1 Kod źródłowy

```
function [a, p] = ii2root(f, x0, x1, x2, K)
% Interpolacja odwrotna w zagadnieniu wyznaczania miesca zerowego funkcji.
% Parametry wej ciowe:
% f - wska nik do funkcji, kt rej miejsce zerowe szukamy
% x0, x1, x2 - pocz tkowe przybli enia miejsca zerowego
% K - maksymalna liczba iteracji
% Parametry wyj ciowe:
% a - ostatnie przybli enie miejsca zerowego funkcji f
% p - liczba faktycznie wykonanych krok w metody
% Autor: Andrii Voznesenskyi

% Startujemy z pocz tkowymi przybli eniami
x = [x0, x1, x2];
% problemhandler do uchwyt w postaci wektorowych z 1./x i 1/x
y = arrayfun(f, x);

% Obliczamy M
M = max(abs(y));

% Inicjalizujemy a
a = [];
w0 = LagrangeInterp(y, x, 0);
% Rozpoczynamy iteracje
for p = 1:K
    if checkValues(y)
        % Obliczamy w(0) za pomoc  interpolacji Lagrange'a
        w0 = LagrangeInterp(y, x, 0);
        % Aktualizujemy przybli enia
        x = [x(2), x(3), w0];
        y = arrayfun(f, x);
    else
        break;
    end
end
% Sprawdzenie tolerancji
if check_tolerance(f, w0, M)
  a = w0;
end
end % function
```

Listing 1: Pierwotny kod programu głównego

2 Używanie funkcji `arrayfun()`

W kodzie źródłowym, w linii 15 oraz 32 programu głównego, przedstawionego w *Listing 1*, została użyta funkcja wbudowana `arrayfun()` do przypisywania wartości funkcji $f(x)$ do wektora y. Jednak w praktyce jest to zbedne, ponieważ MATLAB jest w stanie wykonać przypisanie bez konieczności użycia tej funkcji. Używanie `arrayfun()` może zwiekszać złożoność obliczeniowa i nie jest konieczne w tym przypadku.

3 Obliczenie parametru w_0 zbedne

W kodzie źródłowym, w linii 22 programu głównego, przedstawionego w *Listing 1*, obliczana jest wartość interpolacji wielomianu Lagrange'a. Jednak w rzeczywistości, wartość ta nie jest używana ani zapisywana w żadnym miejscu programu. Dlatego, obliczenie parametru w_0 jest zbedne i można go całkowicie usunać, aby uprościć program.

Ponieważ wartość interpolacji w punkcie x nie jest używana ani wskazana do obliczania pred petla główna programu, warto całkowicie usunać ten fragment kodu, aby program dziawał poprawnie wzgledem specyfikacji.

4 Unikanie dzielania przez 0 w wielomianie Lagrange'a

W funkcji checkValues.m kodu żródłowego warto sprawdzić warunek na to aby każda wartość parametru wejściowego y była różna od 0. Warunek `all(y ≠ 0)` jest w tym przypadku ważny, aby zapobiec dzieleniu przez zero. Jeśli `y` zawiera zero, funkcja `LagrangeInterp` wykonałaby dzielenie przez zero, co spowodowałoby bład. W funkcji `LagrangeInterp` implementujesz interpolacje Lagrange'a, która jest opisana wzorem:

$$L(x) = \sum_{j=0}^{n} y_j * l_j(x)$$

gdzie $l_j(x)$ to wielomian bazowy Lagrange'a:

$$l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}$$

W zaimplementowanej funkcji `x` odpowiada `X`, `y` odpowiada `Y`, oblicza sie wartość wielomianu w punkcie `x`.

Jeśli `y ≠ 0` nie jest prawda dla wszystkich `y` (co jest równoznaczne z stwierdzeniem, że istnieje jakiekolwiek `y` takie, że `y == 0`), to `(X(i) - X(j))` może być równe zero. Ponieważ dzielenie przez te wartość podczas obliczania `p`, prowadzi do dzielenia przez zero.

Dodanie warunku `all(y ≠ 0)` przed wejściem do petli zapewnia, że oblicza sie interpolacje Lagrange'a tylko wtedy, gdy wszystkie wartości `y` sa różne od zera. Jeśli jakiekolwiek `y` wynosi zero, funkcja `checkValues(y)` zwraca `false`, co zatrzymuje iteracje w funkcji `ii2root` z powodu warunku `if checkValues(y)`. W ten sposób sie unika dzielenia przez zero.

Warto również zauważyć, że interpolacja Lagrange'a wymaga, aby wartości `x` (które w tym kontekście sa wartościami `y`, ponieważ wykonujemy interpolacje odwrotna) były różne, dlatego sprawdza sie `uniqueCheck = all(diff(sort(y)) ≠ 0);`. Ten warunek

zapewnia, że żadne dwie wartości y nie sa takie same. Wiec, jeśli y zawiera zero, nie tylko zapobiega potencjalnym błedom dzielenia przez zero, ale również zapewnia spełnienie wymogu unikalności wartości x dla interpolacji Lagrange'a. W kodzie *Listing 2* znajduje sie poprawiona wersja funkcji `checkValues.m`

```
function valid = checkValues(y)
% Sprawd , czy wszystkie elementy y s  rzeczywiste,
   sko czone i unikalne
% Wej cie:
% y - wektor warto ci do sprawdzenia
% Wyj cie:
% valid - warto  logiczna wskazuj ca , czy wszystkie
   elementy y s
% rzeczywiste, sko czone i parami r  ne
% Autor: Andrii Voznesenskyi

% Czy wszystkie elementy y s  rzeczywiste
realCheck = all(isreal(y));
% Czy wszystkie elementy y s  sko czone
finiteCheck = all(isfinite(y));
% Czy wszystkie elementy y s  unikalne
% uniqueCheck = numel(unique(y)) == numel(y);
uniqueCheck = all(diff(sort(y)) ~= 0) && all(y ~= 0);
% Czy wszystkie sprawdzenia przesz y pomy lnie
valid = (realCheck && finiteCheck && uniqueCheck);
end % function
```

Listing 2: Poprawiony kod funkcji checkValues

5 Poprawiony kod programu głównego z uwzglednionymi uwagami

Poprawiony kod programu głównego z uwzglednieniem powuższych uwag został umieszczony w kodzie *Listing 3*.

```
function [a, p] = ii2root(f, x0, x1, x2, K)
% Interpolacja odwrotna w zagadnieniu wyznaczania miesca
   zerowego funkcji.
% Parametry wej ciowe:
% f - wska nik do funkcji, kt rej miejsce zerowe szukamy
% x0, x1, x2 - pocz tkowe przybli enia miejsca zerowego
% K - maksymalna liczba iteracji
% Parametry wyj ciowe:
% a - ostatnie przybli enie miejsca zerowego funkcji f
% p - liczba faktycznie wykonanych krok w metody
% Autor: Andrii Voznesenskyi

% Startujemy z pocz tkowymi przybli eniami
```

```
13 x = [x0, x1, x2];
14 % problemhandler do uchwyt w postaci wektorowych z 1./x i 1/
       x
15 y = f(x);
16
17 % Obliczamy M
18 M = max(abs(y));
19
20 % Inicjalizujemy a
21 a = [];
22
23 % Rozpoczynamy iteracje
24 for p = 1:K
25     if checkValues(y)
26         % Obliczamy w(0) za pomoc  interpolacji Lagrange'a
27         w0 = LagrangeInterp(y, x, 0);
28         % Aktualizujemy przybli enia
29         x = [x(2), x(3), w0];
30         y = f(x);
31     else
32         break;
33     end
34 end
35 % Sprawdzenie tolerancji
36 if check_tolerance(f, w0, M)
37   a = w0;
38 end
39 end % function
```

Listing 3: Poprawiony kod programu głównego