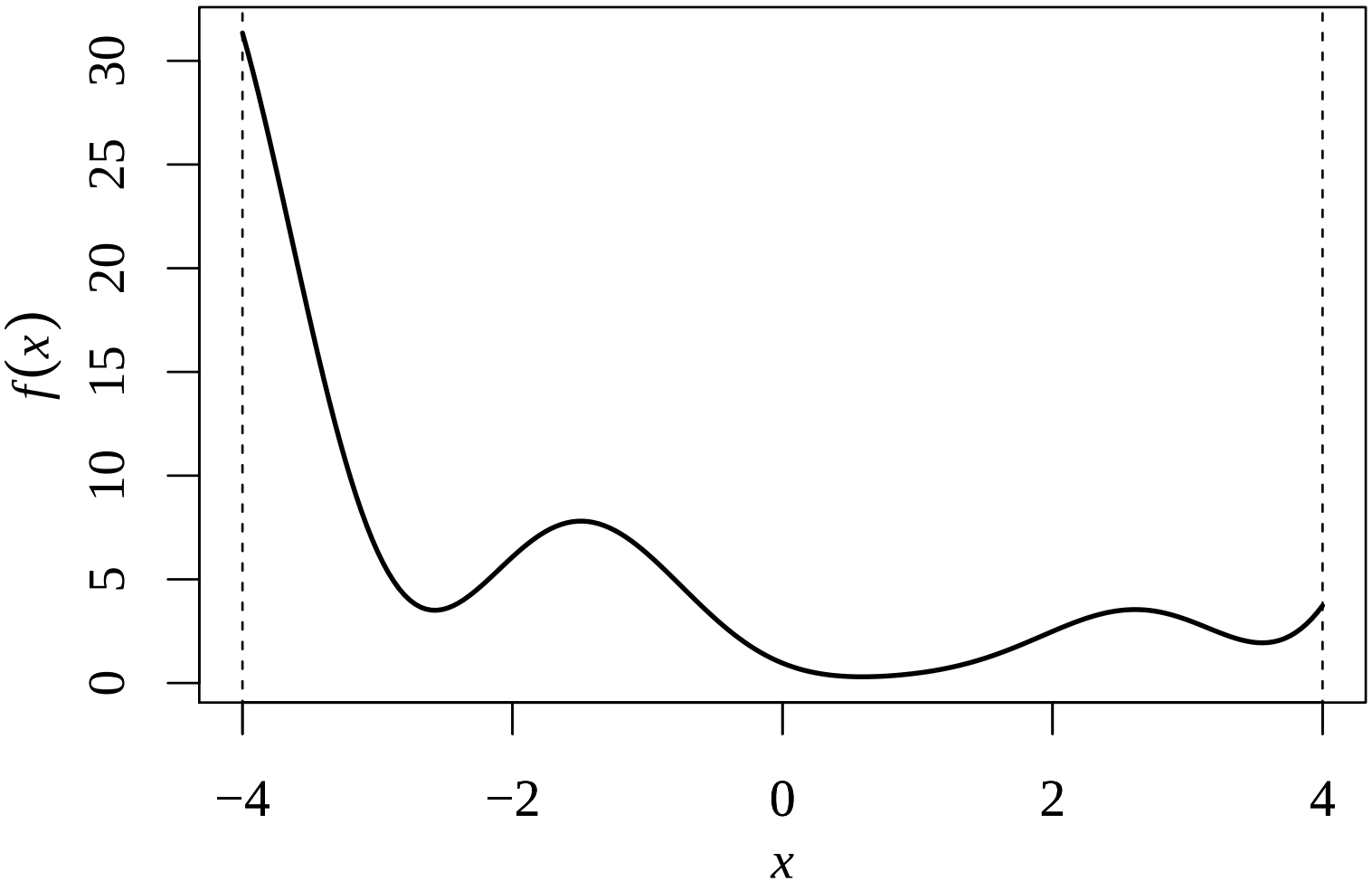
Podziel dziedzinę funkcji $X$ na bardzo wiele krótkich odcinków i sprawdź wartość funkcji w każdym z odcinków.

# Rozwiązanie – metoda „siatki” (*grid*)

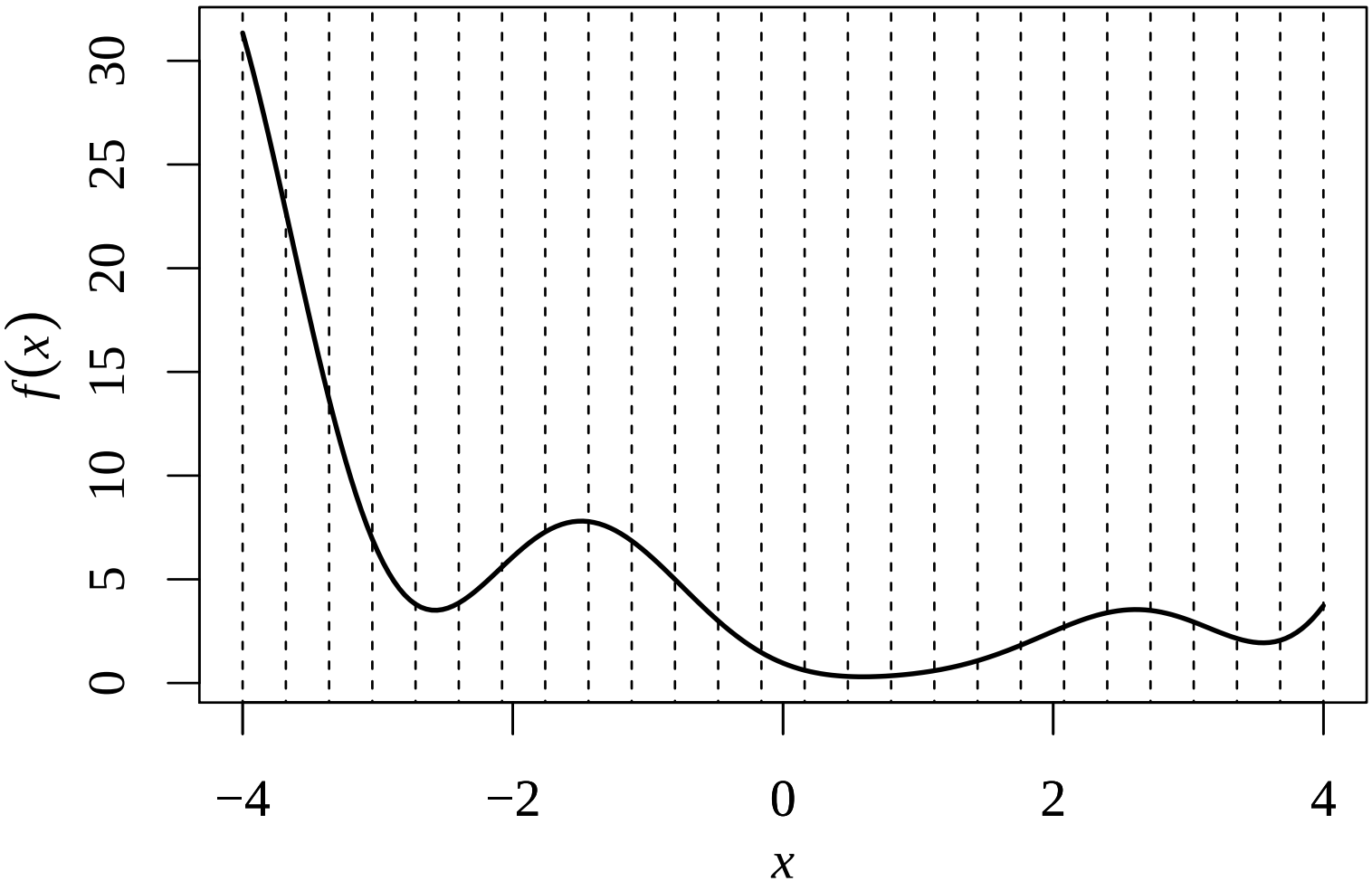
Podziel dziedzinę funkcji $X$ na bardzo wiele krótkich odcinków i sprawdź wartość funkcji w każdym z odcinków.

- Metoda o złożoności liniowej z liczbą odcinków.
- Jakość rozwiązania silnie zależy od regularności funkcji.
- W przypadku gdy chcemy mieć lepszą dokładność, możemy rekurencyjnie rozbić „najlepszy” odcinek na pododcinki.

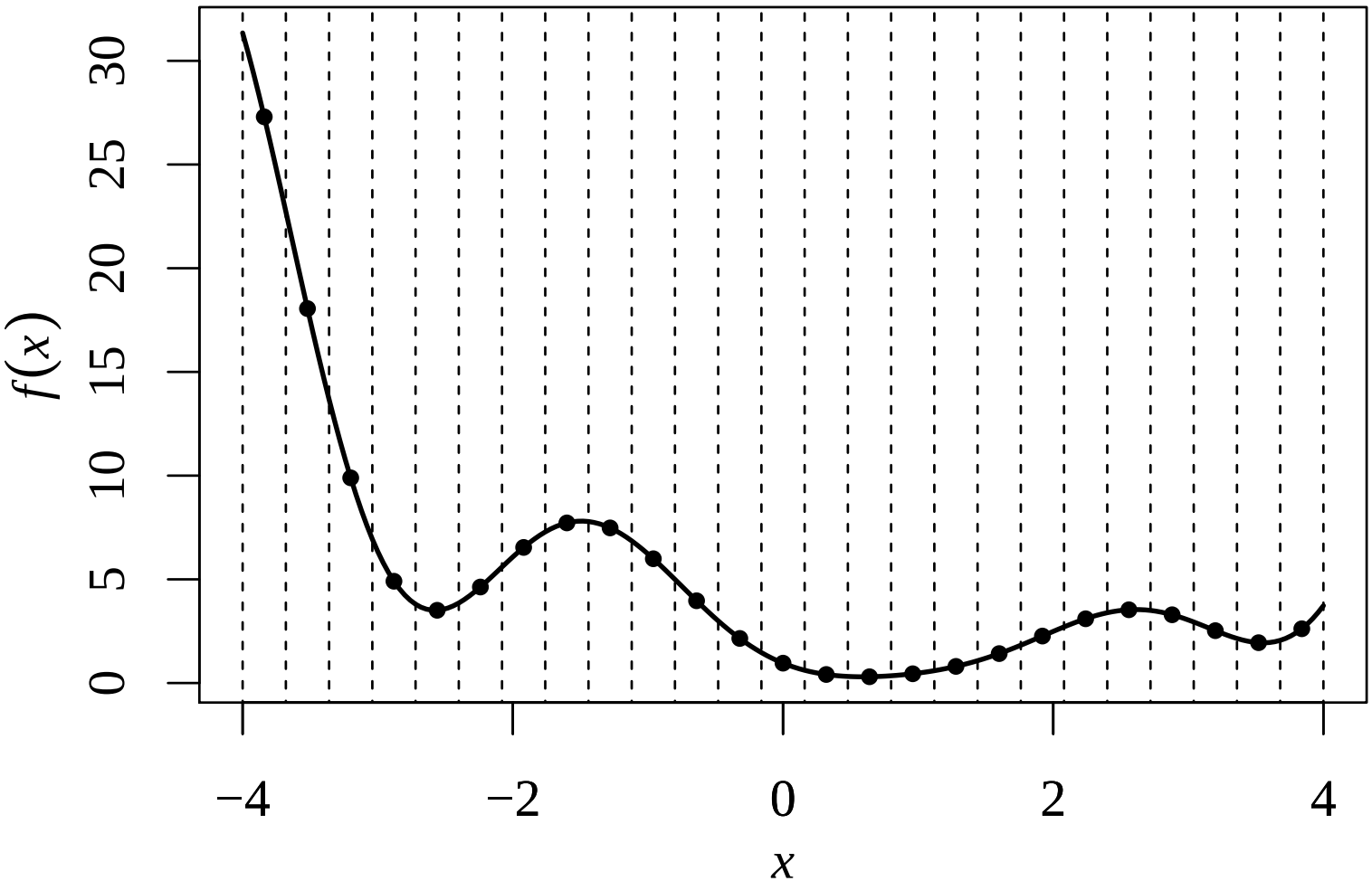
# Przeszukiwanie liniowe – przykład



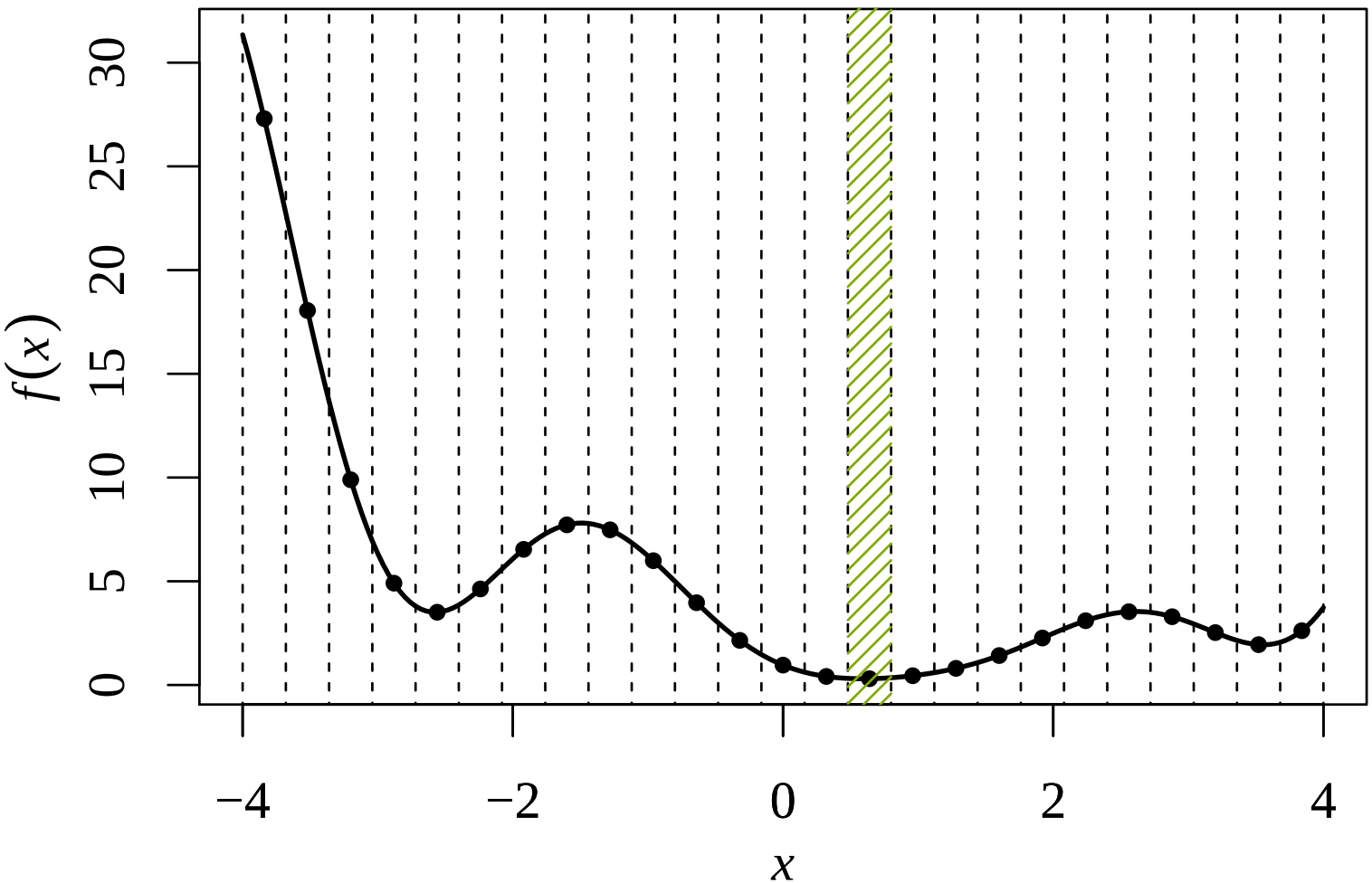
# Przeszukiwanie liniowe – przykład



# Przeszukiwanie liniowe – przykład

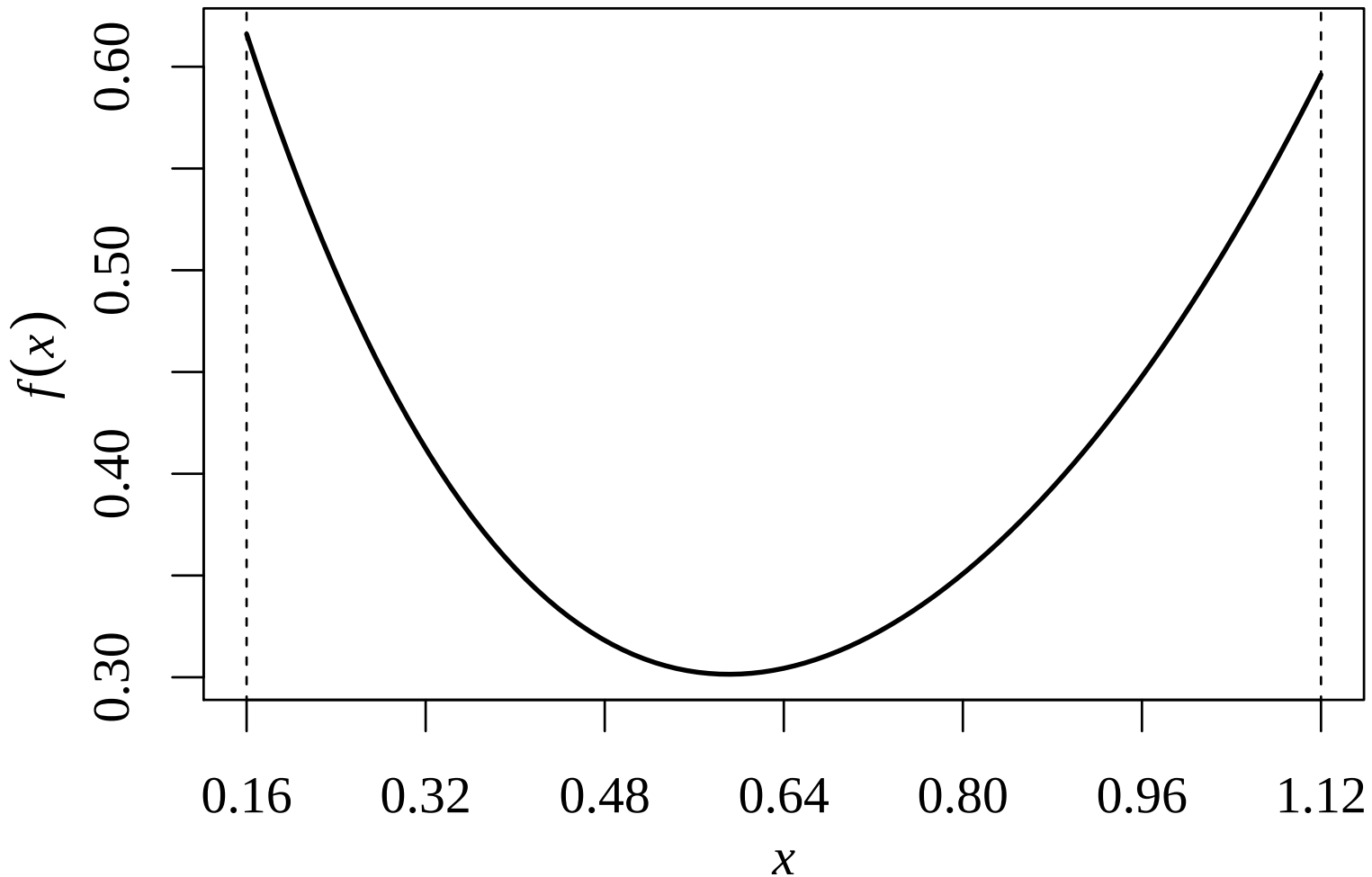


# Przeszukiwanie liniowe – przykład



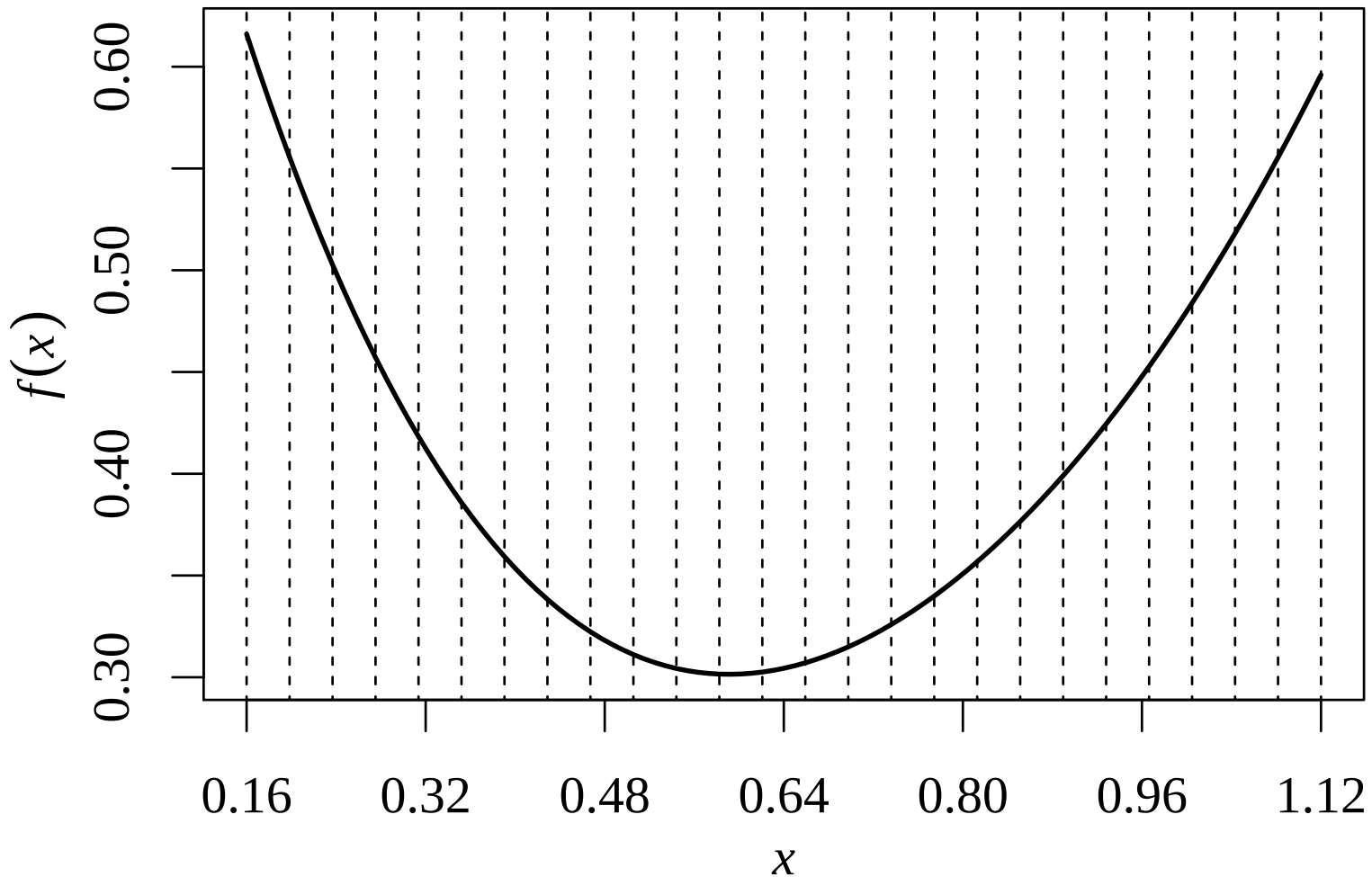
# Przeszukiwanie liniowe – przykład

Rekurencyjnie rozbijamy nalepszy odcinek i dwa wokół niego na pododcinki.



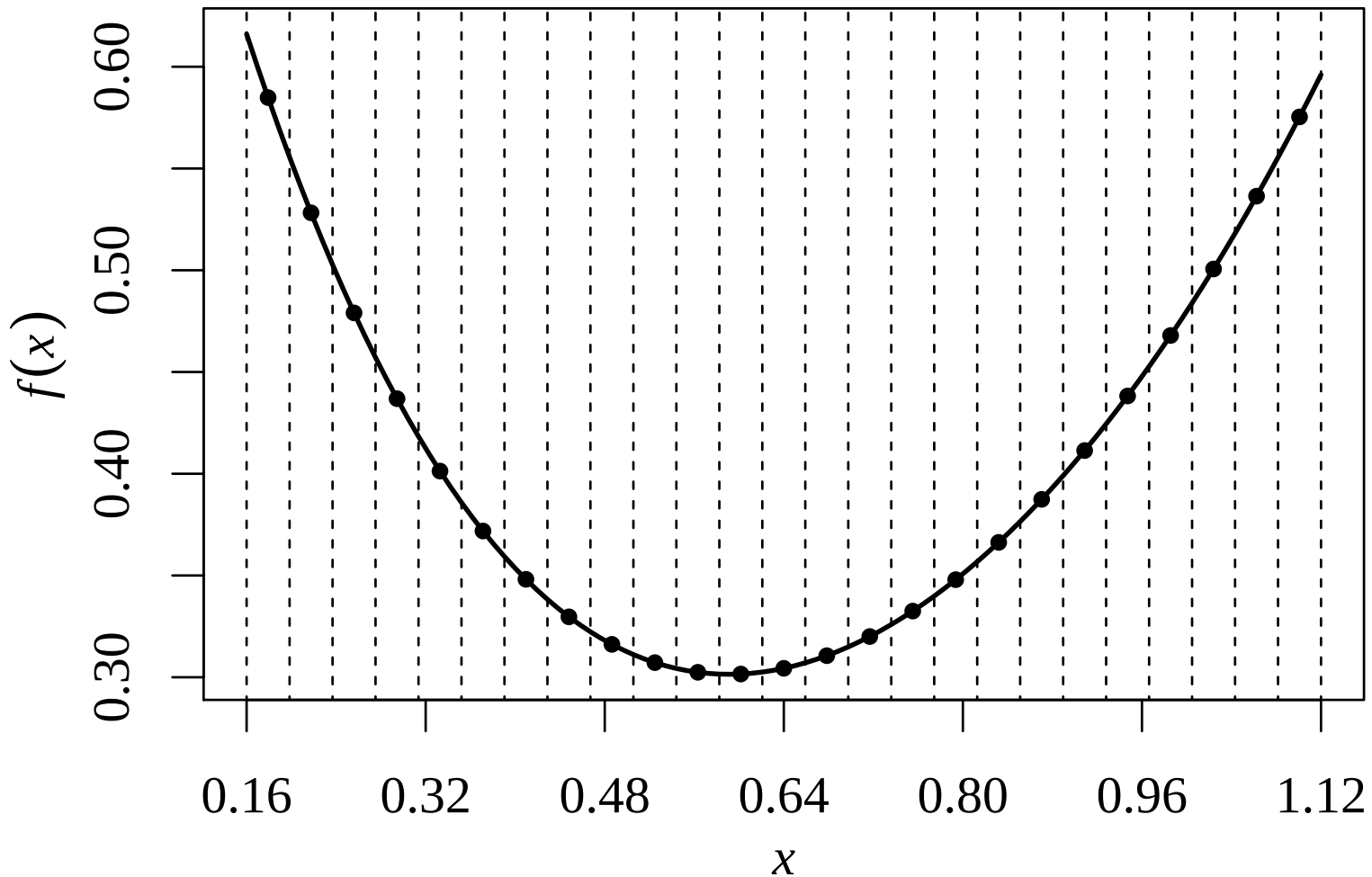
# Przeszukiwanie liniowe – przykład

Rekurencyjnie rozbijamy nalepszy odcinek i dwa wokół niego na pododcinki.



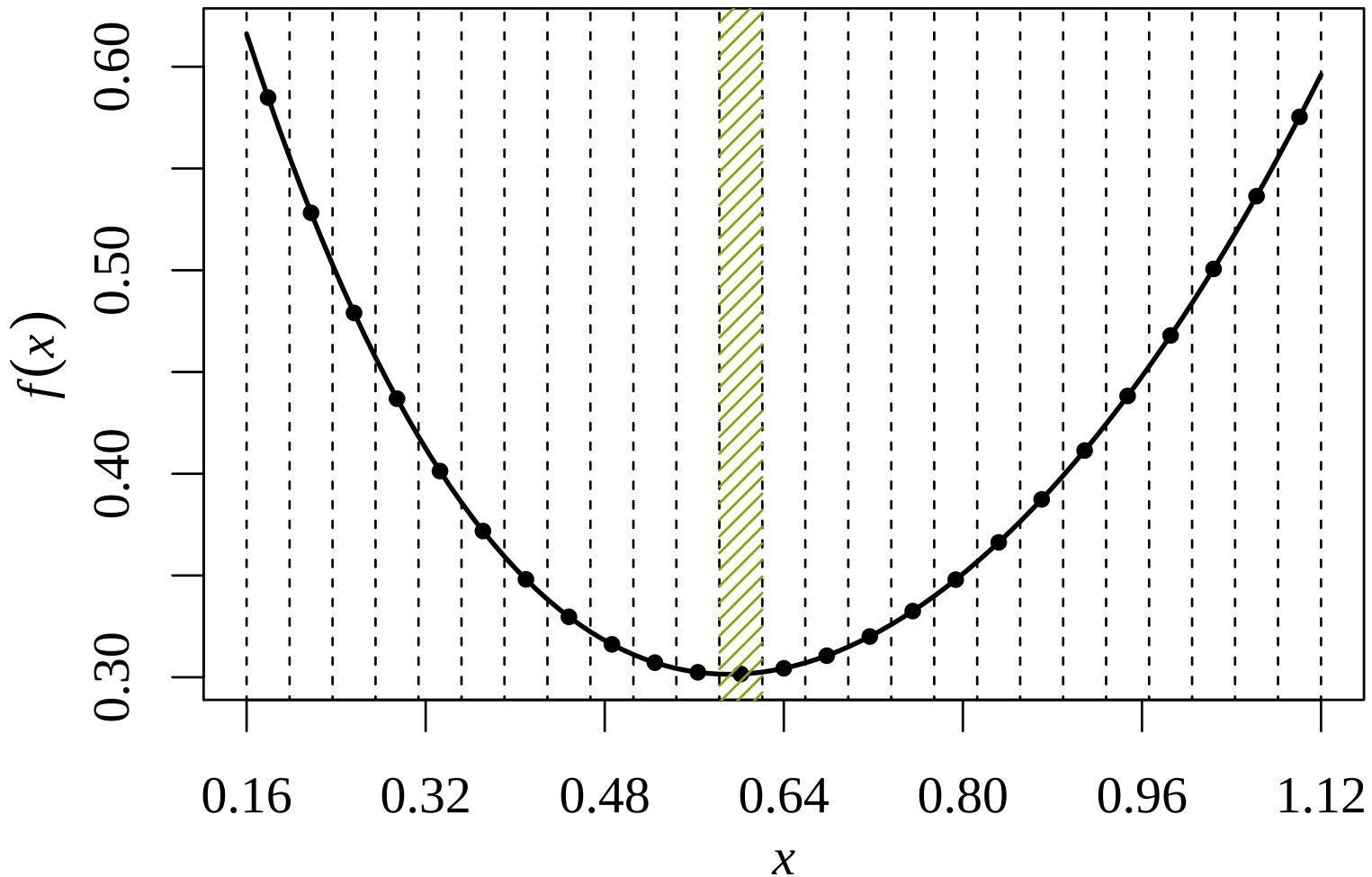
# Przeszukiwanie liniowe – przykład

Rekurencyjnie rozbijamy nalepszy odcinek i dwa wokół niego na pododcinki.

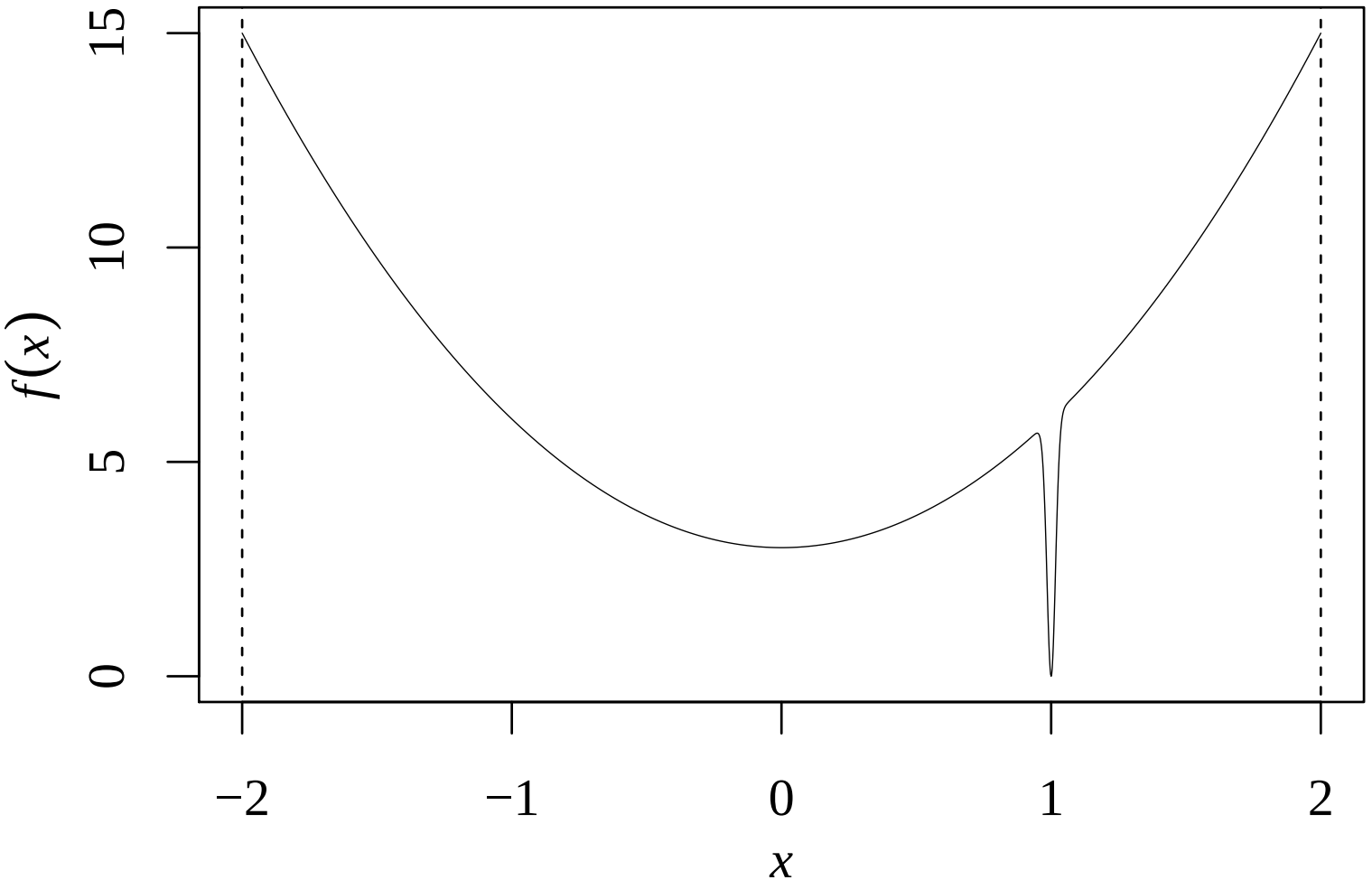


# Przeszukiwanie liniowe – przykład

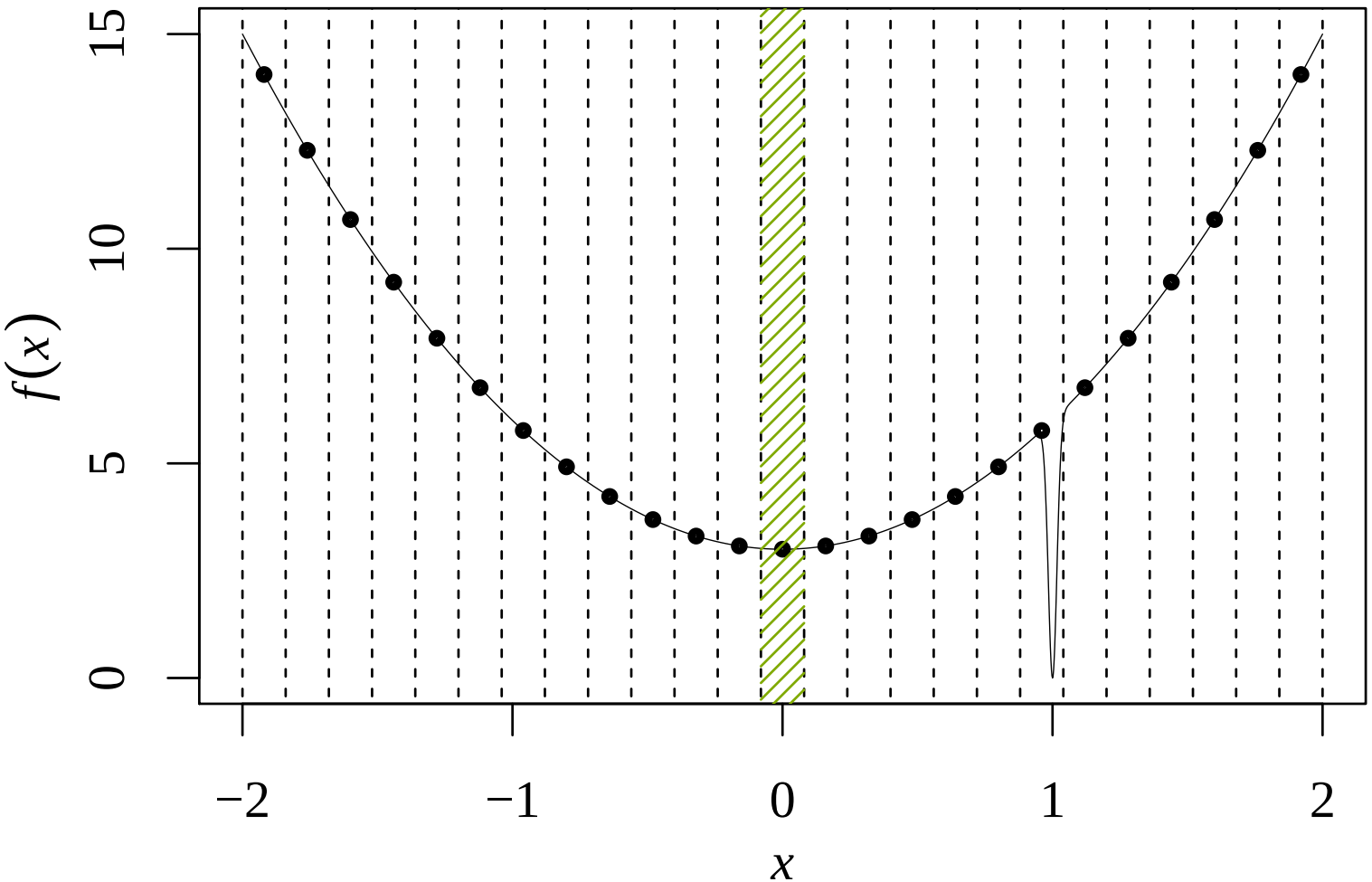
Rekurencyjnie rozbijamy nalepszy odcinek i dwa wokół niego na pododcinki.



# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?



# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?



# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?

- Nie!

# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?

- Nie!
- Jakość rozwiązania zależy od tzw. stałej Lipschitza.

# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?

- Nie!
- Jakość rozwiązania zależy od tzw. stałej Lipschitza.
- Jest to najmniejsza stała $L$ taka, że dla dowolnych $x_1, x_2 \in X$:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|.$$

Stała Lipschitza daje górne ograniczenie na szybkość zmian funkcji (wzrostu lub spadku).

# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?

- Nie!
- Jakość rozwiązania zależy od tzw. stałej Lipschitza.
- Jest to najmniejsza stała $L$ taka, że dla dowolnych $x_1, x_2 \in X$:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|.$$

  Stała Lipschitza daje górne ograniczenie na szybkość zmian funkcji (wzrostu lub spadku).
- Jeśli podzielimy dziedzinę na odcinki o długości $\epsilon$, to dla $x_1, x_2$ z tego samego odcinka, $|x_1 - x_2| \leq \epsilon$, a stąd $|f(x_1) - f(x_2)| \leq L\epsilon$.

# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?

- Nie!
- Jakość rozwiązania zależy od tzw. stałej Lipschitza.
- Jest to najmniejsza stała $L$ taka, że dla dowolnych $x_1, x_2 \in X$:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|.$$

  Stała Lipschitza daje górne ograniczenie na szybkość zmian funkcji (wzrostu lub spadku).
- Jeśli podzielimy dziedzinę na odcinki o długości $\epsilon$, to dla $x_1, x_2$ z tego samego odcinka, $|x_1 - x_2| \leq \epsilon$, a stąd $|f(x_1) - f(x_2)| \leq L\epsilon$.
- Metoda przeszukiwania liniowego gwarantuje rozwiązanie nie gorsze niż $L\epsilon$ od optymalnego!

# Czy przeszukiwanie liniowe gwarantuje znalezienie minimum?

- Nie!
- Jakość rozwiązania zależy od tzw. stałej Lipschitza.
- Jest to najmniejsza stała $L$ taka, że dla dowolnych $x_1, x_2 \in X$:

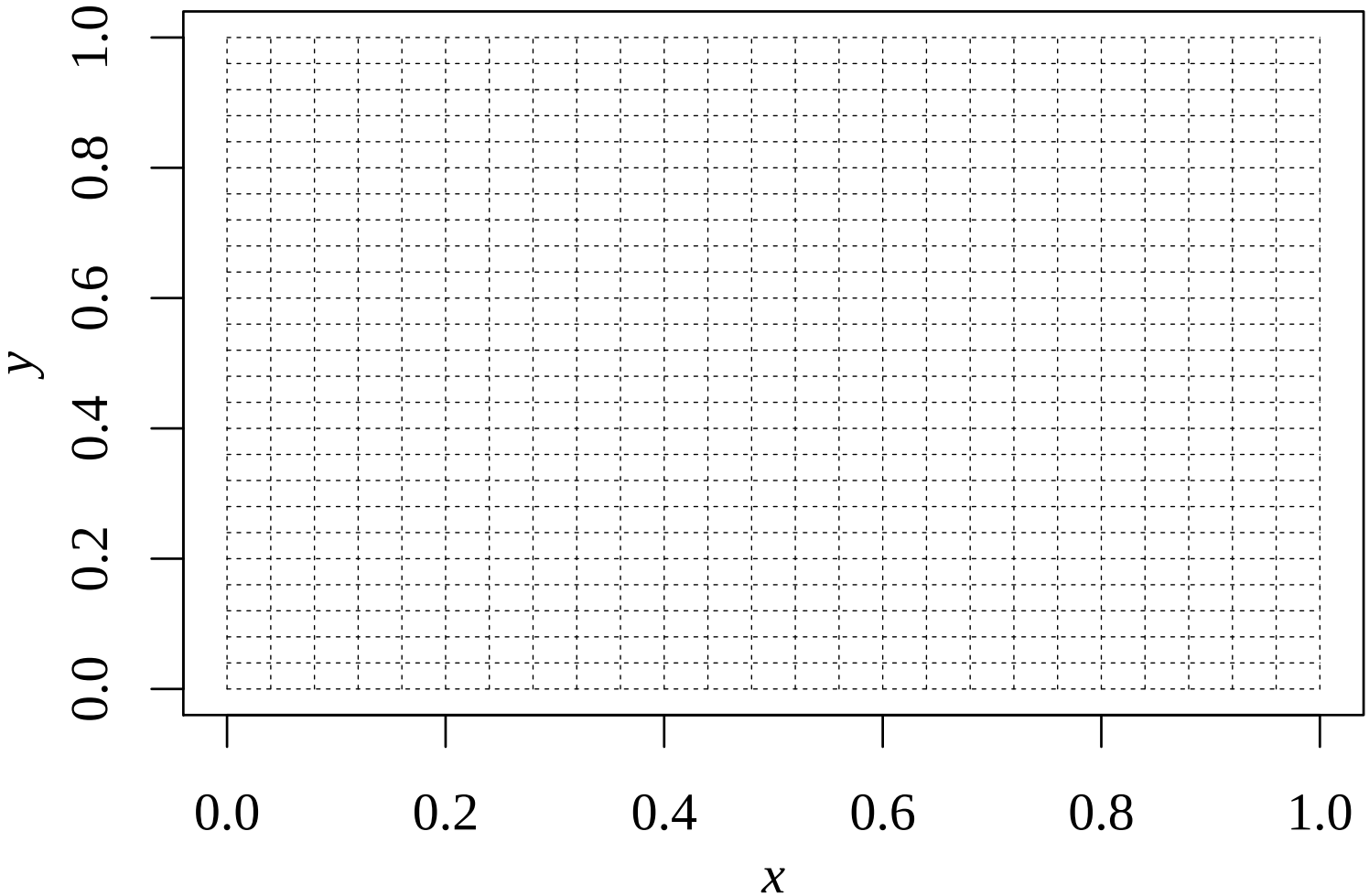
$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|.$$

  Stała Lipschitza daje górne ograniczenie na szybkość zmian funkcji (wzrostu lub spadku).
- Jeśli podzielimy dziedzinę na odcinki o długości $\epsilon$, to dla $x_1, x_2$ z tego samego odcinka, $|x_1 - x_2| \leq \epsilon$, a stąd $|f(x_1) - f(x_2)| \leq L\epsilon$.
- Metoda przeszukiwania liniowego gwarantuje rozwiązanie nie gorsze niż $L\epsilon$ od optymalnego!
- Uwaga: dla niektórych funkcji, $L$ może być bardzo duże lub nawet nieskończone!

# Funkcje wielu zmiennych

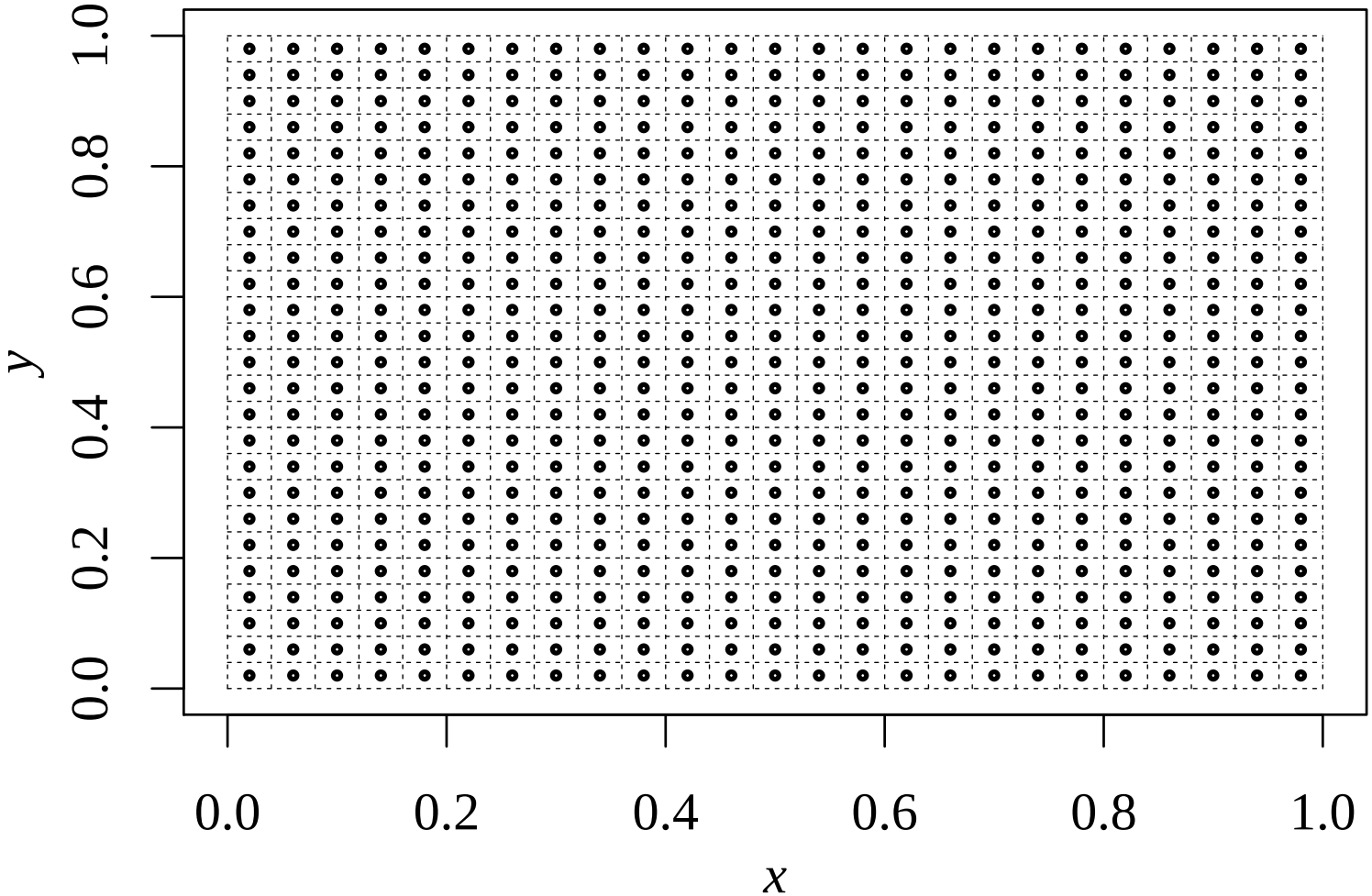
# Funkcje wielu zmiennych

Przykład dziedziny dla funkcji dwóch zmiennych $f(x, y)$



# Funkcje wielu zmiennych

Przykład dziedziny dla funkcji dwóch zmiennych $f(x, y)$



# Funkcje wielu zmiennych

- Liczba „oczek” siatki rośnie wykładniczo z liczbą zmiennych!
- Metoda efektywnie daje się zastosować do maksymalnie $3-4$ zmiennych (w zależności od gęstości siatki i zakładanej jakości rozwiązania).
- Dla większej liczby zmiennych zamiast dzielić dziedzinę na oczka siatki, lepiej losować wielokrotnie punkty z dziedziny (Monte Carlo), a następnie wybrać najlepszy punkt i rekurencyjnie losować kolejne punkty wokół niego.

# Spis treści

1 Przeszukiwanie liniowe dla dowolnych funkcji

2 Przeszukiwanie liniowe dla funkcji jednomodalnych

# Przeszukiwanie liniowe dla funkcji jednomodalnych

- Funkcja jednomodalna to taka, która ma jedno minimum/maksimum.
  $\Longrightarrow$ zakładamy minimum, ponieważ minimalizujemy.

# Przeszukiwanie liniowe dla funkcji jednomodalnych

- Funkcja jednomodalna to taka, która ma jedno minimum/maksimum.
  $\Longrightarrow$ zakładamy minimum, ponieważ minimalizujemy.
- W szczególności: funkcja wypukła jest jednomodalna.

# Przeszukiwanie liniowe dla funkcji jednomodalnych

- Funkcja jednomodalna to taka, która ma jedno minimum/maksimum.
  $\Longrightarrow$ zakładamy minimum, ponieważ minimalizujemy.
- W szczególności: funkcja wypukła jest jednomodalna.
- Dla funkcji jednomodalnej istnieje algorytm gwarantujący zbieżność do minimum.

# Przeszukiwanie liniowe dla funkcji jednomodalnych

- Funkcja jednomodalna to taka, która ma jedno minimum/maksimum.
  $\Longrightarrow$ zakładamy minimum, ponieważ minimalizujemy.
- W szczególności: funkcja wypukła jest jednomodalna.
- Dla funkcji jednomodalnej istnieje algorytm gwarantujący zbieżność do minimum.
- Algorytm zbiega wykładniczo szybko: jeśli dziedziną funkcji jest odcinek o długości $M$, to po $n$ odwołaniach do funkcji, minimum zostanie zlokalizowane wewnątrz odcinka o długości $M\alpha^n$, dla stałej $\alpha < 1$.

# Przeszukiwanie liniowe dla funkcji jednomodalnych

- Funkcja jednomodalna to taka, która ma jedno minimum/maksimum.
  $\Longrightarrow$ zakładamy minimum, ponieważ minimalizujemy.
- W szczególności: funkcja wypukła jest jednomodalna.
- Dla funkcji jednomodalnej istnieje algorytm gwarantujący zbieżność do minimum.
- Algorytm zbiega wykładniczo szybko: jeśli dziedziną funkcji jest odcinek o długości $M$, to po $n$ odwołaniach do funkcji, minimum zostanie zlokalizowane wewnątrz odcinka o długości $M\alpha^n$, dla stałej $\alpha < 1$.
  Przykład: dla $M = 1$, $\alpha = 1/\sqrt{2}$:

$n = 2$ minimum zlokalizowane z dokładnością $\pm 0.5$

$n = 6$ minimum zlokalizowane z dokładnością $\pm 0.125$

$n = 10$ minimum zlokalizowane z dokładnością $\pm 0.03125$

$n = 60$ minimum zlokalizowane z dokładnością $\pm 2^{-30} \simeq 10^{-9}$

# Idea algorytmu

## Twierdzenie

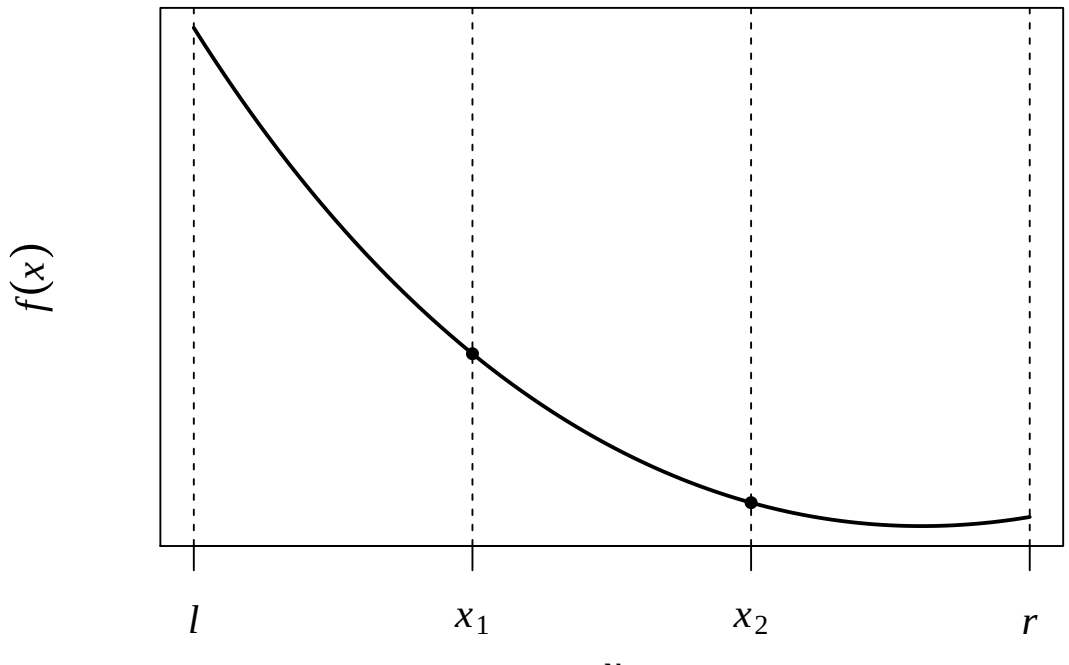
Niech funkcja $f(x)$ posiada jedno (ścisłe) minimum na przedziale $[l, r]$. Jeśli dla dowolnych $x_1, x_2 \in [l, r]$, takich, że $x_1 < x_2$, zachodzi $f(x_1) \geq f(x_2)$, to $f(x) \geq f(x_2)$ dla wszystkich $x \in [l, x_1]$. Podobnie, jeśli $f(x_1) \leq f(x_2)$, to $f(x) \geq f(x_1)$ dla wszystkich $x \in [x_2, r]$.

# Idea algorytmu

## Twierdzenie

Niech funkcja $f(x)$ posiada jedno (ścisłe) minimum na przedziale $[l, r]$. Jeśli dla dowolnych $x_1, x_2 \in [l, r]$, takich, że $x_1 < x_2$, zachodzi $f(x_1) \geq f(x_2)$, to $f(x) \geq f(x_2)$ dla wszystkich $x \in [l, x_1]$. Podobnie, jeśli $f(x_1) \leq f(x_2)$, to $f(x) \geq f(x_1)$ dla wszystkich $x \in [x_2, r]$.

Dowód: trzy przypadki, w zależności od lokalizacji minimum.

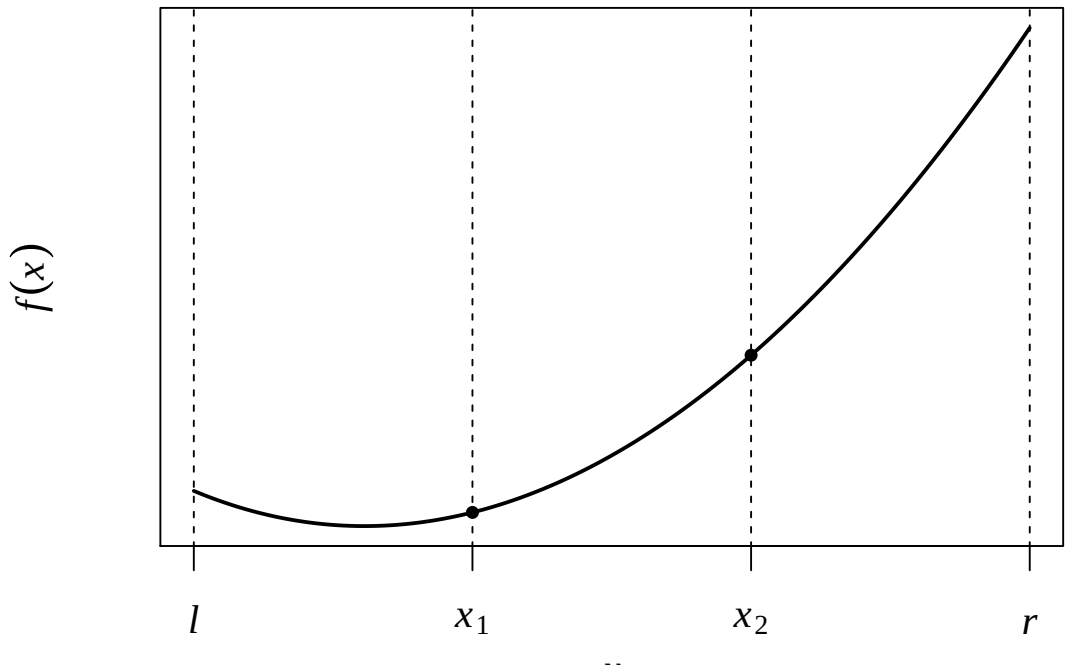


# Idea algorytmu

## Twierdzenie

Niech funkcja $f(x)$ posiada jedno (ścisłe) minimum na przedziale $[l, r]$. Jeśli dla dowolnych $x_1, x_2 \in [l, r]$, takich, że $x_1 < x_2$, zachodzi $f(x_1) \geq f(x_2)$, to $f(x) \geq f(x_2)$ dla wszystkich $x \in [l, x_1]$. Podobnie, jeśli $f(x_1) \leq f(x_2)$, to $f(x) \geq f(x_1)$ dla wszystkich $x \in [x_2, r]$.

Dowód: trzy przypadki, w zależności od lokalizacji minimum.

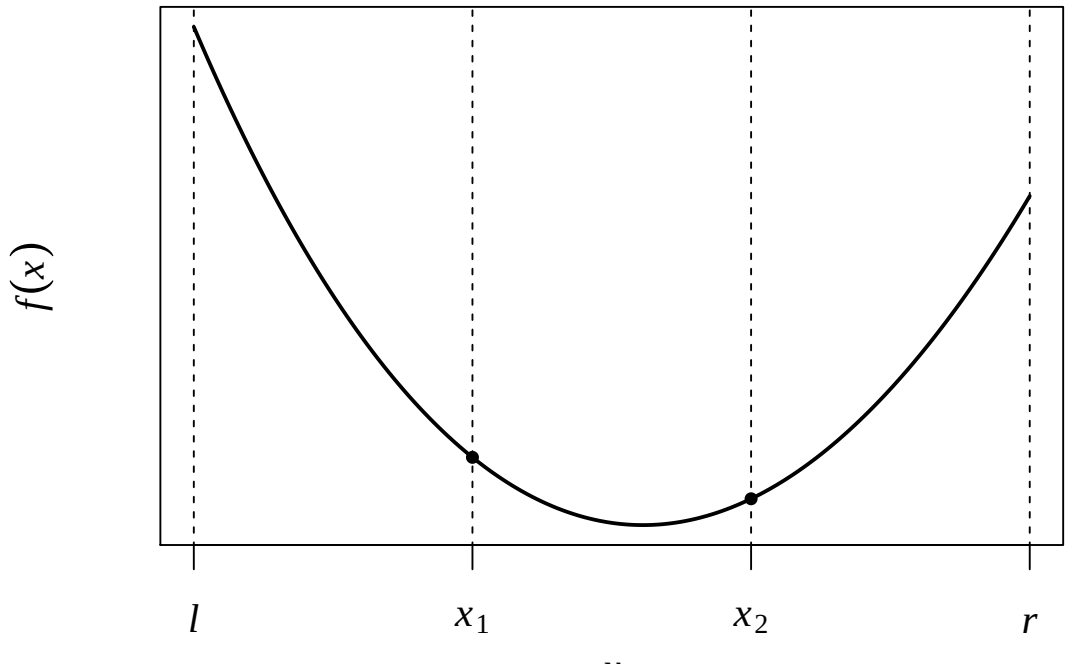


# Idea algorytmu

## Twierdzenie

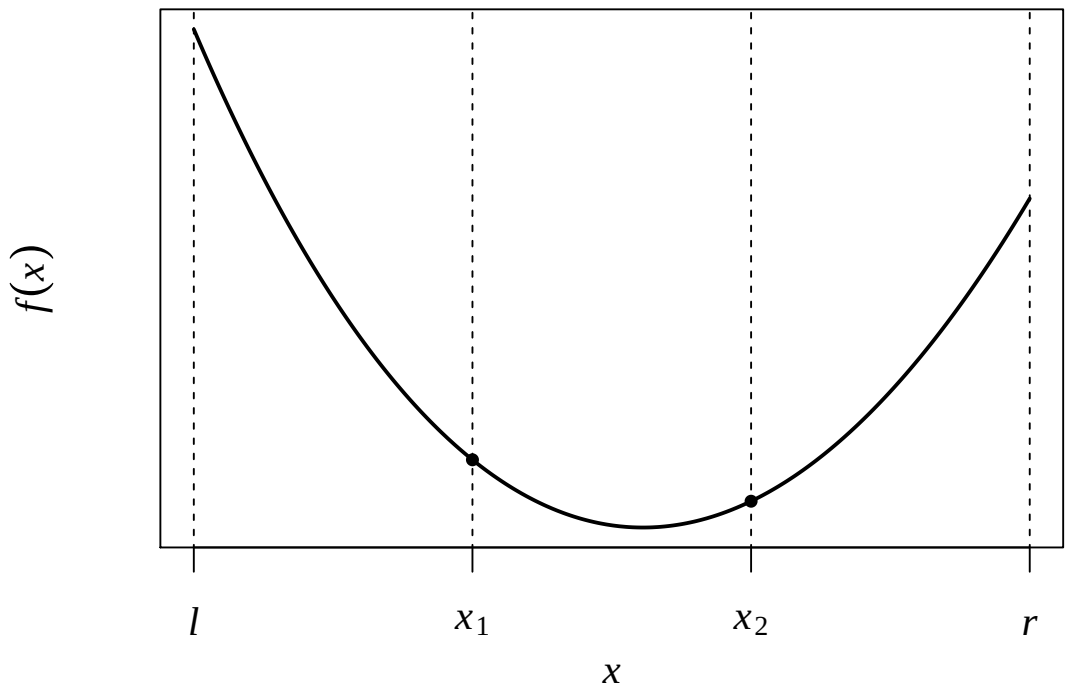
Niech funkcja $f(x)$ posiada jedno (ścisłe) minimum na przedziale $[l, r]$. Jeśli dla dowolnych $x_1, x_2 \in [l, r]$, takich, że $x_1 < x_2$, zachodzi $f(x_1) \geq f(x_2)$, to $f(x) \geq f(x_2)$ dla wszystkich $x \in [l, x_1]$. Podobnie, jeśli $f(x_1) \leq f(x_2)$, to $f(x) \geq f(x_1)$ dla wszystkich $x \in [x_2, r]$.

Dowód: trzy przypadki, w zależności od lokalizacji minimum.



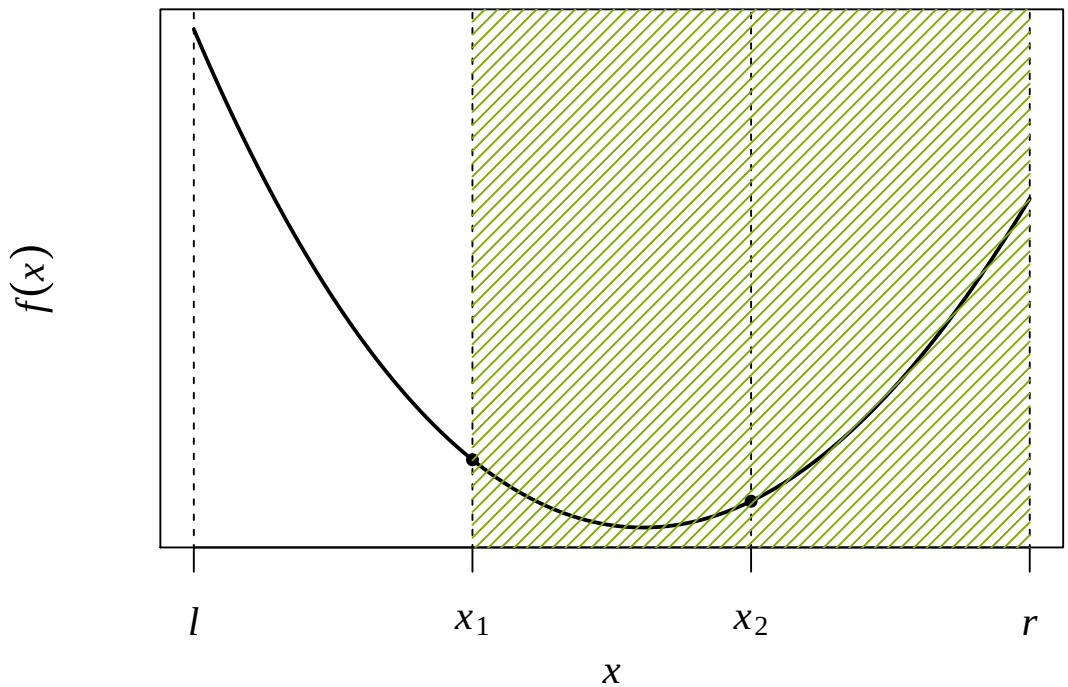
# Idea algorytmu

Każda iteracja zmniejsza zakres przeszukiwania z $[l, r]$ do jednego z dwóch przedziałów: $[l, x_2]$ lub $[x_1, r]$.



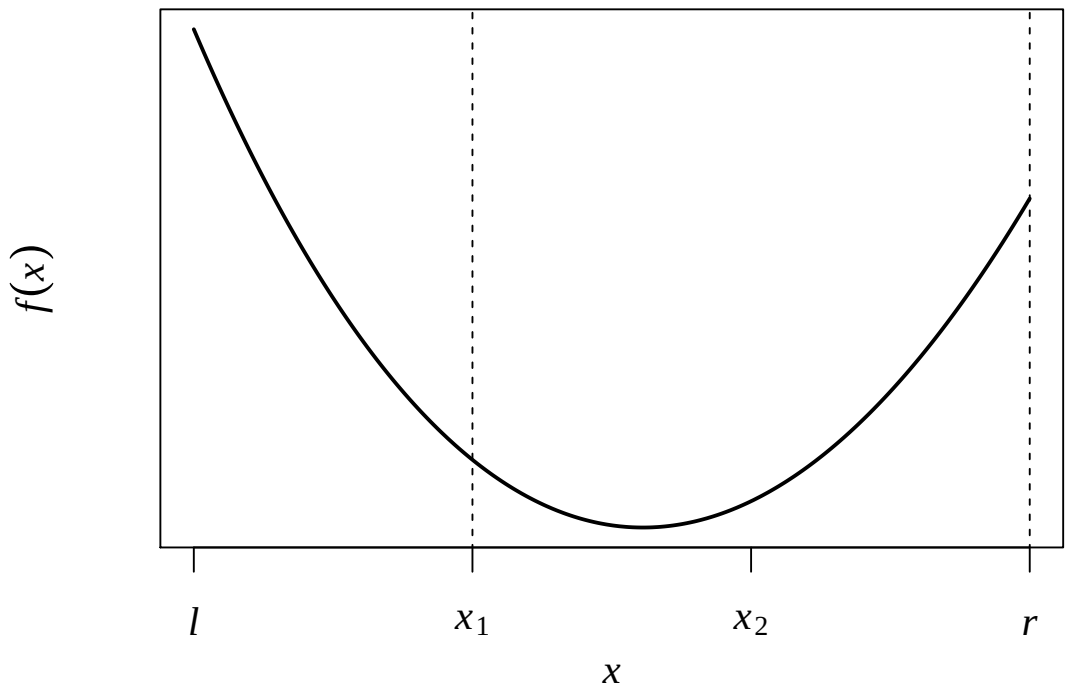
# Idea algorytmu

Każda iteracja zmniejsza zakres przeszukiwania z $[l, r]$ do jednego z dwóch przedziałów: $[l, x_2]$ lub $[x_1, r]$.



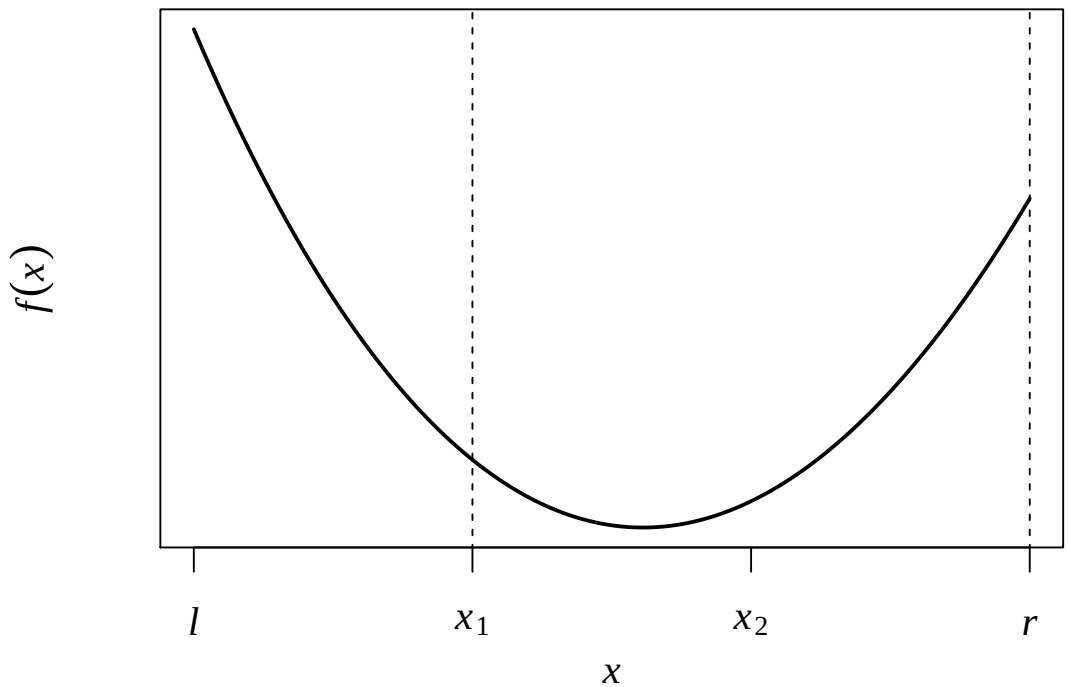
# Idea algorytmu

Każda iteracja zmniejsza zakres przeszukiwania z $[l, r]$ do jednego z dwóch przedziałów: $[l, x_2]$ lub $[x_1, r]$.



# Idea algorytmu

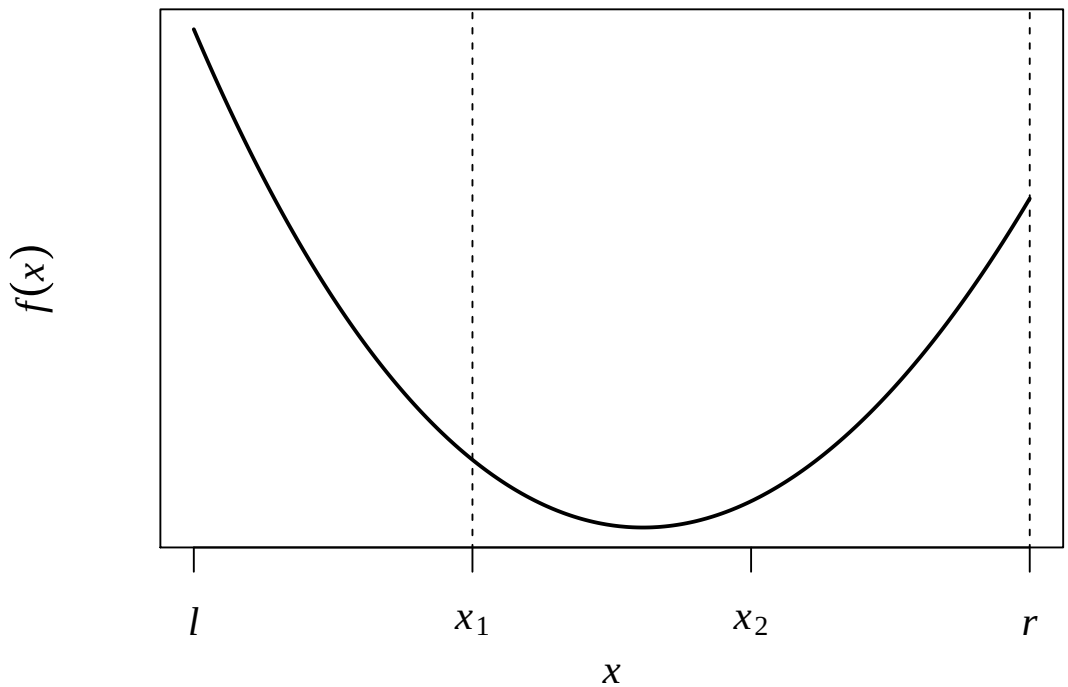
Każda iteracja zmniejsza zakres przeszukiwania z $[l, r]$ do jednego z dwóch przedziałów: $[l, x_2]$ lub $[x_1, r]$.




Jak dobrać $x_1$ i $x_2$, aby maksymalnie zawęzić zakres przeszukiwań?

# Idea algorytmu

Każda iteracja zmniejsza zakres przeszukiwania z $[l, r]$ do jednego z dwóch przedziałów: $[l, x_2]$ lub $[x_1, r]$.




Jak dobrać $x_1$ i $x_2$, aby maksymalnie zawęzić zakres przeszukiwań?
$\Longrightarrow$ punkt $x_1$ i $x_2$ w środku, jak najbliżej siebie!

# Algorytm I: bisekcja (*dichotomous search*)

Wejście: Procedura wyznaczająca wartość funkcji $f(x)$ w dowolnym punkcie dziedziny $X = [a, b]$; liczba odwołań do funkcji $n$; stała $\epsilon$ mniejsza od oczekiwanej dokładności (np. $\epsilon = 10^{-10}$)

Wyjście: Punkt $x^*$ odległy od minimum funkcji $f$ o nie więcej niż $(b-a) \cdot \left(\frac{1}{2} + \epsilon\right)^{n/2}$.

1. Przypisz $\ell = a$, $r = b$.
2. Powtarzaj dla $i = 1, 2, \ldots, n/2$:
   1. Wyznacz punkt środkowy $m = \frac{l+r}{2}$.
   2. Wyznacz wartości funkcji punktach $m - \epsilon$ i $m + \epsilon$.
   3. Jeśli $f(m - \epsilon) < f(m + \epsilon)$ to $r = m + \epsilon$, w przeciwnym przypadku $\ell = m - \epsilon$.
3. Jako wynik zwróć punkt $x^* = \frac{\ell+r}{2}$.

# Algorytm I: bisekcja (*dichotomous search*)

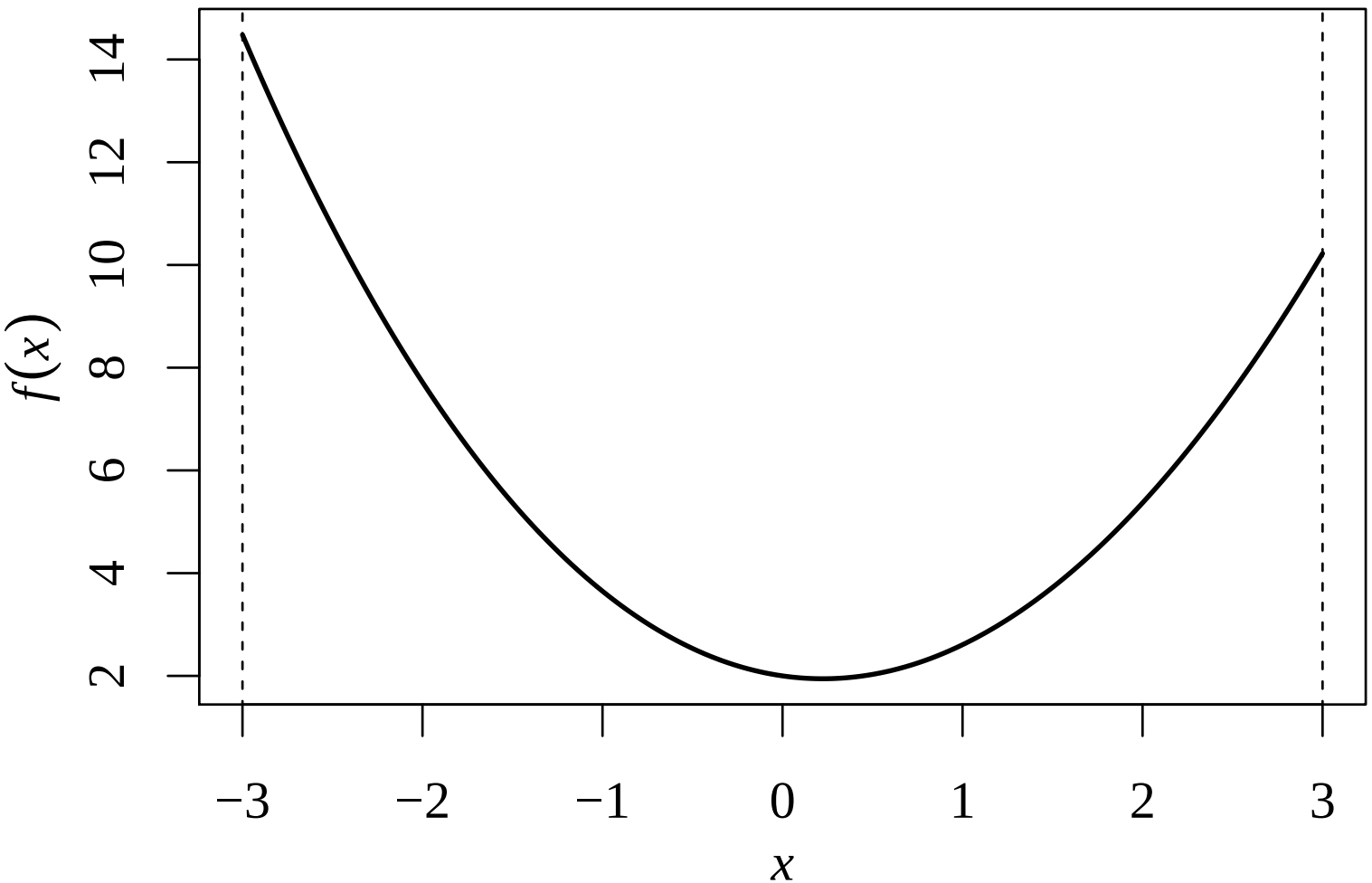
Wejście: Procedura wyznaczająca wartość funkcji $f(x)$ w dowolnym punkcie dziedziny $X = [a, b]$; liczba odwołań do funkcji $n$; stała $\epsilon$ mniejsza od oczekiwanej dokładności (np. $\epsilon = 10^{-10}$)

Wyjście: Punkt $x^*$ odległy od minimum funkcji $f$ o nie więcej niż $(b-a) \cdot \left(\frac{1}{2} + \epsilon\right)^{n/2}$.

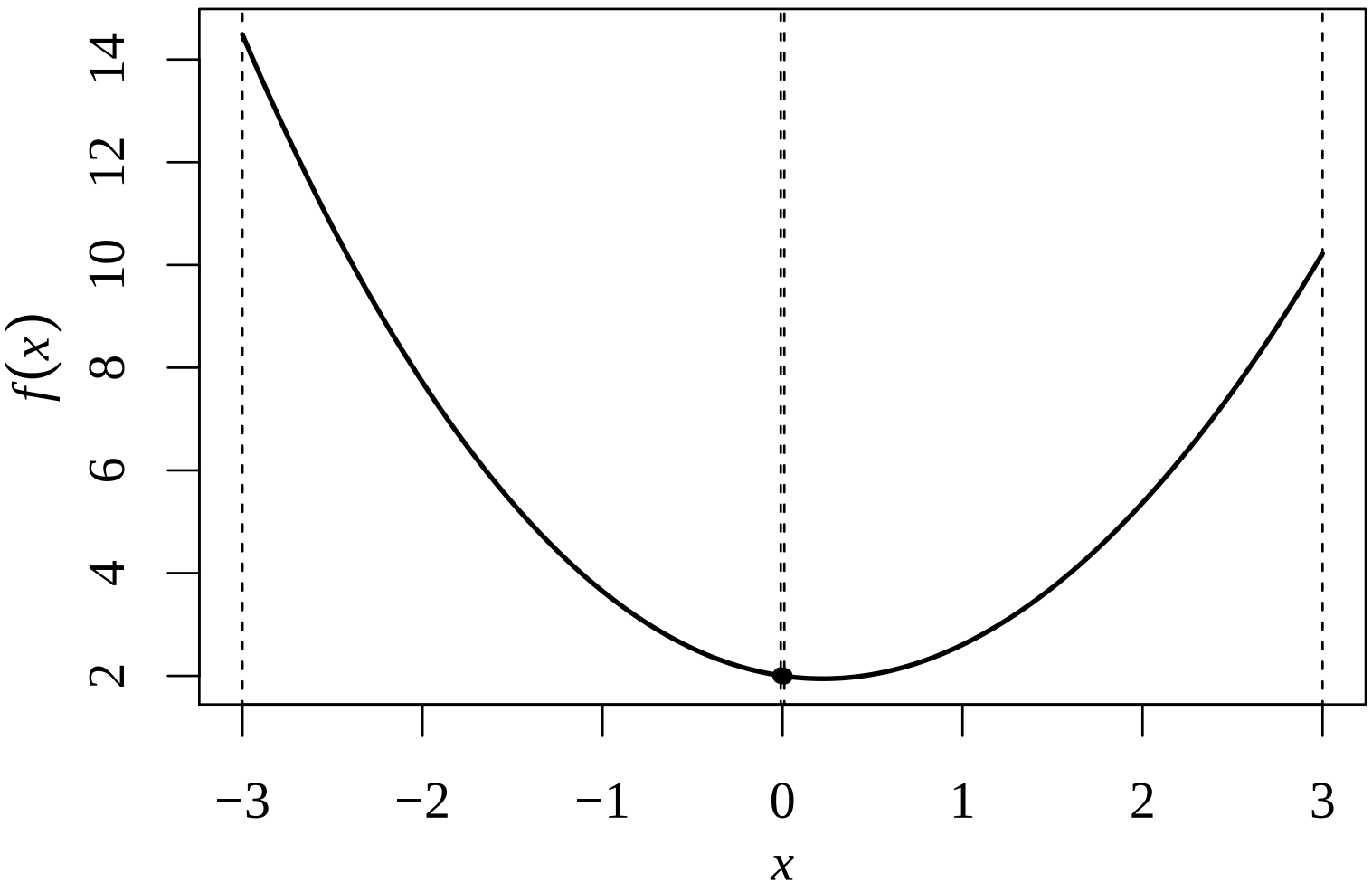
1. Przypisz $\ell = a$, $r = b$.
2. Powtarzaj dla $i = 1, 2, \ldots, n/2$: $\Leftarrow$ po 2 odwołania na iter.
   1. Wyznacz punkt środkowy $m = \frac{l+r}{2}$.
   2. Wyznacz wartości funkcji punktach $m - \epsilon$ i $m + \epsilon$.
   3. Jeśli $f(m-\epsilon) < f(m+\epsilon)$ to $r = m + \epsilon$, w przeciwnym przypadku $\ell = m - \epsilon$.
3. Jako wynik zwróć punkt $x^* = \frac{\ell+r}{2}$.

Gwarancja na odległość od optimum: $(b-a) \cdot \left(\frac{1}{2} + \epsilon\right)^{n/2} \simeq \frac{b-a}{\sqrt{2}^n}$.

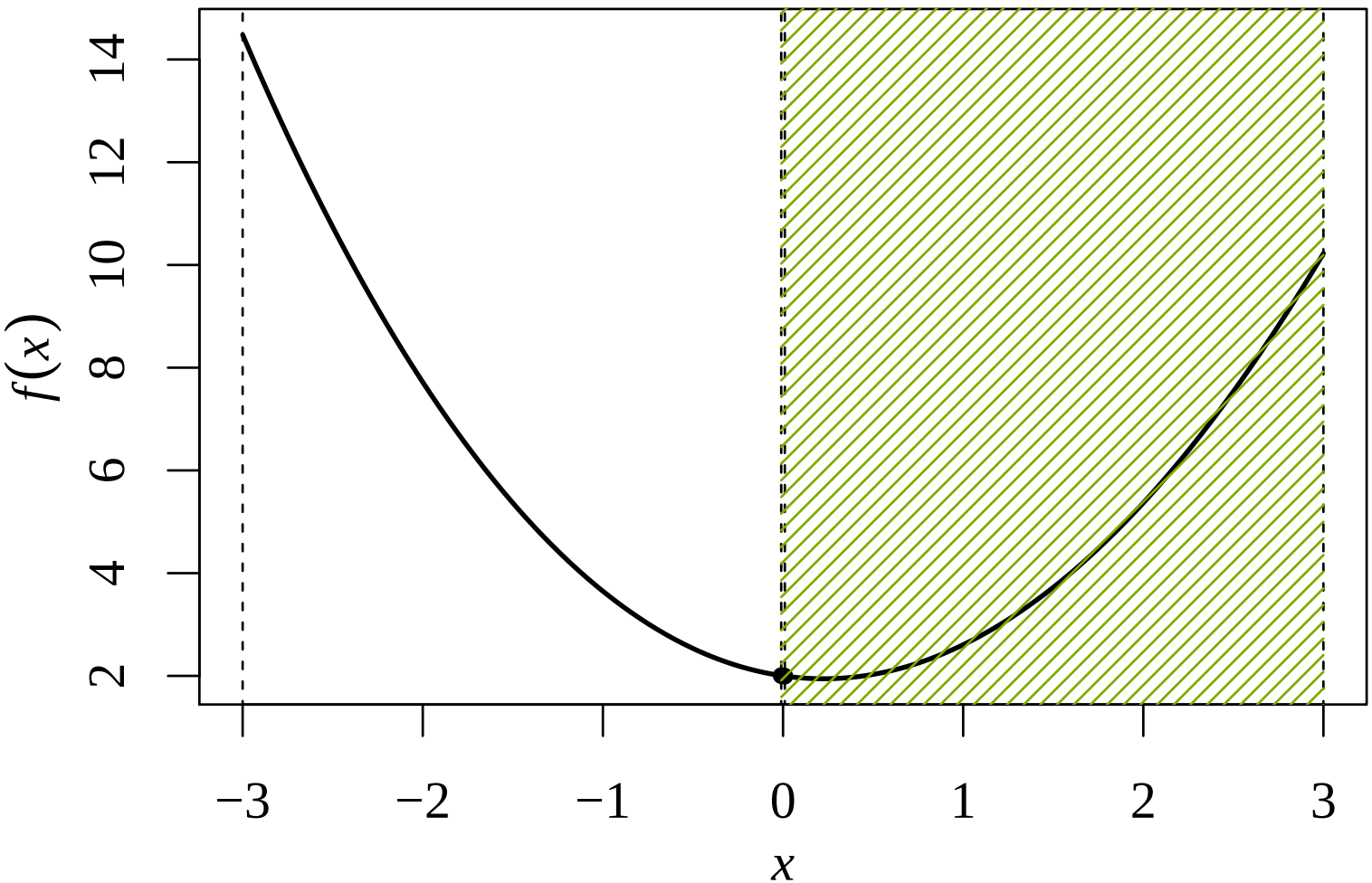
# Algorytm bisekcji – przykład



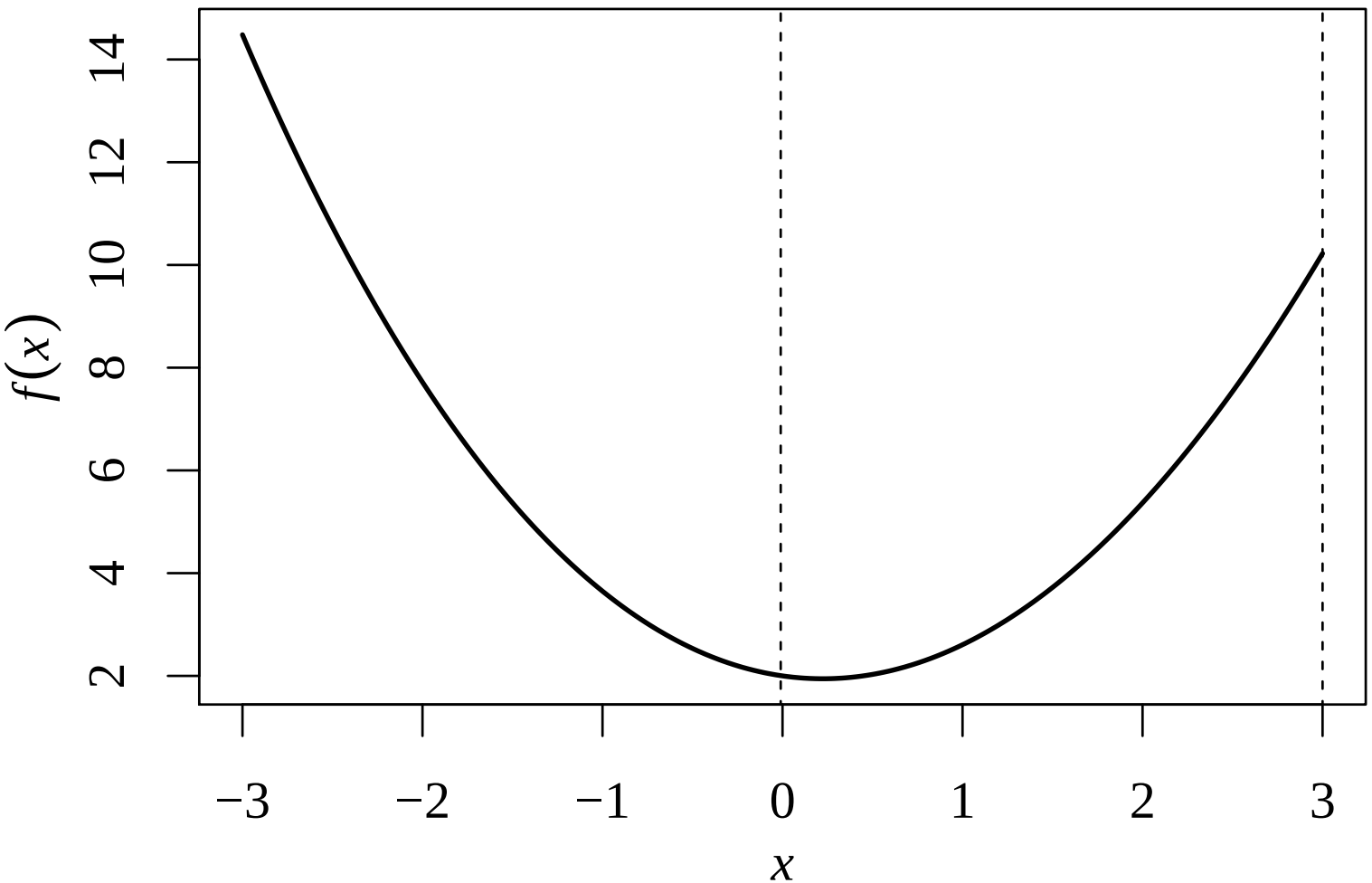
# Algorytm bisekcji – przykład



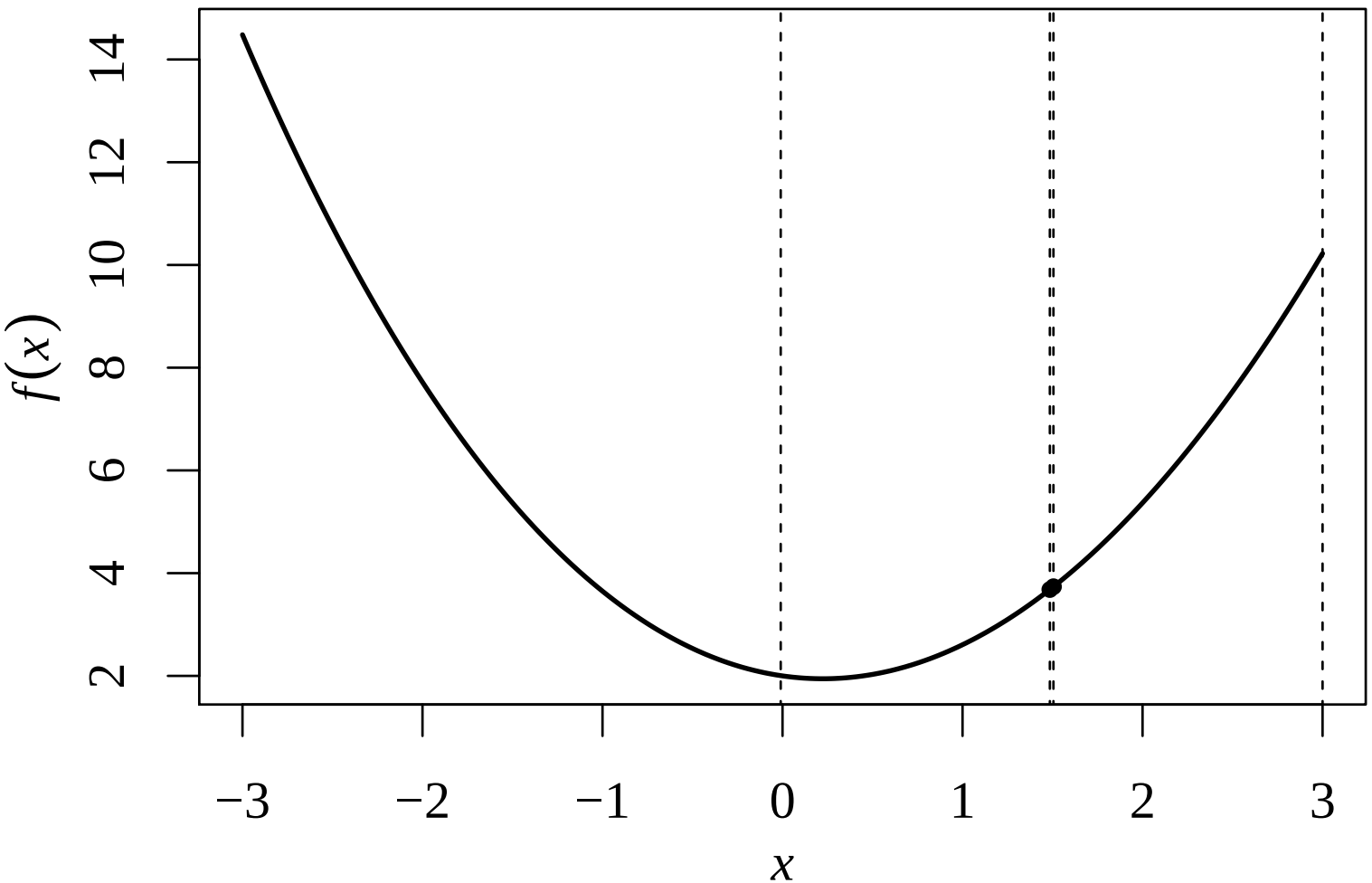
# Algorytm bisekcji – przykład



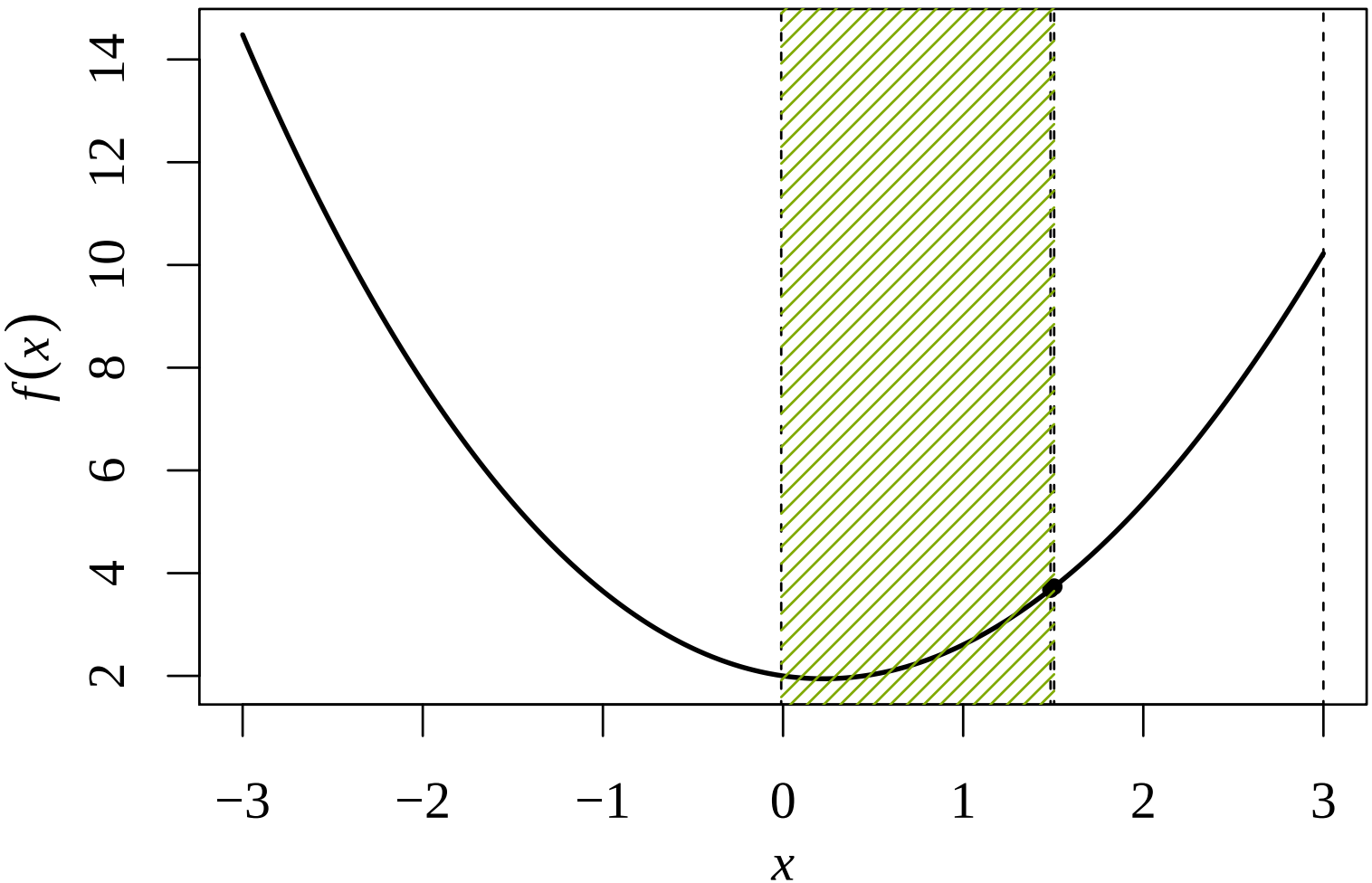
# Algorytm bisekcji – przykład



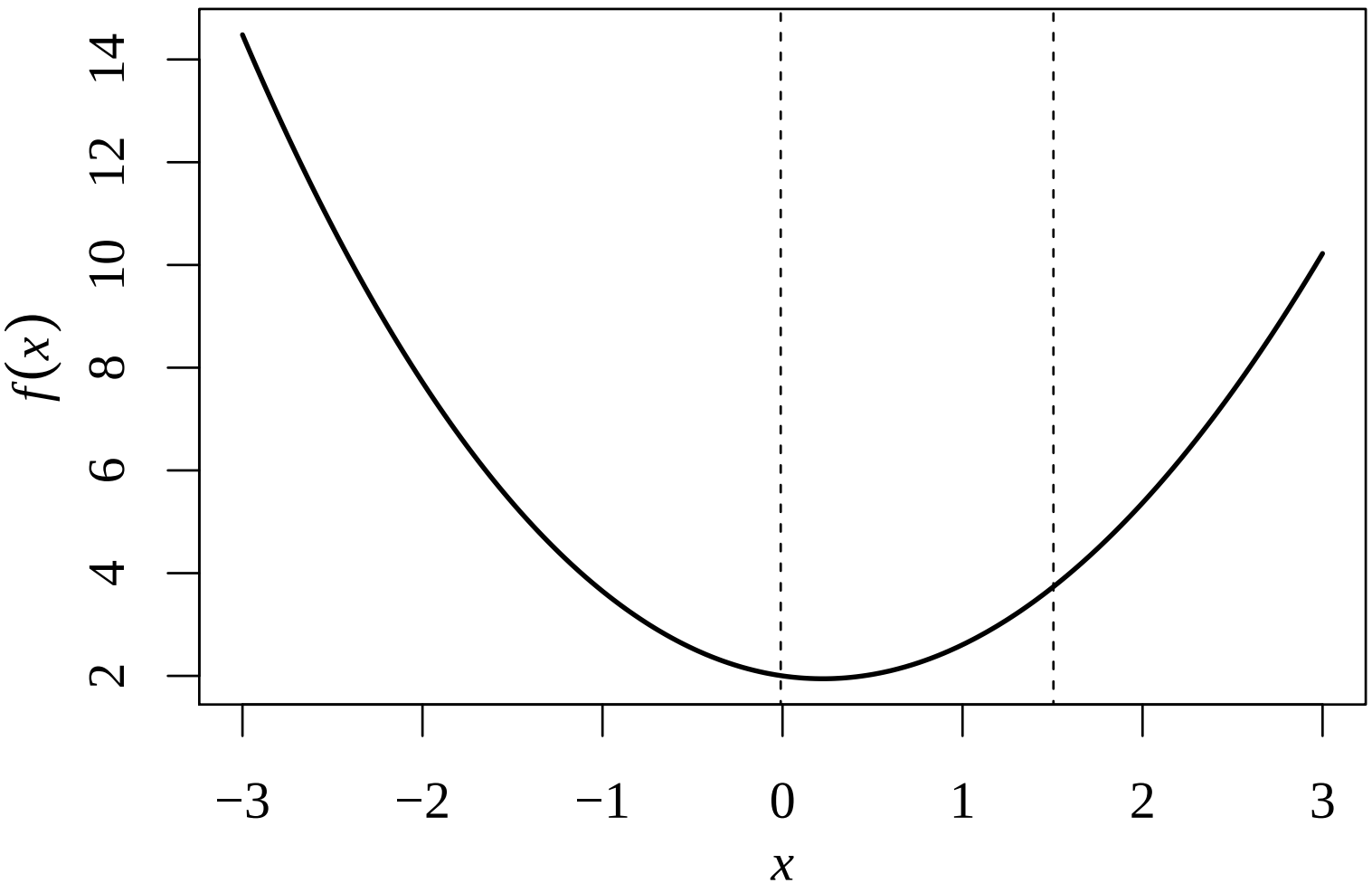
# Algorytm bisekcji – przykład



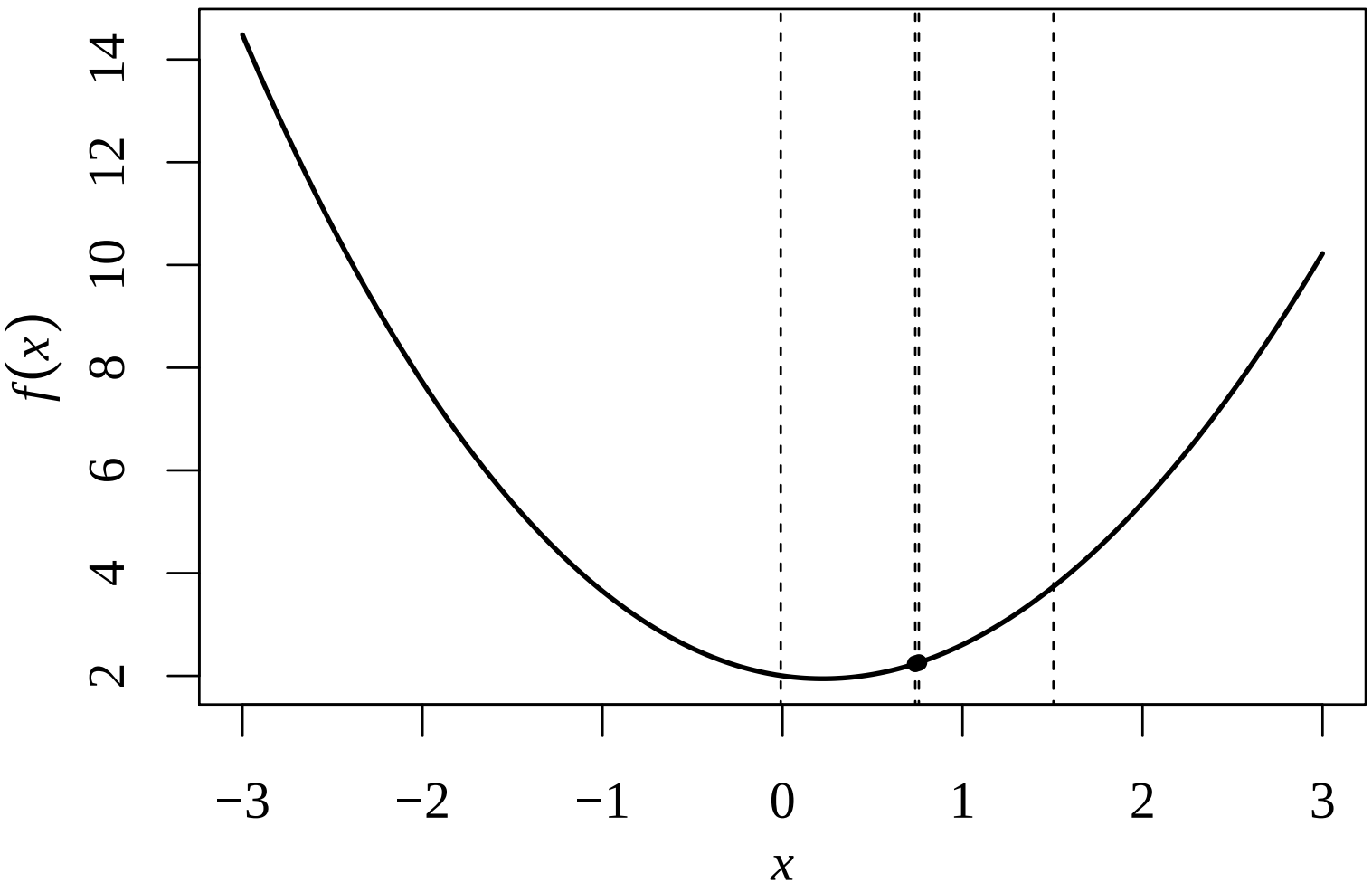
# Algorytm bisekcji – przykład



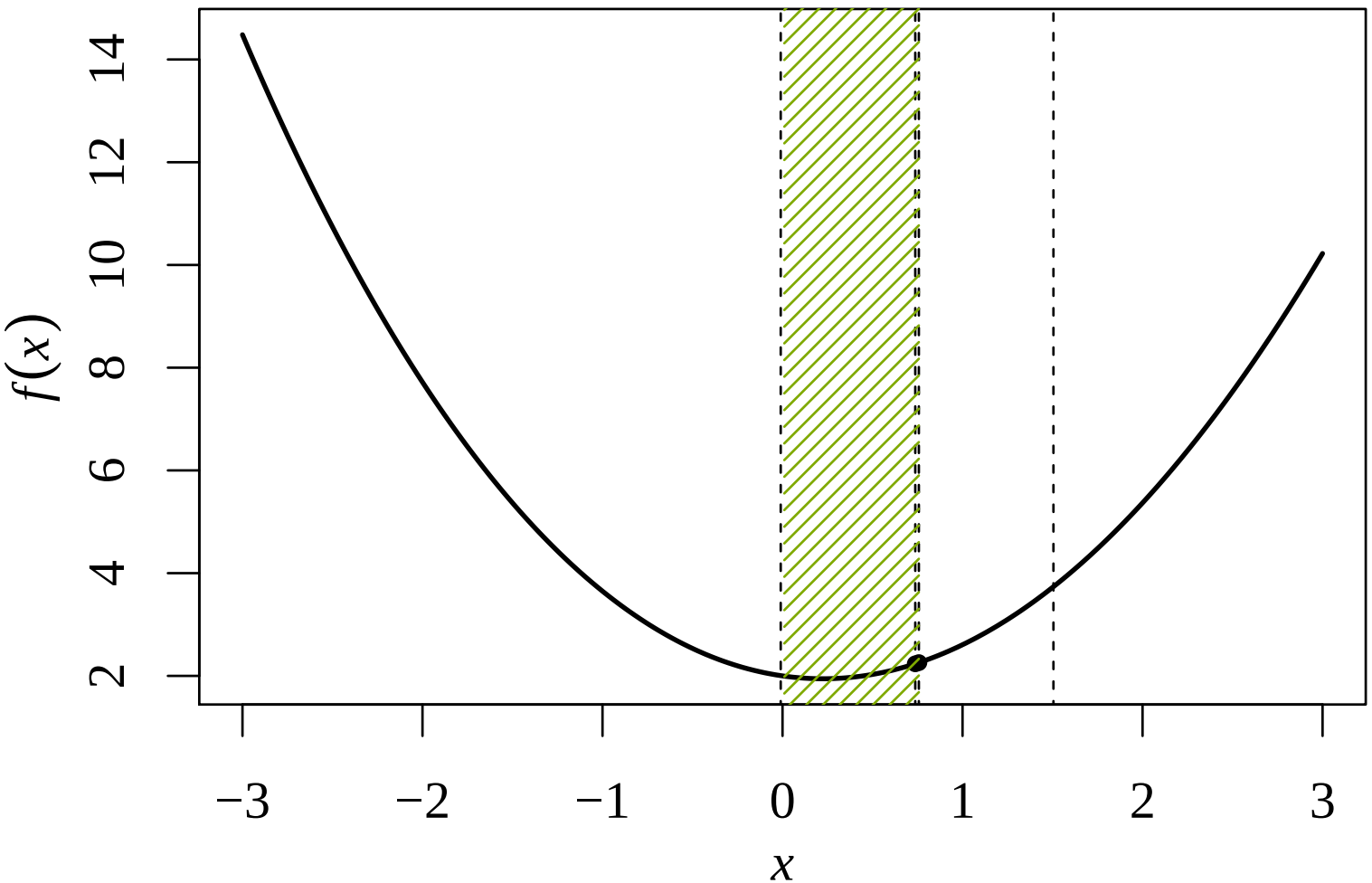
# Algorytm bisekcji – przykład



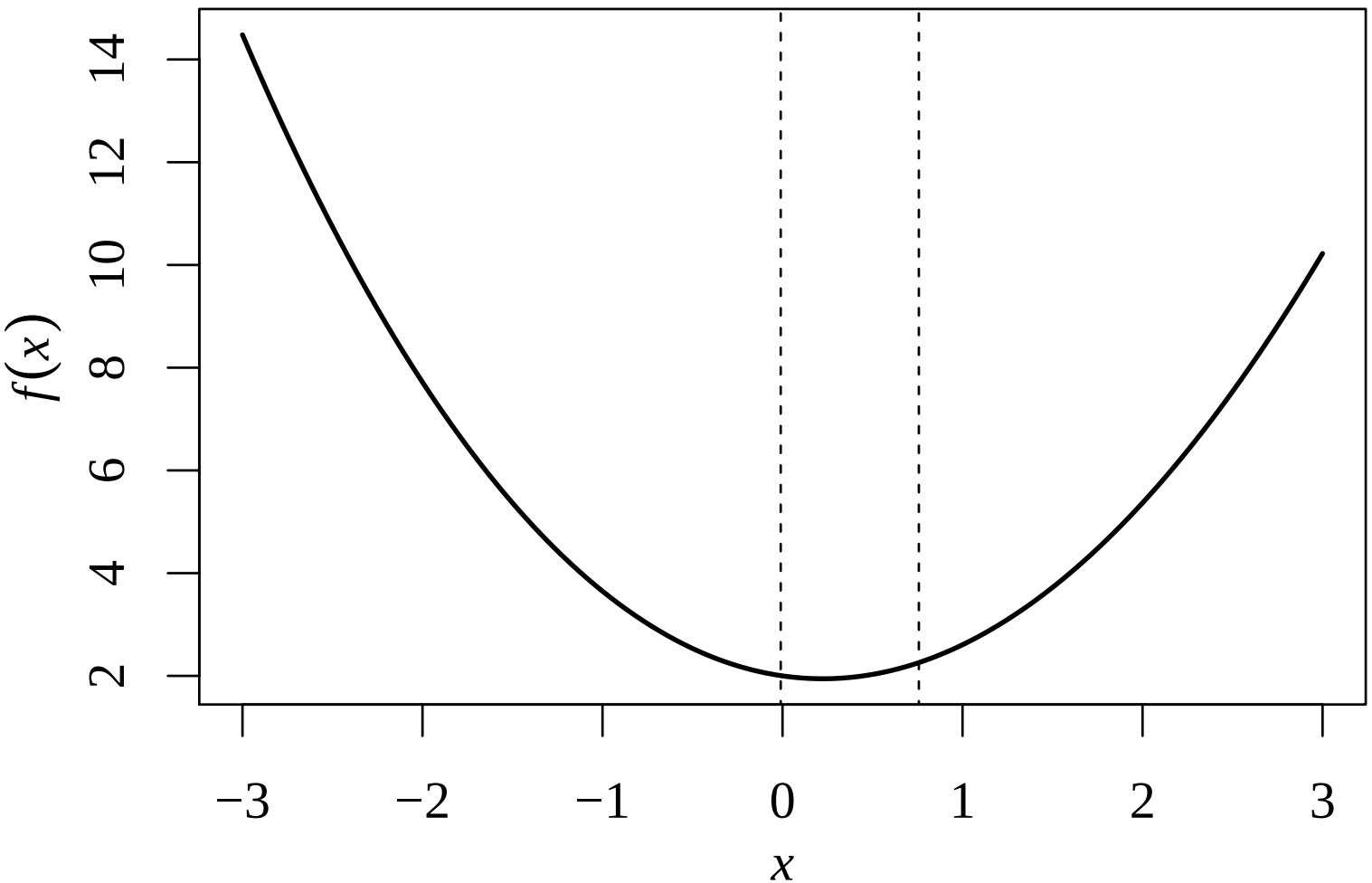
# Algorytm bisekcji – przykład



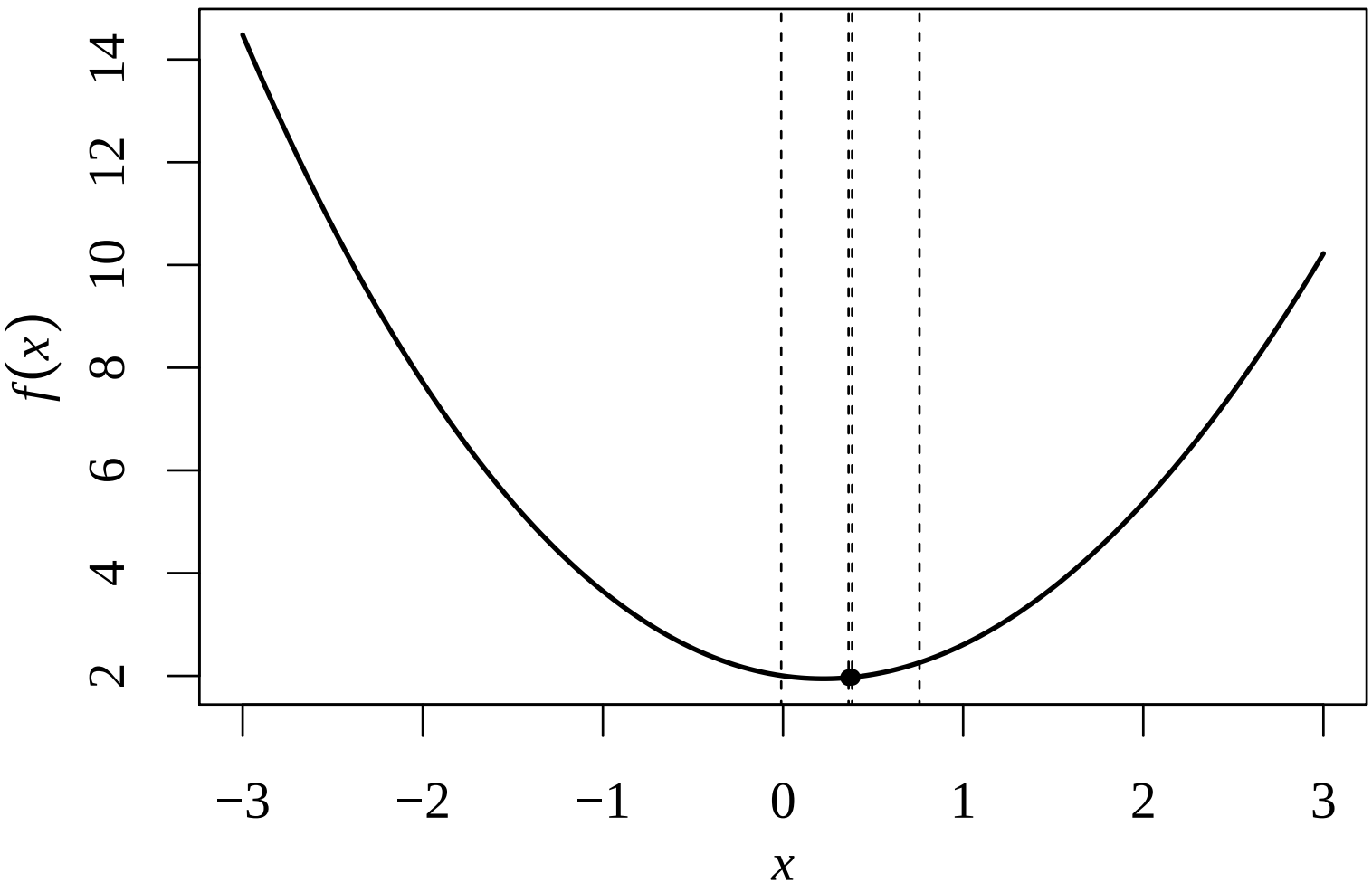
# Algorytm bisekcji – przykład



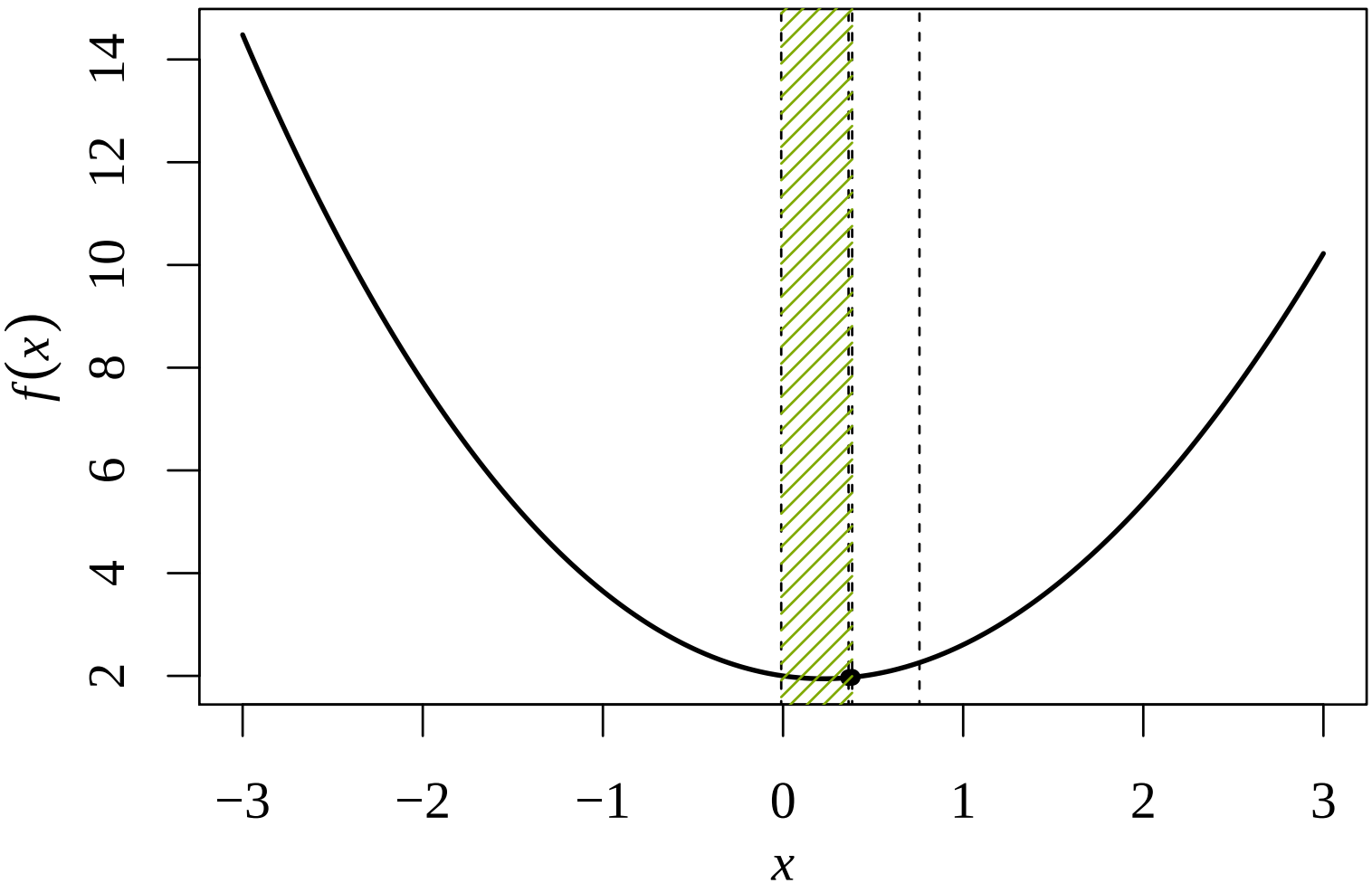
# Algorytm bisekcji – przykład



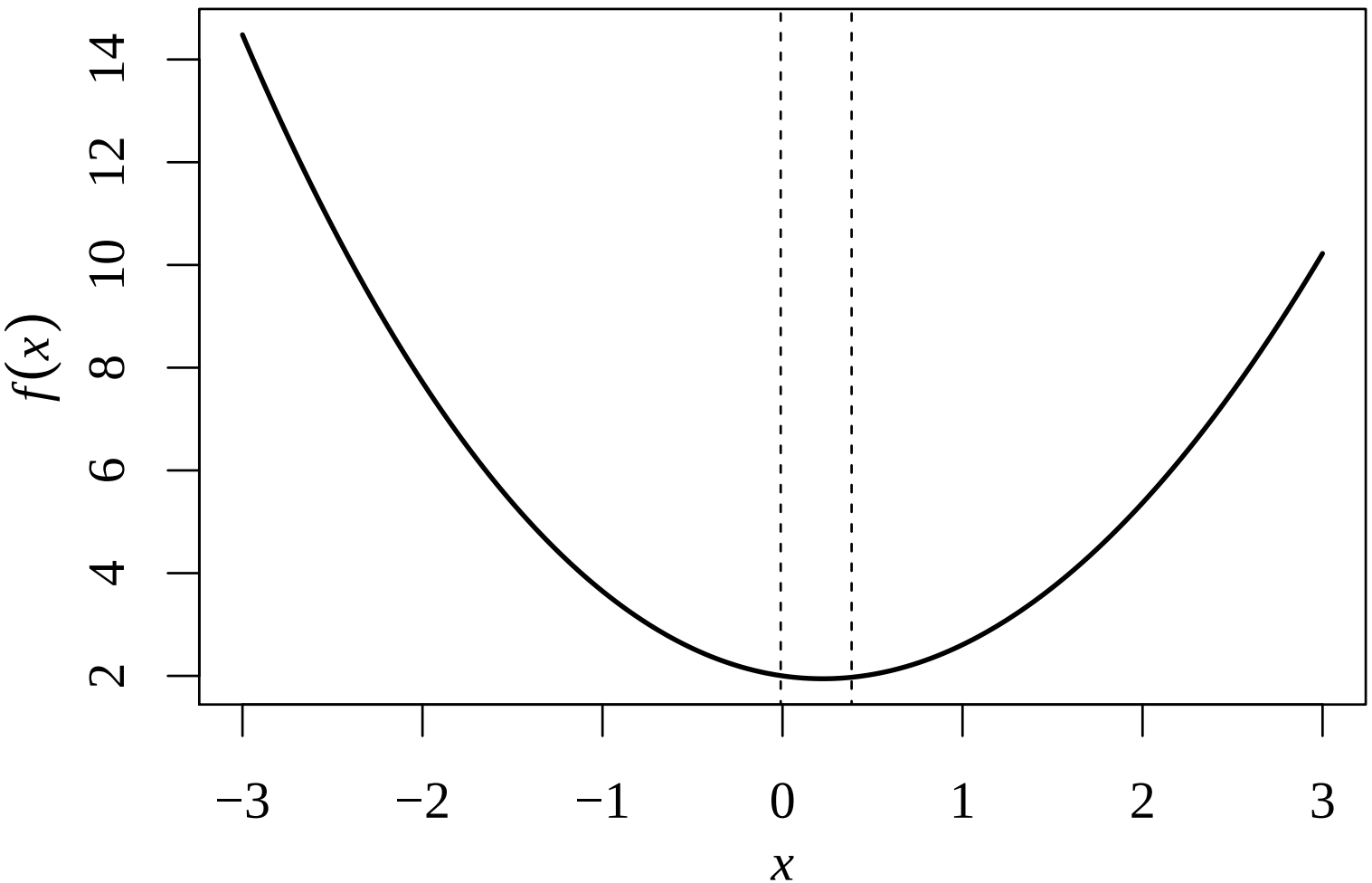
# Algorytm bisekcji – przykład



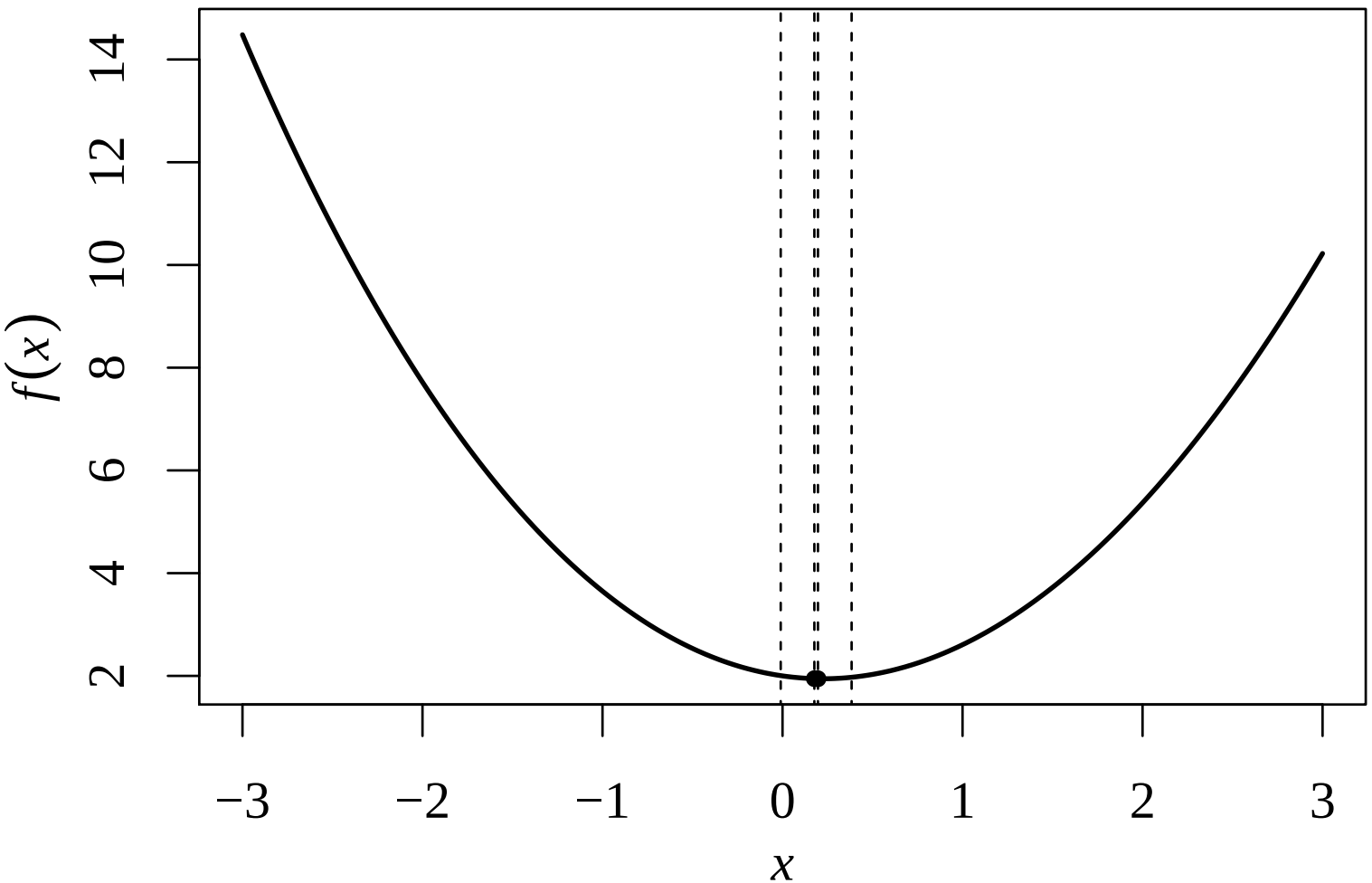
# Algorytm bisekcji – przykład



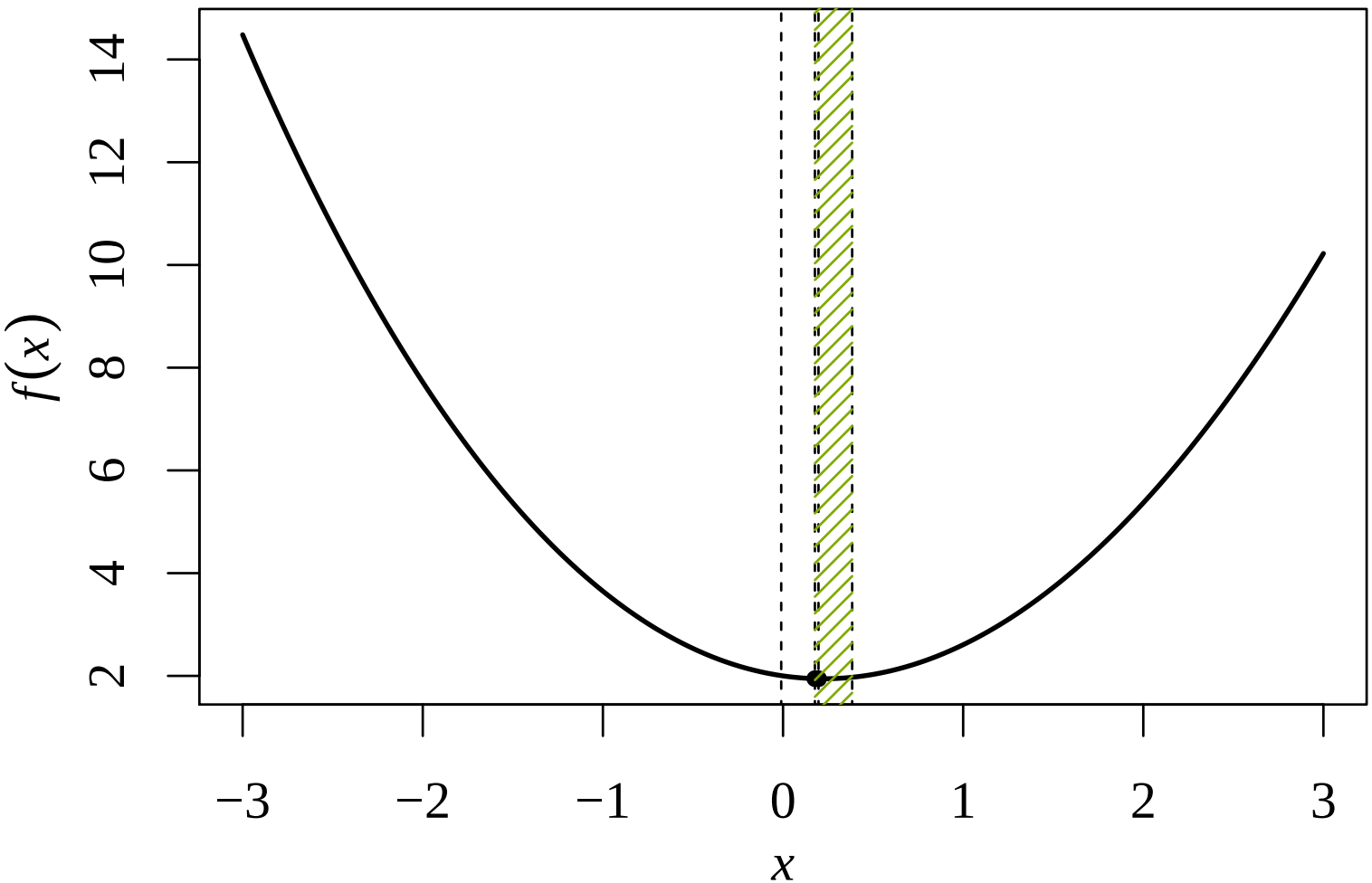
# Algorytm bisekcji – przykład



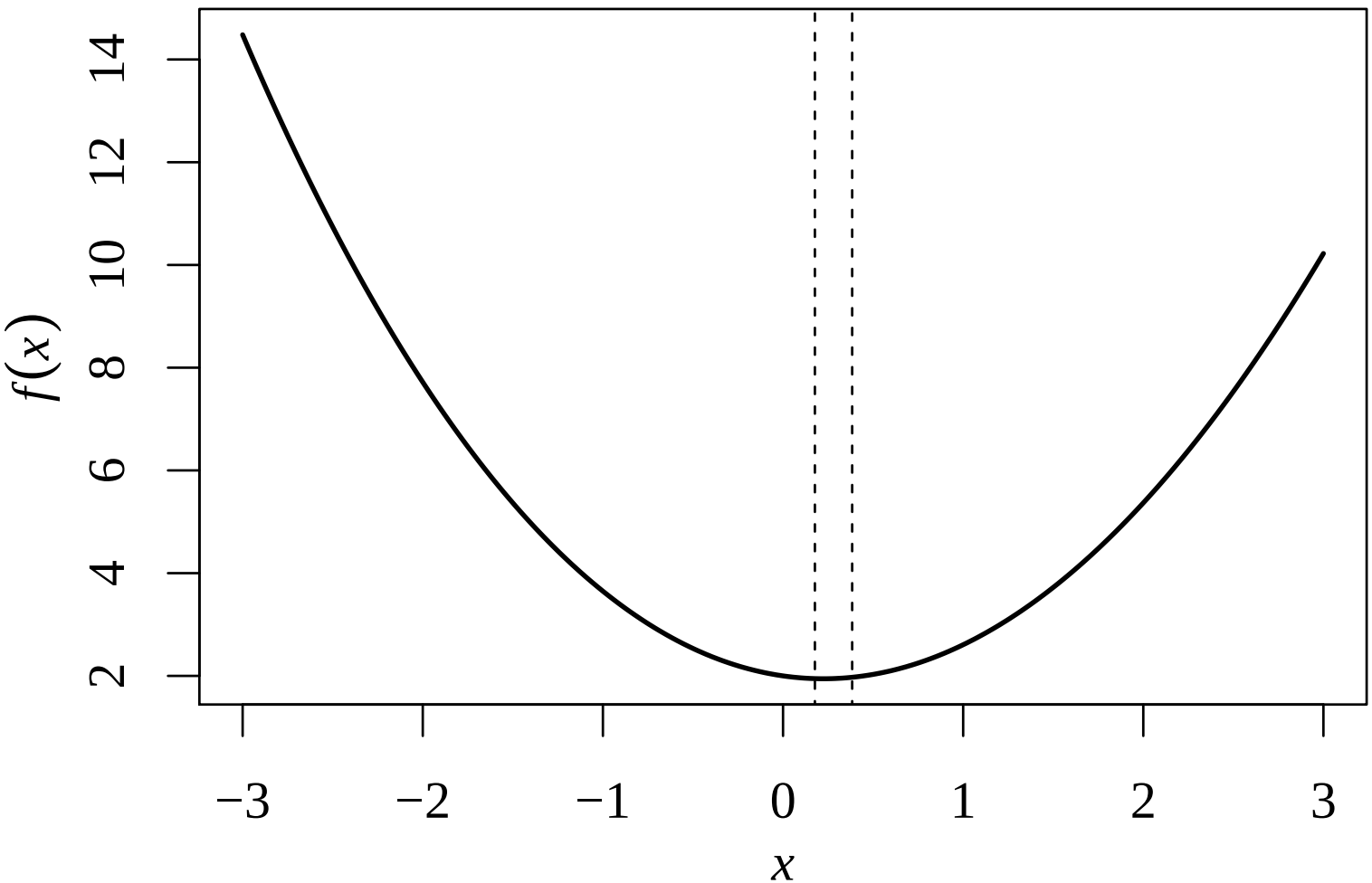
# Algorytm bisekcji – przykład



# Algorytm bisekcji – przykład



# Algorytm bisekcji – przykład



# Czy można to zrobić szybciej?

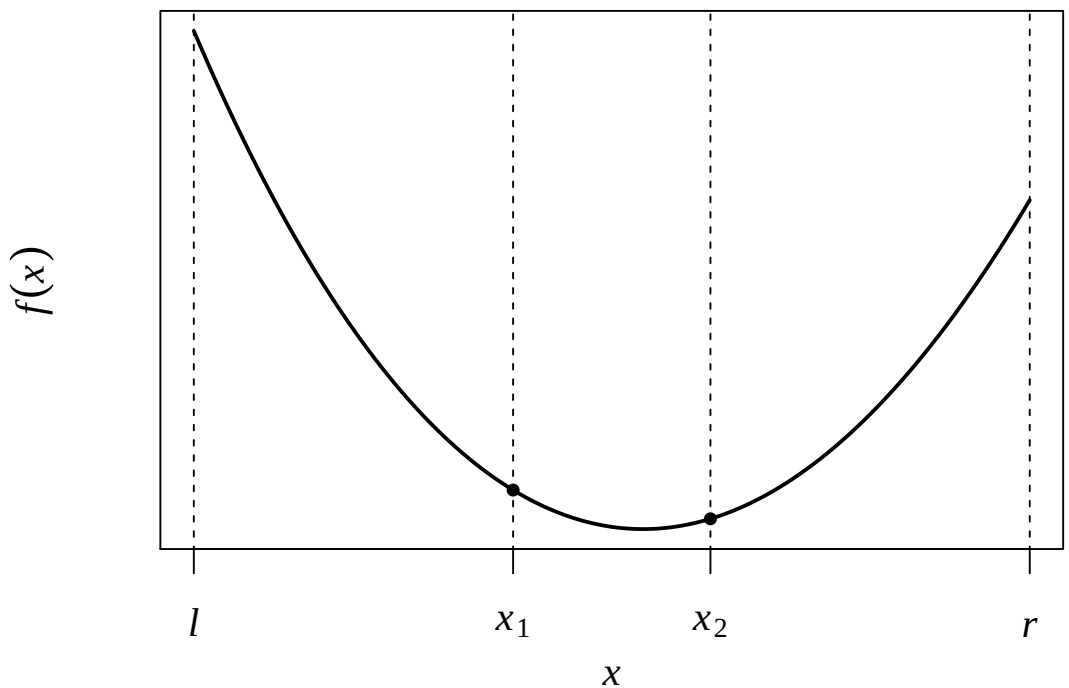
- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.

# Czy można to zrobić szybciej?

- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.
- Pomysł: wykorzystać jeden z punktów z poprzedniej iteracji i próbkować funkcję tylko raz na iterację!

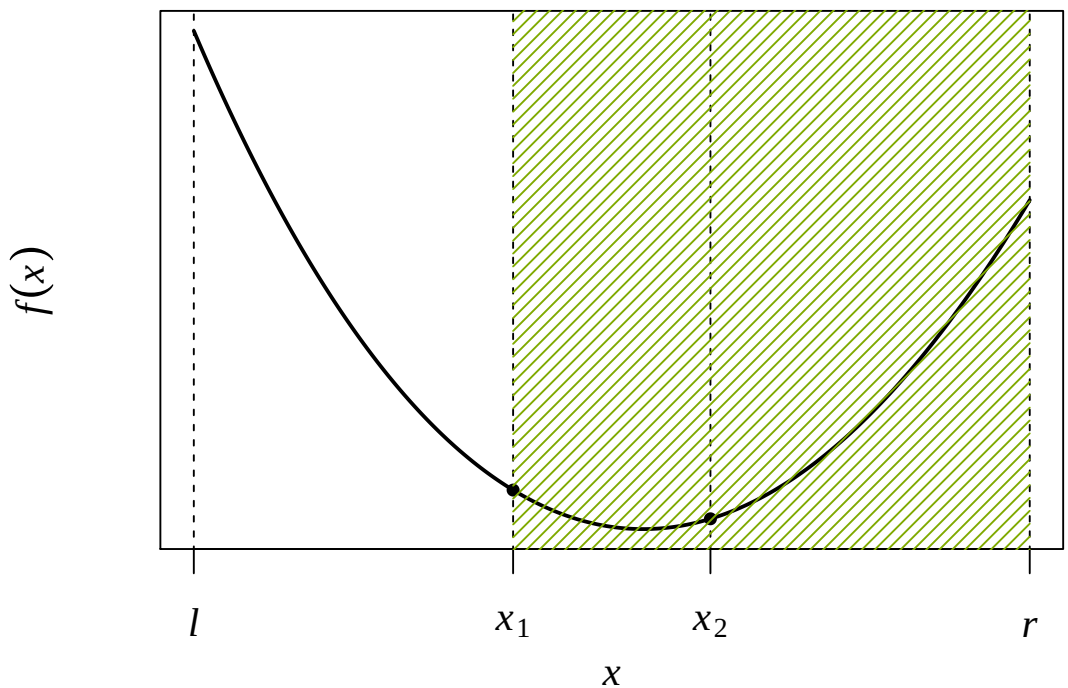
# Czy można to zrobić szybciej?

- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.
- Pomysł: wykorzystać jeden z punktów z poprzedniej iteracji i próbkować funkcję tylko raz na iterację!



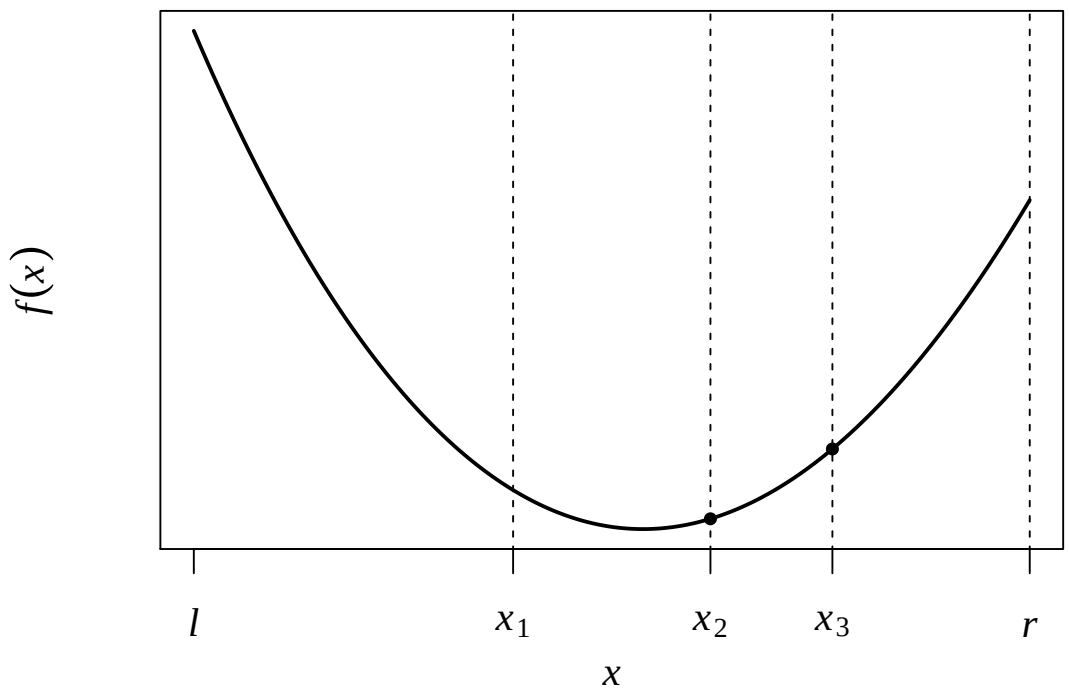
# Czy można to zrobić szybciej?

- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.
- Pomysł: wykorzystać jeden z punktów z poprzedniej iteracji i próbkować funkcję tylko raz na iterację!



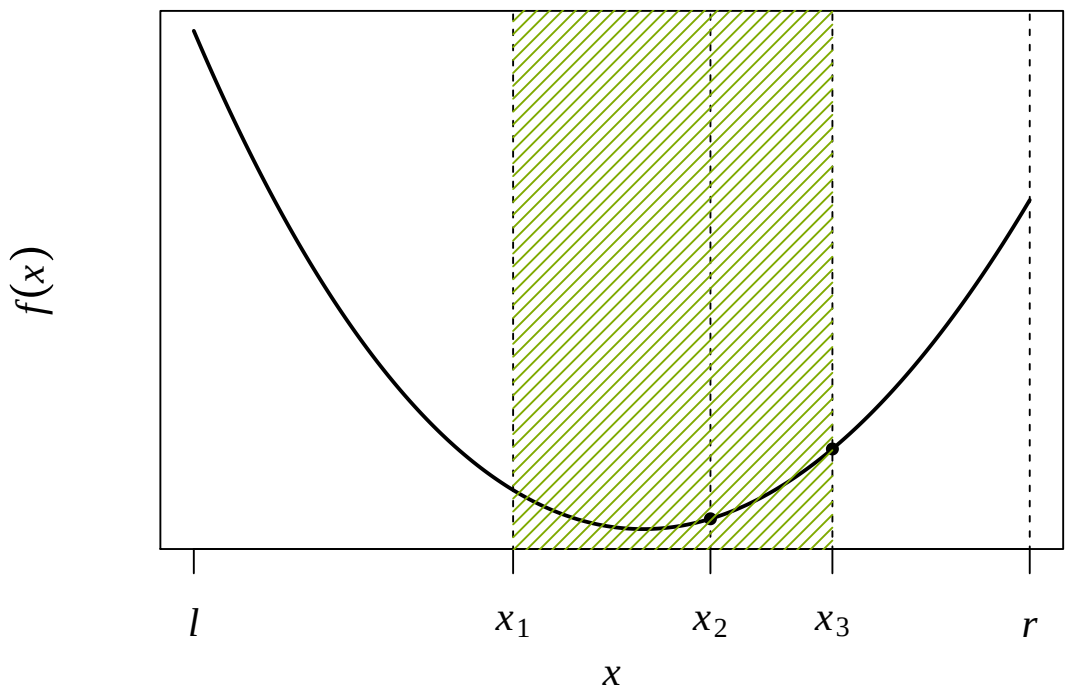
# Czy można to zrobić szybciej?

- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.
- Pomysł: wykorzystać jeden z punktów z poprzedniej iteracji i próbkować funkcję tylko raz na iterację!



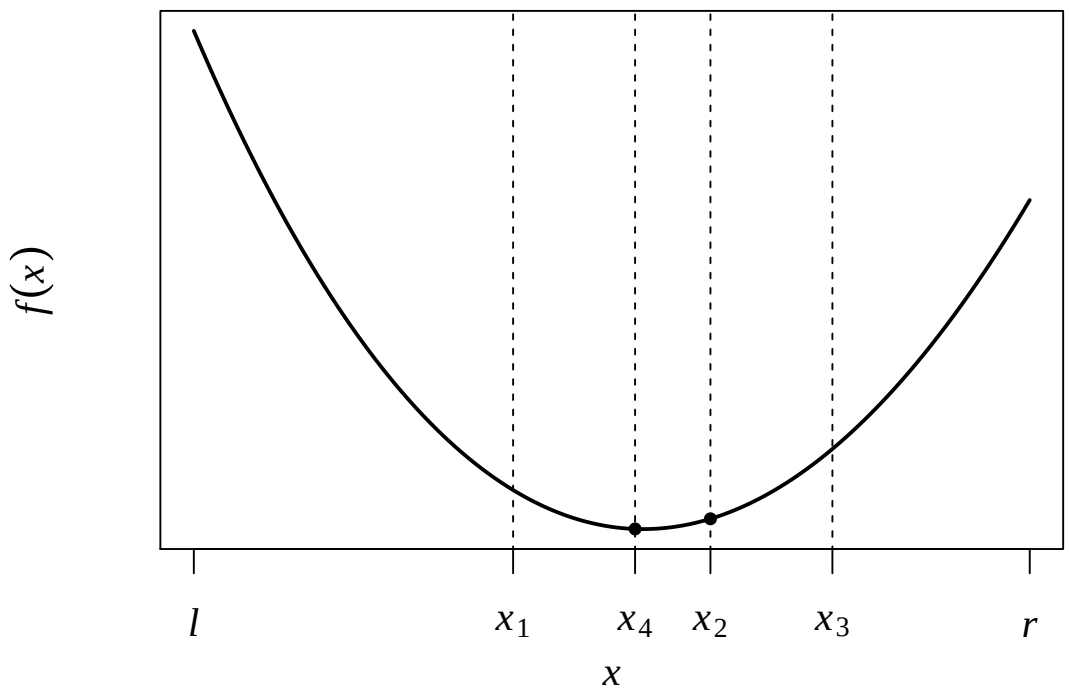
# Czy można to zrobić szybciej?

- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.
- Pomysł: wykorzystać jeden z punktów z poprzedniej iteracji i próbkować funkcję tylko raz na iterację!

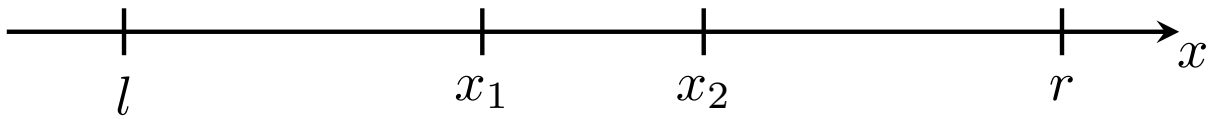


# Czy można to zrobić szybciej?

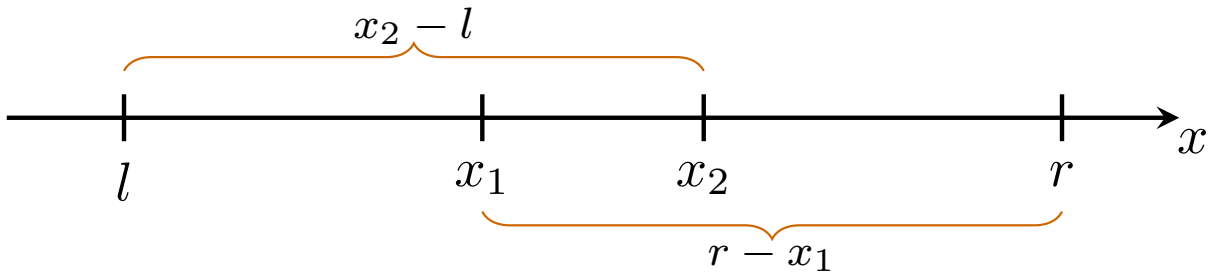
- Dotychczas w każdej iteracji sprawdzaliśmy funkcję w dwóch punktach i zmniejszaliśmy przedział przeszukiwań o połowę.
- Jeden z dwóch punktów staje się nową granicą przedziału, po czym próbkujemy funkcję w dwóch nowych punktach.
- Pomysł: wykorzystać jeden z punktów z poprzedniej iteracji i próbkować funkcję tylko raz na iterację!



# Konstrukcja algorytmu

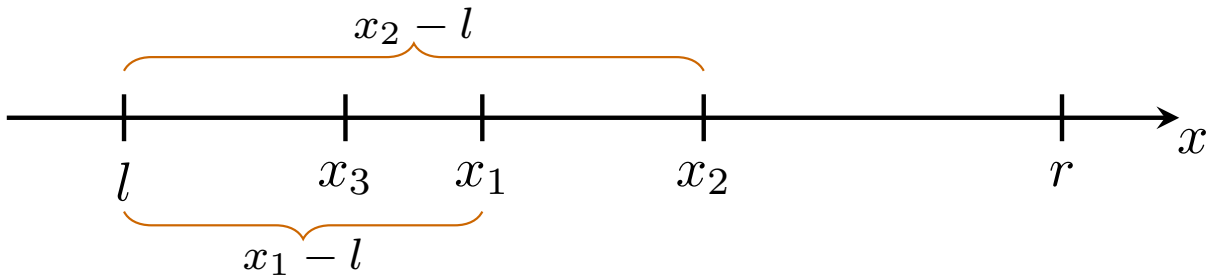


# Konstrukcja algorytmu




- W następnej iteracji obszar zwęża się do $x_2 - l$ lub $r - x_1$. Ponieważ nie wiemy, który przypadek nastąpi, ustalamy:
$$x_2 - l = r - x_1.$$

# Konstrukcja algorytmu




- W następnej iteracji obszar zwęża się do $x_2 - l$ lub $r - x_1$. Ponieważ nie wiemy, który przypadek nastąpi, ustalamy:
$$x_2 - l = r - x_1.$$

- Jeśli wybierzemy obszar np. $x_2 - l$, punkt $x_1$ zostanie użyty ponownie. Aby zachować proporcje podziału, ustalamy:
$$\frac{x_2 - l}{r - l} = \frac{x_1 - l}{x_2 - l}$$

# Złoty podział

- Otrzymaliśmy równania:

$$x_2 - l = r - x_1, \qquad \frac{x_2 - l}{r - l} = \frac{x_1 - l}{x_2 - l}.$$

# Złoty podział

- Otrzymaliśmy równania:

$$x_2 - l = r - x_1, \qquad \frac{x_2 - l}{r - l} = \frac{x_1 - l}{x_2 - l}.$$

- Podstawiając pierwsze równanie do drugiego:

$$\frac{x_2 - l}{r - l} = \frac{r - x_2}{x_2 - l} = \frac{r - l + l - x_2}{x_2 - l} = \frac{r - l}{x_2 - l} - 1.$$

# Złoty podział

- Otrzymaliśmy równania:

$$x_2 - l = r - x_1, \qquad \frac{x_2 - l}{r - l} = \frac{x_1 - l}{x_2 - l}.$$

- Podstawiając pierwsze równanie do drugiego:

$$\frac{x_2 - l}{r - l} = \frac{r - x_2}{x_2 - l} = \frac{r - l + l - x_2}{x_2 - l} = \frac{r - l}{x_2 - l} - 1.$$

- Definiując proporcję $p = \frac{x_2 - l}{r - l}$, dostajemy równanie:

$$p = \frac{1}{p} - 1 \qquad \Longrightarrow \qquad p^2 + p - 1 = 0,$$

# Złoty podział

- Otrzymaliśmy równania:

$$x_2 - l = r - x_1, \qquad \frac{x_2 - l}{r - l} = \frac{x_1 - l}{x_2 - l}.$$

- Podstawiając pierwsze równanie do drugiego:

$$\frac{x_2 - l}{r - l} = \frac{r - x_2}{x_2 - l} = \frac{r - l + l - x_2}{x_2 - l} = \frac{r - l}{x_2 - l} - 1.$$

- Definiując proporcję $p = \frac{x_2 - l}{r - l}$, dostajemy równanie:

$$p = \frac{1}{p} - 1 \qquad \Longrightarrow \qquad p^2 + p - 1 = 0,$$

mające rozwiązanie:

$$p = \frac{\sqrt{5} - 1}{2} \simeq 0.618.$$

Złoty podział odcinka.

# Algorytm II: złoty podział

Oznaczamy $\alpha := \frac{\sqrt{5}-1}{2} \simeq 0.618$.

Wejście: Procedura wyznaczająca wartość funkcji $f(x)$ w dowolnym punkcie dziedziny $[a, b]$; liczba odwołań do funkcji $n$.

Wyjście: Punkt $x^*$ odległy od minimum $f$ o najwyżej $(b-a)\alpha^{n-1}$.

1. Przypisz $\ell = a$, $r = b$, $x_1 = \alpha(b-a)$, $x_2 = (1-\alpha)(b-a)$.
2. Wyznacz $f(x_1)$, $f(x_2)$.
3. Powtarzaj dla $i = 1, 2, \ldots, n$:
   1. Jeśli $f(x_1) < f(x_2)$, przypisz $r = x_2$, $x_2 = x_1$, $x_1 = (1-\alpha)(r-l)$, i wyznacz $f(x_1)$.
   2. Jeśli $f(x_1) \geq f(x_2)$, przypisz $l = x_1$, $x_1 = x_2$, $x_2 = \alpha(r-l)$, i wyznacz $f(x_2)$.
4. Jako wynik zwróć punkt $x^* = \frac{\ell+r}{2}$.

# Porównanie

| algorytm | zbieżność | |
|---|---|---|
| bisekcji | $\left(\frac{1}{\sqrt{2}}\right)^n$ | $\simeq (0.707)^n$ |
| złotego podziału | $\left(\frac{\sqrt{5}-1}{2}\right)^{n-1}$ | $\simeq (0.618)^n$ |

# Porównanie

| algorytm | zbieżność | |
|---|---|---|
| bisekcji | $\left(\frac{1}{\sqrt{2}}\right)^n$ | $\simeq (0.707)^n$ |
| złotego podziału | $\left(\frac{\sqrt{5}-1}{2}\right)^{n-1}$ | $\simeq (0.618)^n$ |

- Zbieżność $(0.618)^n$ jest asymptotycznie optymalna.
- Algorytm optymalny to metoda Fibonacciego:
  - Zbieżność bardzo podobna jak dla algorytmu złotego podziału (asymptotycznie taka sama).
  - Bardziej skomplikowany i przez to mniej praktyczny.

Koniec na dzisiaj :)