Numeryszne metody przybliżania lokalnych ekstremum - sprawozdanie

Bartosz Zasieczny

17 listopada 2013

Spis treści

1 Zadanie

Korzystając z omówionych na wykładzie iteracyjnych metod aproksymacji pierwiasków, zaproponować sposób wyznaczania ***ekstremum lokalnego*** funkcji $f \in C^1[a, b]$. Wykonać eksperymenty m. in. dla:

0. $f(x) = sin(2\pi x)$, $x \in [0, 1]$;
1. $f(x) = e^{-x^2}$, $x \in [-1, 1]$;
2. $f(x) = \frac{x}{1+x^2}$, $x \in [0, 10]$;
3. $f(x) = x^2 + x - 1$, $x \in [-1, 2]$.

2 Aparat matematyczny

W poszukiwaniu ekstremów funkcji będziemy używać poniższych metod. Niektóre z nich pozwalają na znalezienie ekstremum wprost, inne będą skupiać się na poszukiwaniu miejsca zerowego pierwszej pochodnej funkcji tam gdzie to możliwe.

2.1 Metoda Newtona

Metoda Newtona polega na iteracyjnym wyznaczaniu kolejnych przybliżeń pierwiastka $f(x)$ poprzez:

- znalezienie stycznej do jej wykresu w punkcie x_i (zaczynając od punktu startowego x_0);
- biorąc wartosć dziedziny w punkcie przecięcia stycznej z osią X za $i + 1$-sze przyblizenie pierwiastka (czyli x_{i+1}).

Kroki powtarzamy aż do otrzymania wymaganej precyzji. Kolejne przybliżenia x_{i+1} wyznaczamy za pomocą wzoru:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

2.1.1 Uwagi

- Charakterystyka tego zadania uniemożliwia użycie samej metody Newtona - dla pewnych danych może ona wskazać przybliżenia pierwiastka $f(x)$ spoza pożądanego przedziału. Problemem też jest dobór odpowiedniego punktu startowego - dlatego w przypadku tego zadania należy stosować tę metodę tylko po wstępnym przybliżania pierwiastka funkcji przez inne metody iteracyjne.
- W przypadku tego zadania każda badana funkcja musi posiadać co najmniej dwie pochodne.

2.2 Metoda bisekcji

Dla funkcji $f(x)$ ciągłej w przedziale $[a, b]$ i przyjmującej na jego końcach wartości o różnych znakach ($f(a)f(b) < 0$) należy wykonać następujące kroki:

1. sprawdzić, czy srodek przedziału jest pierwiastkiem funkcji (sprawdzić czy $f(x)$ dla wartości dziedziny $x_0 = \frac{a+b}{2}$ ma wartość $f(x_0) = 0$;
2. jeśli tak, to zakończyć algorytm i zwrócić x_0;
3. w p. p. sprawdzić który z przedziałów ($[a, x_0]$ czy $[x_0, b]$) spełnia własnosć $f(a')f(b') < 0$ i zastosować do niego pierwszy krok algorytmu.

2.3 Metoda *złotego podziału*

Ta metoda w odróżnieniu od poprzednich pozwala szukać lokalnego ektremum wprost, bez konieczności odwoływania się do pochodnych danej funkcji i poszukwiania ich zer. Żeby funkcja $f(x)$ mogła zostać zbadana za pomocą tej metody, musi być ona w przedziale $[a, b]$, w którym poszukujemy ekstremum, ***unimodalna*** – tzn. ciągła i posiadać w tym przedziale dokładnie jedno ekstremum.

2.3.1 Algorytm

Pierwszy krok algorytmu:

$$\begin{cases} x_L^{(0)} := b^{(0)} - (b^{(0)} - a^{(0)})k \\ \\ x_R^{(0)} := a^{(0)} + (b^{(0)} - a^{(0)})k \end{cases}$$

Następnie iterujemy po przypadkach, aż do uzyskania zadowalającej precyzji:

- $f(x_L^{(i)}) > f(x_R^{(i)}) \Rightarrow \begin{cases} a^{(i+1)} := x_L^{(i)} \\ b^{(i+1)} := b^{(i)} \\ x_L^{(i+1)} := x_R^{(i)} \\ x_R^{(i+1)} := a^{(i+1)} + (b^{(i+1)} - a^{(i+1)})k \end{cases}$
- $f(x_L^{(i)}) < f(x_R^{(i)}) \Rightarrow \begin{cases} a^{(i+1)} := a^{(i)} \\ b^{(i+1)} := x_R^{(i)} \\ x_L^{(i+1)} := b^{(i+1)} - (b^{(i+1)} - a^{(i+1)})k \\ x_R^{(i+1)} := x_L^{(i)} \end{cases}$

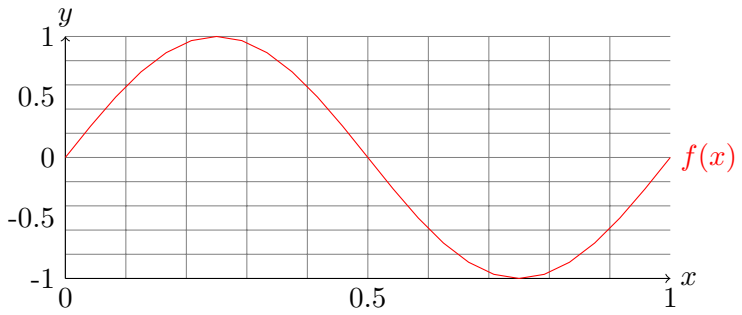
Po zakończeniu iteracji, środek przedziału $[a, b]$ jest brany jako przybliżenie lokalnego ekstremum funkcji.

3 Badanie funkcji

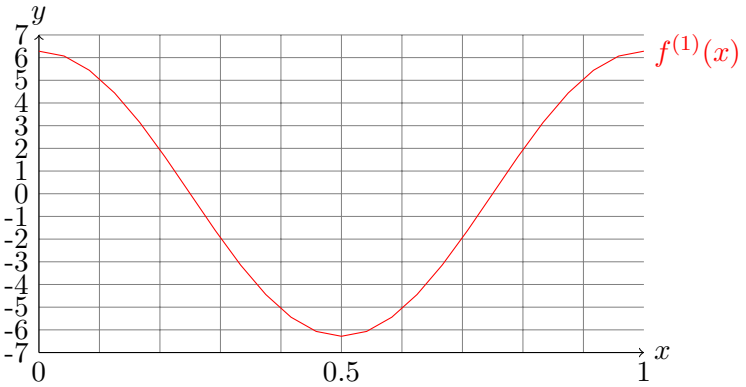
Do wykonywania obliczeń używana jest arytmetyka w standardzie IEEE 754 *double*.

3.1 Funkcja 0.

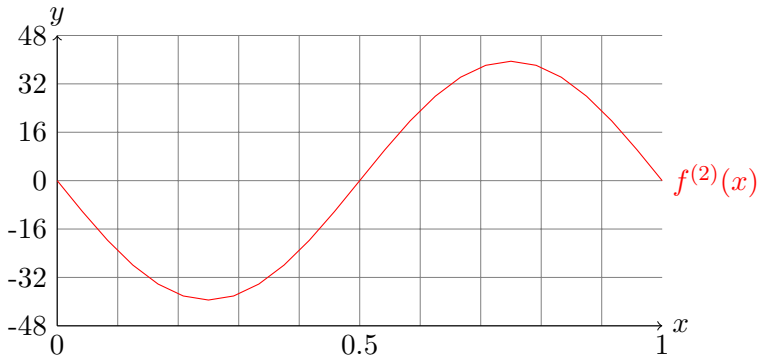
Wzory i wykresy funkcji i pochodnych w podanym przedziale:



$$f(x) = sin(2\pi x),\ x \in [0, 1]$$



$$f^{(1)}(x) = 2\pi \cdot cos(2\pi x)$$



$$f^{(2)}(x) = -4\pi^2 \cdot sin(2\pi x)$$

Z wykresów można wywnioskować, że w tym przedziale funkcja $f \in C^2$. Na wykresie f można zauważyć, że w przedziale $[0, 1]$ ma ona 2 ekstrema – jedno minimum, jedno maksimum. Sugeruje to możliwość użycia metody *złotego podziału*. Możemy natomiast użyć metody bisekcji, ew. łącząc ją z metodą Newtona (trzeba jednak uważać, gdyż na końcach przedziału $f^{(2)}(0) = 0$ i $f^{(2)}(1) = 0$ – nie mozemy użyć $x = 0$ i $x = 1$ jako punktów startowych tej metody).

3.1.1 Metoda *złotego podziału*

Polecenie: `./main -m golden_section -f 0 -s 0 1 -i 50 -error 10e-8`

i	x_i	$f(x_i)$
0	0.690 983 006	−0.932 032 423 813 227 595 13
1	0.809 016 994	−0.932 032 423 813 227 706 16
2	0.736 067 977	−0.996 171 040 864 827 661 57
3	0.781 152 949	−0.980 904 061 602 817 178 84
4	0.753 288 904	−0.999 786 490 708 711 950 15
5	0.736 067 977	−0.996 171 040 864 827 661 57
31	0.750 000 032	−0.999 999 999 999 980 127 01
32	0.749 999 993	−0.999 999 999 999 998 889 78
33	0.750 000 017	−0.999 999 999 999 994 448 88

Metoda *złotego podziału* dla tej funkcji osiąga wymaganą precyzję (badany przedział jest mniejszy niż 10^{-8}) dopiero po 34 iteracjach. W ten sposób znaleźliśmy przybliżenie tylko lokalnego minimum funkcji $f(x) = -1$, $x = 0.75$ i błąd przybliżenia wynosi $|x_{33} - x| \approx 0.00000001674441429955$.
Chcąc przybliżyć minimum nalezy wykonać to samo polecenie dodając paramatr `-e max`. Lokalne maksimum funkcji znajduje się w $x = 0.25$ i $f(x) = 1$

- aby otrzymać wynik o takiej samej precyzji należy wykonać dokładnie tyle samo iteracji i błąd przybliżenia jest bardzo zbliżony. Trzeba jednak pamiętać, że dokładność przybliżeń w przypadku tej funkcji zależy nie tylko od precyzji arytmetyki, ale również od dokładności reprezentacji piczby $\pi = 3.14159265358979323846$ (reprezentacja w `cmath`).
Dla tej funkcji metoda *złotego podziału* daje wyniki o rządanej precyzji po dość dużej liczbie iteracji. Aby otrzymać precyzję rzędu 10^{-16} program musi wykonać 72 iteracje.

3.1.2 Użycie metody *złotego podziału* jako wstęp do metody Newtona

Warto tej metody użyć jako wstęp do metody Newtona. Weźmy 6-tą iterację z powyższej tabeli ($i = 5$) i użyjmy jej jako punkt startowy do metody Newtona (polecenie: `./main -m newton -f 1 -d 2 -x 0.75328890437410611636 -error 10e-8`). Otrzymujemy następujące dane:

i	x_i	$f^{(1)}(x_i)$
0	0.753 288 904	0.129 831 499 475 402 373 23
1	0.749 999 532	−0.000 018 485 284 875 833 62
2	0.750 000 000	0.000 000 000 000 004 426 39

Otrzymujemy w tym wypadku dość szybko (tylko 3 iteracje!) dane bardzo dokładne (błąd: $|x_2 - x| \approx 0.00000000000000011102$) przybliżenie, które jest znacznie lepsze od uzyskanego na drodze zasosowania samej metody *złotego podziału*.

3.1.3 Metoda bisekscji

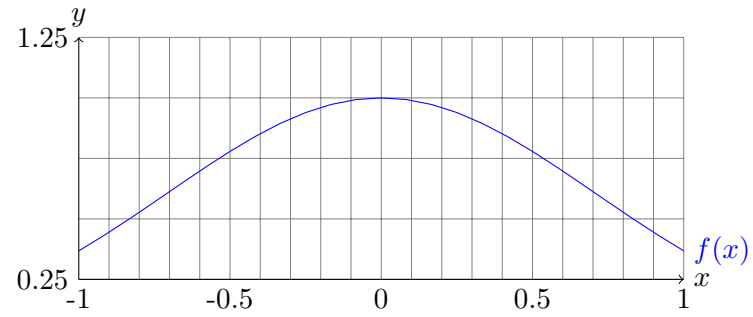
Uzywając tej metody musimy podzielić przedział na dwa: $[0, 0.5]$ i $[0.5, 1]$. Łatwo zauważyć, że metoda bisekcji zakończy się wtedy już po pierwszej iteracji, gdyż już wtedy dokładnie w połówach przedziałów będziemy mieć zera $f^{(1)}$. Aby się o tym przekonać wystarczy wykonać następujące polecenia:

- `./main -m bisection -f 1 -s 0 0.5` - dla przedziału $[0, 0.5]$,
- `./main -m bisection -f 1 -s 0.5 1` - dla przedziału $[0.5, 1]$

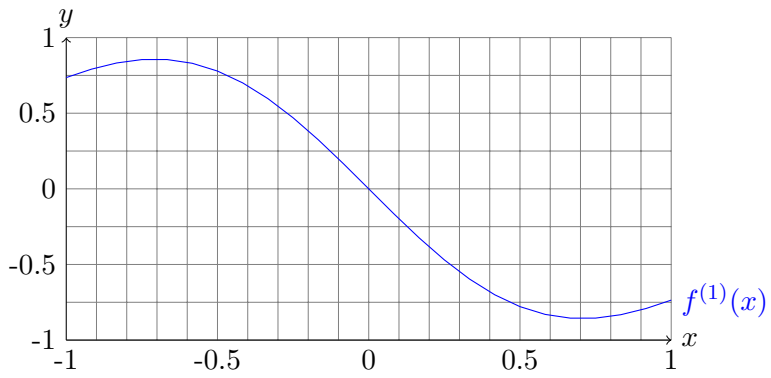
Jak widać nie ma sensu łączenie tej metody z metodą Newtona, gdyż daje ona bardzo dokładne wyniki w czasie praktycznie stałym dla tej funkcji i tych danych wejściowych.

3.2 Funkcja 1.

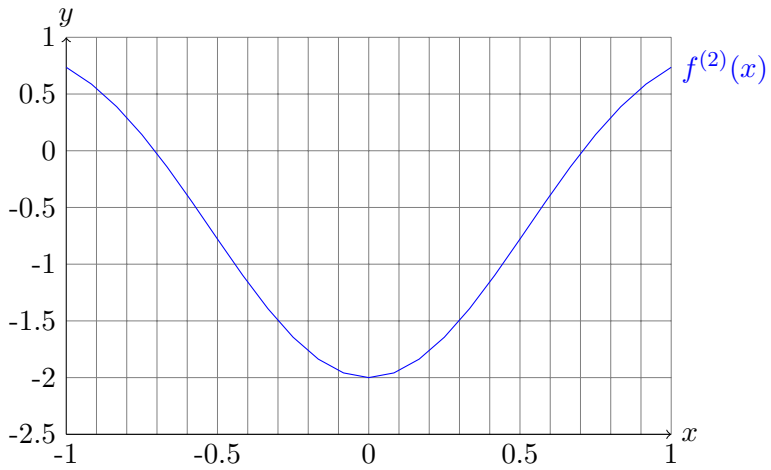
Wzory i wykresy funkcji i pochodnych w podanym przedziale:



$$f(x) = e^{-x^2},\ x \in [-1, 1]$$



$$f^{(1)}(x) = -2e^{-2^x} \cdot x)$$



$$f^{(2)}(x) = e^{-x^2} \cdot (4x^2 - 2)$$

3.2.1 Metoda *złotego podziału*

Funkcja f jest unimodalna w przedziale $[-1, 1]$, możemy więc wprost zastosować tę metodę (polecenie: `./main -m golden_section -f 3 -s -1 1 -i 50 -error 10e-8`). Wyniki przezentują się następująco:

i	x_i	$f(x_i)$
0	0.381 966 011	0.864 245 822 597 299 206 97
1	0.145 898 034	0.978 938 716 713 411 483 11
2	−0.000 000 000	1.000 000 000 000 000 000 00
3	0.090 169 944	0.991 902 345 324 935 708 07
4	0.034 441 854	0.998 814 462 016 431 980 91
5	−0.000 000 000	1.000 000 000 000 000 000 00
32	−0.000 000 000	1.000 000 000 000 000 000 00
33	0.000 000 048	0.999 999 999 999 997 668 53
34	0.000 000 019	0.999 999 999 999 999 666 93

Dla tej funkcji uzyskanie zadowalającej precyzji wyniku wymaga równie wieli iteracji. Patrząc na dane otrzymane z przybliżania ekstremum tej i poprzedniej funkcji można dojść do wniosku, że dla pewnych funkcji algorytm dość szybko zaczyna *kręcić się* w okół rozwiązania, ale szerokość badanego aktualnie przedziału nie pozwala na zakończenie algorytmu przy obecnej iteracji. Tym razem błąd przybliżenia wynosi $|x_{34} - x| = 0.000000018512169099$, a dokładnym ekstremum jest $x = 0$, $f(x) = 1$.

3.2.2 Użycie metody *złotego podziału* jako wstęp do metody Newtona

Jako punkt startowy weźmy $x_2 = -0.00000000000000005551$ (polecenie: `./main -m newton -f 4 -d 5 -x -0.00000000000000005551 -error 10e-8`).

i	x_i	$f^{(1)}(x_i)$
0	0.000 000 000	−0.000 000 000 000 000 000 111 02
1	0.000 000 000	0.000 000 000 000 000 000 000 00

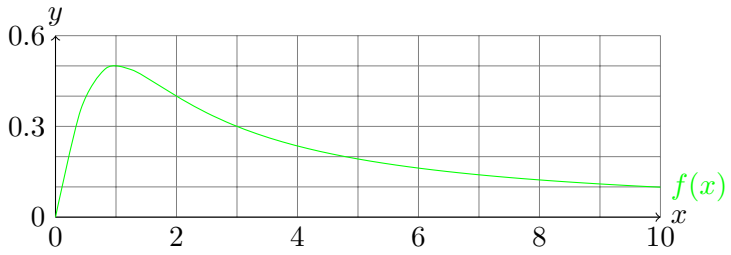
Jak widać w tym przypadku rozsądne wybranie przyblizenia z metody *złotego podziału* pozwala bardzo szybko przybliżyć ekstremum funkcji. Trzeba jednak uważać na dobór punktu wyjściowego, ponieważ w tym wypadku możemy wybrać punkt który spowoduje, że metoda Newtona nie zakończy swojego działania. Łatwo się można o tym przekonać wybierając takie $|x| > |x_{root}| \approx \pm 0.707106$, $f^{(2)}(x_{root}) = 0$ (polecenie: `./main -m newton -f 4 -d 5 -i 100 -x 0.8`).

3.2.3 Metoda bisekcji

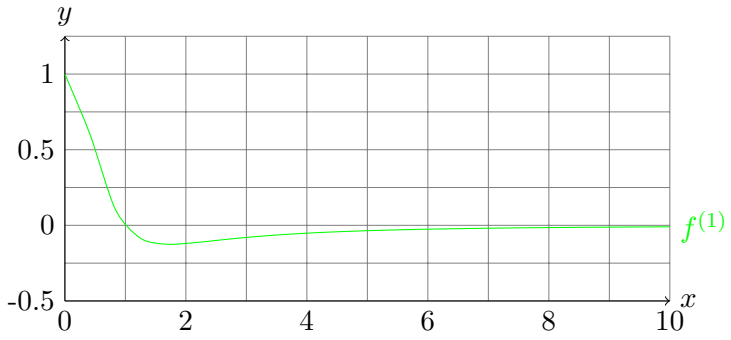
Zastosowanie metody bisekcji będzie miało podobny efekt jak przy poprzedniej funkcji. Aby się o tym przekonać nalezy wykonać polecenie: `./main -m bisection -f 4 -s -1 1`.

3.3 Funkcja 2.

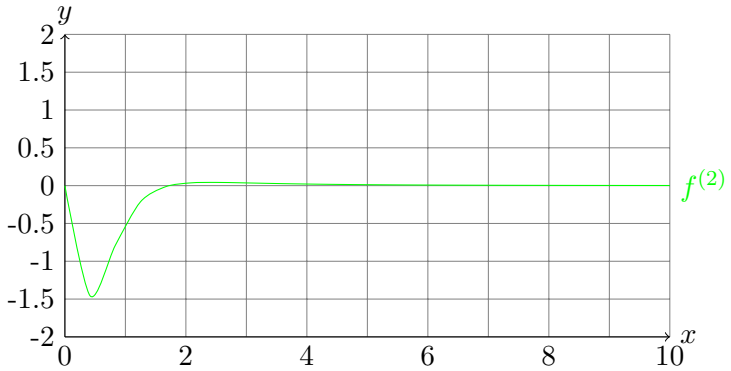
Wzory i wykresy funkcji i pochodnych w podanym przedziale:



$$f(x) = \frac{x}{1+x^2},\ x \in [0, 10]$$



$$f^{(1)}(x) = \frac{1-x^2}{(1+x^2)^2}$$



$$f^{(2)}(x) = \frac{2x(x^2-3)}{(1+x^2)^3}$$

3.3.1 Metoda *złotego podziału*

Zastosowanie tej metody dla tej funkcji potwierdza jej właściwości - szybko zbiega ku prawidłowemu rozwiązaniu, jednak algorytm się nie zatrzymuje gdyż aktualnie badany przedział jest za duży. (polecenie: `./main -m golden_section -f 6 -s 0 10 -i 50 -error 10e-8 -e max`).

i	x_i	$f(x_i)$
0	3.090 169 944	0.292 930 696 915 101 707 05
1	1.909 830 056	0.410 941 421 535 564 732 63
2	1.180 339 887	0.493 205 238 917 260 813 11
3	0.729 490 169	0.476 120 064 380 878 560 42
4	1.008 130 619	0.499 983 607 086 038 672 75
36	1.000 000 031	0.499 999 999 999 999 777 96
37	0.999 999 996	0.500 000 000 000 000 000 00
38	1.000 000 017	0.499 999 999 999 999 888 98

3.3.2 Użycie metody *złotego podziału* jako wstęp do metody Newtona

Jako punkt wyjściowy do metody Newtona weźmiemy $x_4 = 1.00813061875578369175$ (polecenie: `./main -m newton -f 7 -d 8 -x 1.00813061875578369175 -error 10e-8`).

i	x_i	$f^{(1)}(x_i)$
0	1.008 130 619	−0.004 015 997 888 929 203 13
1	0.999 899 483	0.000 050 266 298 445 957 92
2	0.999 999 985	0.000 000 007 576 547 674 60

3.3.3 Metoda bisekcji

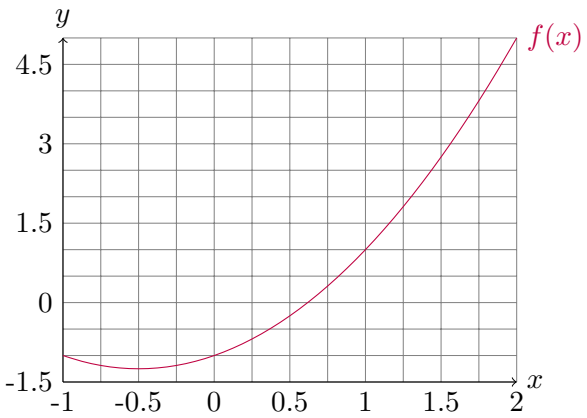
Metoda bisekcji prezentuje następujące wyniki (polecenie: `./main -m bisection -f 7 -s 0 10 -error 10e-8 -i 30`):

i	x_i	$f(x_i)$
0	5.000 000 000	−0.035 502 958 579 881 657 71
1	2.500 000 000	−0.099 881 093 935 790 726 81
2	1.250 000 000	−0.085 663 295 657 346 816 28
3	0.625 000 000	0.315 111 728 317 131 689 63
4	0.937 500 000	0.034 301 373 178 712 055 16
5	1.093 750 000	−0.040 692 755 893 088 988 77
6	1.015 625 000	−0.007 631 301 536 946 475 19
22	1.000 000 238	−0.000 000 119 209 246 918 22
23	0.999 999 642	0.000 000 178 814 030 249 46
24	0.999 999 940	0.000 000 029 802 325 052 23

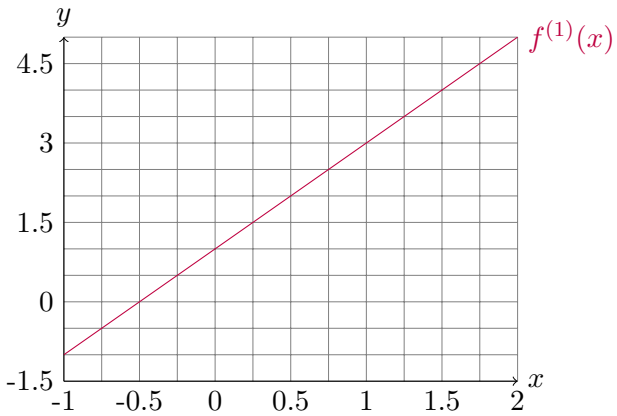
Metoda bisekcji szybciej niż metoda *złotego podziału* kończy swoje działania, zwracając satysfakcjonujący wynik, niemniej ciąg zwracanych przybliżeń zbiega wolniej ku rozwiązaniu, dlatego też połączenie tej metody z metodą Newtona nie da szybciej dobrego rowiązania - dlatego pominiemy ten przypadek, gdyż jest mało interesujący.

3.4 Funkcja 3.

Wzory i wykresy funkcji i pochodnych w podanym przedziale:



$$f(x) = x^2 + x - 1,\ x \in [-1, 2]$$



$$f^{(1)}(x) = 2x + 1$$

$$f^{(2)}(x) = 2$$

Metody *złotego podziału* i bisekcji w tym przypadku wykonują kolejno 36 i 24 iteracje, potwierdzają się poprzednie obserwacje - obydwie szybko wskazują ekstremum, jednakże długo się wykonują, gdyż algorytm czeka aż przedział stanie się odpowiednio "wąski". Metoda newtona daje odpowiedź w jednym kroku, gdyż $f^{(1)}$ jest funkcją liniową. Polecenia potrzebne do wykonania obliczeń:

- `./main -m golden_section -f 9 -s -1 2 -error 10e-8 -i 50`,
- `./main -m bisection -f 10 -s -1 2 -error 10e-8 -i 50`,
- `./main -m newton -f 10 -d 11 -x -1 -error 10e-8`.

4 Kompilacja i obsługa programu

4.1 Wymagania

Aby skompilować program należy spełnić następujące wymagania dotyczące oprogramowania:

- kompilator $G{+}{+}$ w wersji 4.7 lub późniejszej - kompilator musi obsługiwać standard $C^{++}11$,
- obecność narzędzia GNU Make

Powyższe wymagania powinny być automatycznie spełnione w każdej aktualnej dystrybucji GNU/Linux.

4.2 Kompilacja

Należy przejść do katalogu `prog` i wykonać polecenie `make` - kompilacja wykona się automatycznie. W pliku `Makefile` podane są polecenia, które należy wykonać aby skompilować program ręcznie.

4.3 Obsługa programu

Program uruchamiamy za pomoca pliku `main`, po jego nazwie podając ciąg bedący kombinacją ponizszych parametrów:

- `-f <nr_funkcji>` – za pomocą tego argumentu wybieramy jedną z dostępnych funkcji - liczba przyporzadkowana funkcji to jej liczba porządkowa z treści zadania ·3, dodanie 1 to pierwsza pochodna, dodanie 2 to druga pochodna,
- `-d <nr_funkcji>` – podobnie jak powyżej, tyle, że podajemy liczbę pochodnej,
- `-m <metoda>` – wybór jednej z metod:
  - `newton` – metoda newtona (obowiązkowe parametry wywołania to `-f -d -x`)
  - `regula_falsi` – *regula falsi* (obowiązkowe parametry to `-f -s`)
  - `bisection` – bisekcja (obowiązkowe parametry to `-f -s`)
  - `golden_section` – metoda *złotego podziału* (obowiązkowe parametry to `-f -s`)
  - `plot` – "wykres" funkcji (punkty) (obowiązkowe parametry to `-f -s -step`)
- `-p <n>` – wypisz wyniki z precyzją n cyfr po przecinku (domyślnie 20),

- `-s <a> <b>` – określ badany przedział od a do b,
- `-x <y>` – y jako punkt startowy,
- `-e (min|max)` – określ czy szukać lokalnego mininum czy maximum w przedziale (działa tylko z `-m golden_section`, domyślnie `min`),
- `-error <e>` – określ tolerancję błędu (domyślnie 10^{-10}),
- `-step <s>` – wielkość kroku przy obliczaniu punktów wykresu (działa tylko z `-m plot`, domyślnie 0.1),
- `-i <i>` – ilość iteracji (domyślnie 20),

Przykład: szukamy lokalnego minimum dla pierwszej funkcji z zadania, w podanym przedziale, za pomocą metody *złotego podziału*, z tolerancją błędu na poziomie 10^{-12}, maksymalnie 30 iteracjami i precyzją 25 liczb po przecinku.

```
./main -f 0 -s 0 1 -m golden_section -e 10e-12 -i 30 -p 25
```