# Pierwszy projekt z labu

Zamiast w octave można zaprogramować odpowiednią funkcję i testy w Pascalu, C/C++, fortranie.

Projekt składa się z dwóch części:

1. Funkcja octave'a z metodą bisekcji: tzn. zaprogramować funkcję octave'a w m-pliku bisekcja.m z ***metodą bisekcji*** znajdującą przybliżenie zero funkcji ciągłej $f(x^*) = 0$ generującą ciąg zstępujących przedziałów $[a_k, b_k]$ o długości $(b_k - a_k) = 2^{-k}(b_0 - a_0)$ takich, że $sign(f(a_k)) \neq sign(f(b_k))$, za przybliżenie zera bierzemy $x_k = 0.5 * (b_k - a_k)$. W kolejnym kroku zastępujemy $a_k$, tzn. $b_{k+1} = b_k, a_{k+1} = x_k = 0.5 * (a_k + b_k)$ jeśli $sign(f(x_k)) == sign(f(a_k))$, w przeciwnym razie $a_{k+1} = a_k, b_{k+1} = x_k$. Zakładamy, że $sign(f(a_0)) \neq sign(f(b_0))$. Jako warunek stopu wziąć $|b_k - a_k| < TOL * |b_0 - a_0|$.

   Parametrami funkcji:

   ```
   function [x,a,b]=bisekcja(FCN,a0,b0,TOL)
   % kod funkcji
   end
   ```

   mają być:

   - FCN- 'wskaźnik' do funkcji $f$ (function handle),
   - a0,b0 - końce startowego odcinka
   - TOL - tolerancja - warunek stopu

   Funkcja ma zwracać:

   (a) obliczony przybliżony pierwiastek x

   (b) a,b - wektory zawierające lewe i odpowiednio prawe końce odcinków tzn $a = [a_0, \ldots, a_k]$ i $b = [b_0, \ldots, b_k]$.

   W razie gdyby znaki $f(a0)$ i $f(b0)$ były takie same to funkcja powinno zwrócić ostrzeżenie.

2. **Testy**:

   (a) **Test czy metoda działa** Przetestować na kilku prostych przykładach z TOL=1e−12 np. $f(x) = x^k - 2; a0 = 1, b0 = 2$ dla $k = 2, 4, 10, 100$, $f(x) = x + \sqrt{x} - 2; a0 = 0, b0 = 2$, $f(x) = \cos(x)$ z $a0 = 0, b0 = 2$. Sprawdzić czy $|x - x^*|$ jest mniejsze od zadanej tolerancji (o ile znamy dokładne $x^*$ ).

   (b) **Test czy metoda zwróci rozwiązanie na poziomie błędu zaokrągleń**
   Przetestować dla $f(x) = (x-c)^3$ z $a0 = 2; b0 = 3$ dla danej liczby $c \in (2, 3)$ np losowej. Funkcję $f$ obliczamy na 2 sposoby: raz wprost ze wzoru $(x - c)^3$ a raz z rozwinięcia dwumianu Newtona: $(x^3 - \ldots - c^3)$ dla bardzo małych tolerancji $TOL = 1e-13, 1e-14$ itp - sprawdzać czy kolejne $a_k < c$ a $b_k > c$.