

# MOwNiT Lab5 - sprawozdanie

---

Autor: Michał Flak

## Zadanie 1

### Polecenie

Uruchomić program `root_finding.tgz`

- \* Umieć odpowiedzieć na pytanie, co on robi.
- \* Narysować (np. za pomocą `gnuplot`) wykres funkcji, której miejsc zerowych szukamy.

### Rozwiązanie

Program szuka miejsca zerowego funkcji  $y=x^2-5$  metodą bisekcji w dziedzinie  $[0, 5]$ . Znajduje je w  $x=2.2357178$  po 12 iteracjach. Wykres przybliżony do  $[0, 2.5]$ :

```
x <- seq(0, 2.5, 0.1)
plot(x, x^2-5,
main="x^2-5",
type="l",
col="blue")
grid()
```

## Zadanie 2

### Polecenie

Zmienić program tak, aby znajdował pierwiastek metodą siecznych oraz Brent-Dekker'a.

- \* Porównać metody.
- \* Zamienić program tak, aby spróbował znaleźć pierwiastek równania  $x^2-2x+1=0$ .
- \* Narysować wykres tej funkcji za pomocą np. `gnuplot`.
- \* Wyjaśnić działanie programu - dlaczego nie może znaleźć miejsc zerowych dla tego równania?

### Porównanie metod:

1. Metoda Brenta to połączenie metody siecznych, metody bisekcji oraz odwrotnej interpolacji kwadratowej. Łączy niezawodność bisekcji z szybkością pozostałych metod.

$x^2-5$ : 6 kroków  $x=2.2360634$

$x^2-2x+1$ : ERROR: endpoints do not straddle  $y=0$

2. Metoda siecznych wymaga użycia solvera wykorzystującego pochodne funkcji. Jest to uproszczenie metody Newtona niewymagające liczenia pochodnej tak często.

$x^2-5$ : 5 kroków  $x=2.2360845$

$x^2-2x+1$ : 16 kroków  $x=1.0015480$

Wykres nowej funkcji  $x^2-2x+1$ :

```
x <- seq(0, 2.5, 0.1)
plot(x, x^2-2*x+1,
main="x^2-2*x+1",
type="l",
col="blue")
grid()
```

Wyjaśnienie zachowania programu:

Przy metodzie Brenta (jak również bisekcji) GSL oczekuje, że wartości funkcji w obu końcach przedziału będą miały różne znaki. W tym przypadku tak nie ma ponieważ  $\Delta=0$ , dlatego program wypisuje błąd.

## Zadanie 3

Polecenie

Napisać program szukający miejsc zerowych za pomocą metod korzystających z pochodnej funkcji.  
Czym różni się od poprzednich metod i dlaczego potrafią znaleźć pierwiastek równania  $x^2-2x+1=0$ ?

Porównać metodę Newtona, uproszczoną Newtona i Steffensona.

Opis metod

Program napisano w zadaniu 2 w celu korzystania z metody siecznych (czyli uproszczonej Newtona).

Metody nie potrzebują żeby wartości funkcji na krańcach przeszukiwanego przedziału były różnych znaków, dlatego można ich używać do znajdowania pierwiastków w funkcjach o wartościach lokalnie wyłącznie nieujemnych / niedodatnich (styk wykresu z osią  $X$  w punkcie).

Kod programu

```
#include <stdio.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
```

```
#include <gsl/gsl_roots.h>

#include "demo_fn.h"

int
main (void)
{
    int status;
    int iter = 0, max_iter = 100;
    const gsl_root_fdfsolver_type *T;
    gsl_root_fdfsolver *s;
    double x0, x = 5.0, r_expected = sqrt (5.0);
    gsl_function_fdf FDF;
    //struct quadratic_params params = {1.0, 0.0, -5.0};
    struct quadratic_params params = {1.0, -2.0, 1};

    FDF.f = &quadratic;
    FDF.df = &quadratic_deriv;
    FDF.fdf = &quadratic_fdf;
    FDF.params = &params;

    T = gsl_root_fdfsolver_secant;
    s = gsl_root_fdfsolver_alloc (T);
    gsl_root_fdfsolver_set (s, &FDF, x);

    printf ("using %s method\n",
            gsl_root_fdfsolver_name (s));

    printf ("%5s %10s %10s %10s\n",
            "iter", "root", "err", "err(est)");
    do
    {
        iter++;
        status = gsl_root_fdfsolver_iterate (s);
        x0 = x;
        x = gsl_root_fdfsolver_root (s);
        status = gsl_root_test_delta (x, x0, 0, 1e-3);

        if (status == GSL_SUCCESS)
            printf ("Converged:\n");

        printf ("%5d %10.7f %+10.7f %10.7f\n",
                iter, x, x - r_expected, x - x0);
    }
    while (status == GSL_CONTINUE && iter < max_iter);

    gsl_root_fdfsolver_free (s);
    return status;
}
```

Porównanie funkcji:

- Newton:  
12 kroków,  $x=1.0009766$
- Newton uproszczony (sieczne):  
16 kroków,  $x=1.0015480$
- Steffenson:  
4 kroki,  $x=1.0000000$