Obliczanie pierwiastków wielomianu metodą połowienia

Witold Kupś 127088

1. Zastosowanie

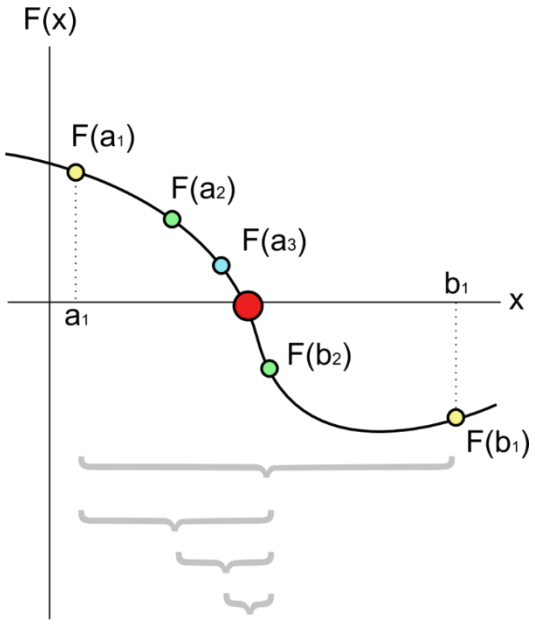
Jest to jedna z metod rozwiązywania równań nieliniowych. Opiera się ona na twierdzeniu Bolzana-Cauchy'ego:

Jeżeli funkcja ciągła $f(\mathrm{x})$ *ma na końcach przedziału domkniętego wartości różnych znaków, to wewnątrz tego przedziału, istnieje co najmniej jeden pierwiastek równania* $f(\mathrm{x}) = 0$.

2. Opis metody

Aby można było zastosować metodę równego podziału, muszą być spełnione założenia:

1. Funkcja $f(x)$ jest ciągła i określona w przedziale domkniętym $< a; b >$
2. Funkcja przyjmuje różne znaki na końcach przedziału: $f(\mathrm{a})f(b) < 0$



Gdy funkcja $f(x)$ spełnia powyższe warunki, w przedziale $< a; b >$ zagwarantowane jest istnienie pierwiastka i możemy go wyszukać algorytmem połowienia (bisekcji).

Zasada jest następująca:

- Wyznaczamy punkt x_0 jako środek przedziału $< a; b >$ zgodnie ze wzorem:

$$x_0 = \frac{a+b}{2}$$

- Obliczamy wartość funkcji w punkcie x_0. Jeśli długość przedziału jest mniejsza od założonej dokładności wyliczeń pierwiastka, to wartość x_0 jest poszukiwanym przybliżeniem. Kończymy algorytm. W przeciwnym razie sprawdzamy, czy $f(x_0)$ znajduje się dostatecznie blisko 0:

$$|f(x_0)| < \varepsilon$$

- Jeśli nierówność jest spełniona, to x_0 jest poszukiwaną wartością pierwiastka. Zwracamy wynik i kończymy algorytm. W przeciwnym razie za nowy przedział poszukiwań pierwiastka przyjmujemy tą połówkę $< a; x_0 >$ lub $< x_0; b >$, w której funkcja zmienia znak na krańcach. Algorytm powtarzamy od początku dotąd, aż znajdziemy pierwiastek lub przedział $< a; b >$ osiągnie założoną długość (może to być również epsilon). Wtedy kończymy zwracając ostatnio wyliczone x_0.

3. Wywołanie funkcji

Aby wywołać funkcję należy utworzyć obiekt `Bisection<V>` , gdzie V jest typem, jakiego chcemy użyć w obliczeniach. Konieczne jest jednak, by została dla niego zdefiniowana klasa `NumberWrapper<V>`. Konstruktor obiektu `Bisection` ma następującą semantykę:

```
Bisection(
    Polynomial<V> polynomial,
    NumberWrapper<V> scopeEpsilon,
    NumberWrapper<V> resultEpsilon
)
```

Następnym krokiem jest wywołanie metody `findZero` wykonywującą właściwy algorytm. Jej nagłówek jest następujący:

```
Result<V> findZero(
    Interval<V> scope,
    Integer allowedIterations
)
```

4. Dane

`polynomial` – wielomian składający się z elementów posiadających wykładniki będące obiektami typu `String` (będący jednak liczbą całkowitą), oraz współczynników opakowanych w typ `NumberWrapper<V>`. Wewnętrzna implementacja współczynnika elementu jest oparta o przedział, który dla *zwykłej arytmetyki* jest ***przedziałem punktowym***, natomiast dla *arytmetyki przedziałowej* staje się ***faktycznym przedziałem*** zawierającym maszynowe odpowiedniki danych wartości (górna oraz dolna granica, lub dokładna wartość, jeśli jest ona reprezentowana maszynowo). Możliwe jest również wprowadzenie jako współczynnik przedziału żądanego przez użytkownika. Nie jest natomiast możliwe zadanie wartości bezwzględnej, pierwiastków czy liczb zespolonych; jedyne akceptowalne wartości to liczby rzeczywiste w formie prostej (1.73… z możliwym rozszerzeniem „E”) oraz przedziały je zawierające.

`scopeEpsilon` – maksymalna akceptowalna szerokość otrzymanego przedziału argumentów, w jakim funkcja ma pierwiastek.

`resultEpsilon` – maksymalna akceptowalna dokładność otrzymanego wyniku dla argumentów będących potencjalnymi miejscami zerowymi.

`scope` – przedział, w jakim poszukujemy rozwiązania.

`allowedIterations` – maksymalna ilość iteracji, jakie algorytm może wykonać. Można również zadać wartość `null` – wtedy parametr nie jest brany pod uwagę.

5. Wyniki

`result` – instancja klasy `Result<V>` składająca się z następujących pól:

`Polynomial<V> polynomial` – wielomian, na jakim wykonywane były obliczenia – może być on nieznacznie zmieniony; przykładowo dla zadanego współczynnika –[4, 6] wewnętrzna reprezentacja to przedział [-6, -4] – w takiej też postaci zostanie on zaprezentowany.

`Interval<V> result` – wynik dla przedziału argumentów, w jakim znaleziono pierwiastek wielomianu. Jest to przedział reprezentujący odpowiednio minimalną (≤ 0) oraz maksymalną (≥ 0) możliwą wartość otrzymaną dla zadanego przedziału argumentów. Prezentowana jest również szerokość danego przedziału.

`Interval<V> scope` – otrzymany przedział argumentów (wraz z jego szerokością), w jakim znajduje się miejsce zerowe wielomianu. Może być pojedynczą wartością, jeżeli znaleziono dokładną wartość pierwiastka.

`int iteration` – iteracja, w której otrzymano prezentowany wynik.

`String reason` – powód zakończenia działania algorytmu. Może przyjąć następujące wartości:

`narrow enough interval` – otrzymany przedział argumentów, w których może znajdować się pierwiastek wielomiany osiągnął wystarczającą szerokość.

`good enough point` – znaleziono punktowy argument będącym miejscem zerowym wielomianu.

`exceeded allowed iterations numer` – przekroczono zadaną liczbę iteracji; prezentowany wynik jest wartościami osiągniętymi w zadanej iteracji.

6. Inne parametry

Funkcja `findZero` może wyrzucić wyjątek czasu wykonania (`RuntimeException`) w przypadku, gdy użytkownik zadał ilość iteracji mniejszą niż 1, lub gdy dla zadanego przedziału argumentów nie jest możliwe znalezienie miejsca zerowego (nie spełniony jeden z warunków działania algorytmu). W treści wyjątku uwzględniony został powód w formie ciągu znaków – dla braku spełnienia warunków działania algorytmu dodatkowo prezentowany jest przedział będący powodem wystąpienia wyjątku, oraz iteracja w której wystąpił.

7. Typy parametrów

V – typ, na którym wykonywane są obliczenia. W celu wykonywania obliczeń konieczne jest zdefiniowanie klasy rozszerzającej `NumberWrapper<V>`. Domyślnie zaimplementowane zostały klasy DecimalWrapper dla typu BigDecimal (precyzja arbitralna) oraz FloatWrapper dla typu BigFloat, będący odpowiednikiem typu mpfr w języku C++. Z tego też powodu konieczna jest instalacja biblioteki natywnej mpfr skompilowanej pod daną architekturę (domyślnie dostępna wersja linux86_64).

`NumberWrapper<V>: scopeEpsilon, resultEpsilon`

Uwaga: wartości reprezentowane wewnątrz przedziałów `Interval<V>` są również typu `NumberWrapper<V>`:

```
class Interval<V extends Number & Comparable<V>> {
    private NumberWrapper<V> lower; // dolna granica
    private NumberWrapper<V> upper; // górna granica
    private NumberWrapper<V> delta; // szerokość przedziału
}
```

`Interval<V>:` `scope, result`

Uwaga: wartości współczynników w wielomianie (`polynomial`) są również typu `Interval<V>`:

```
class Element<V extends Number & Comparable<V>> {
    private Interval<V> factor; // współczynnik
    private String exponent; // wykładnik
}
```

`Polynomial<V>:` `polynomial`

`Integer:` `allowedIterations // może przyjmować wartość null`

`int:` `iteration`

`String:` `reason`

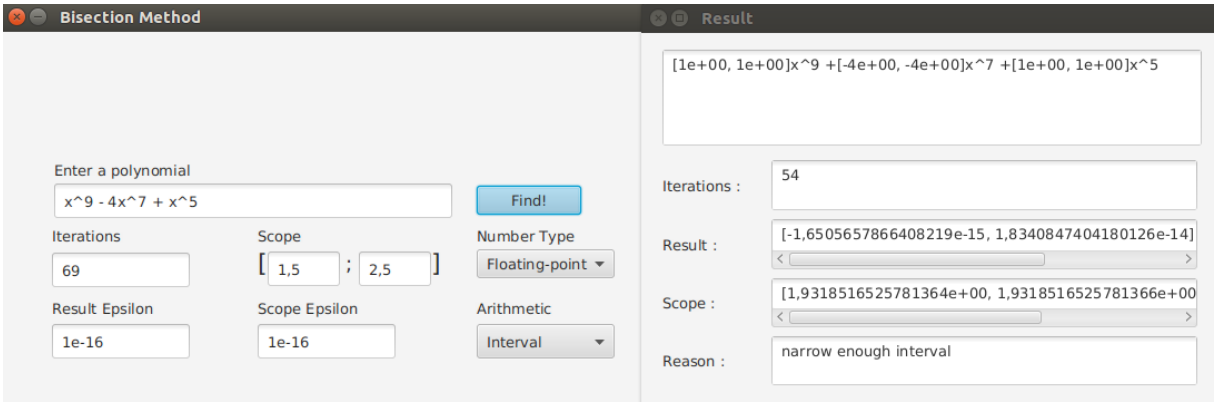
8. Identyfikator nielokalny

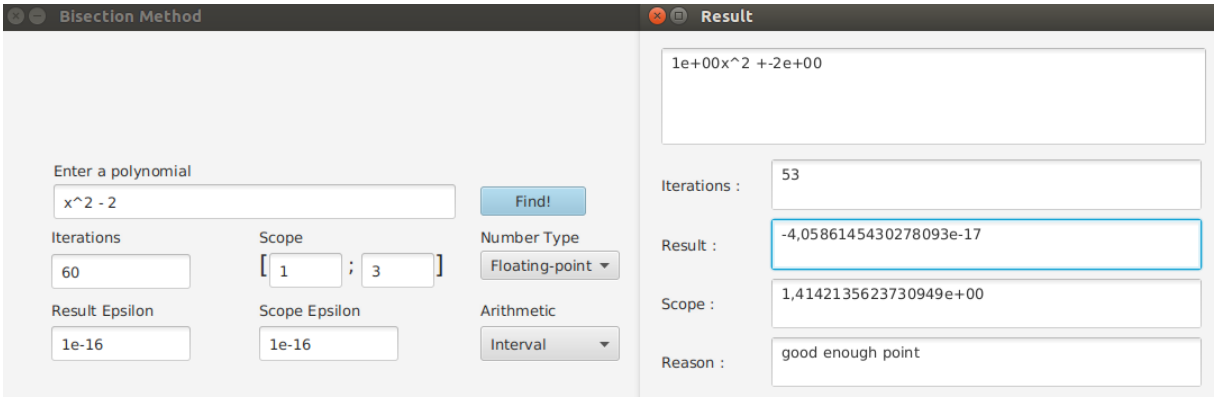
Brak

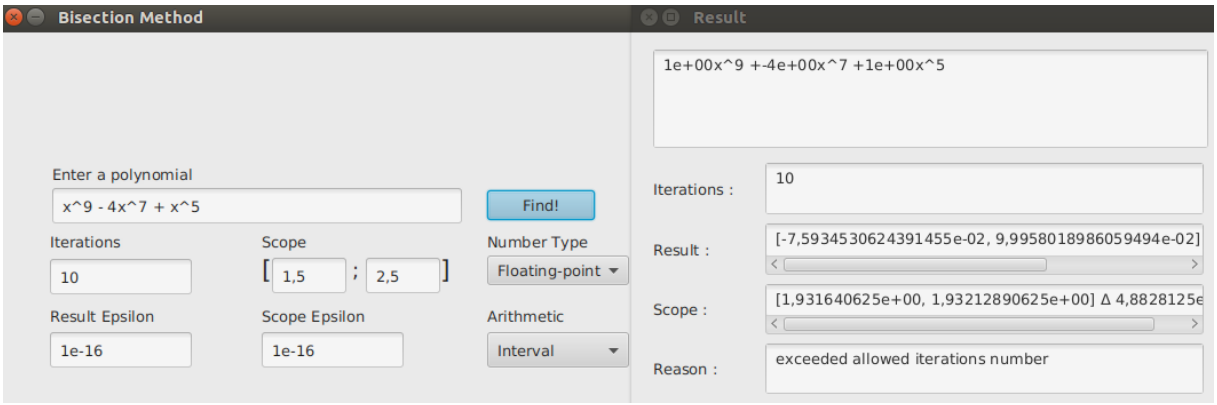
9. Treść funkcji

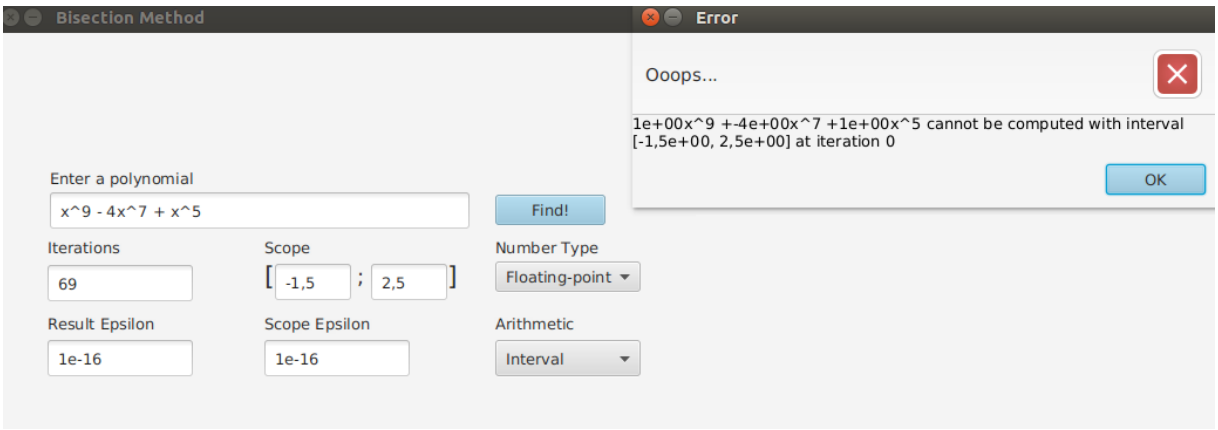
```
Result<V> findZero(Interval<V> scope, Integer allowedIterations) {
   if (allowedIterations != null && allowedIterations < 1) {
       throw new RuntimeException("Allowed iterations less than one");
   }
   for (int iteration = 0; allowedIterations == null || iteration <= allowedIterations; iteration++) {
       if (!polynomial.canBeComputedWith(scope)) {
           throw new RuntimeException(
                  polynomial + " cannot be computed with interval " + scope + " at iteration " +
                  iteration);
       } else {
           if (scope.isNarrowerThan(scopeEpsilon)) {
               final Interval<V> solution = polynomial.countForInterval(scope);
               return new Result<>(polynomial, iteration, solution, scope,
               Result.REASON_NARROW_SCOPE);
           } else {
               final Interval<V> centerPoint = scope.getCenterPoint();
               final Interval<V> result = polynomial.countForInterval(centerPoint);
               if (result.isLowerOrEqualValueWithAbs(resultEpsilon)) {
                   return new Result<>(polynomial, iteration, result, centerPoint,
                                       Result.REASON_POINT);
               } else {
                   scope = scope.findSubInterval(polynomial);
               }
           }
       }
   }
   final Interval<V> solution = polynomial.countForInterval(scope);
   return new Result<>(polynomial, allowedIterations, solution, scope,
                       Result.REASON_EXCEEDED_ITERATIONS);
}
```

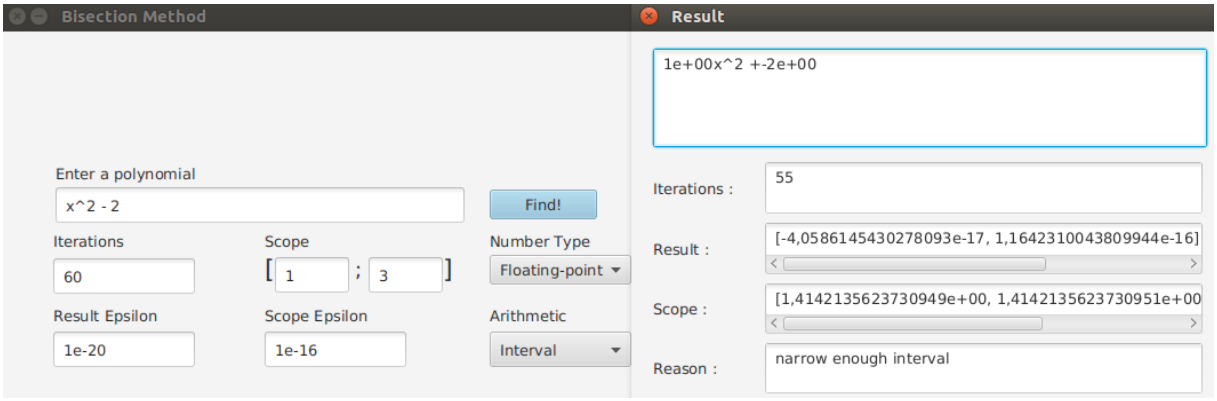
10. Przykłady

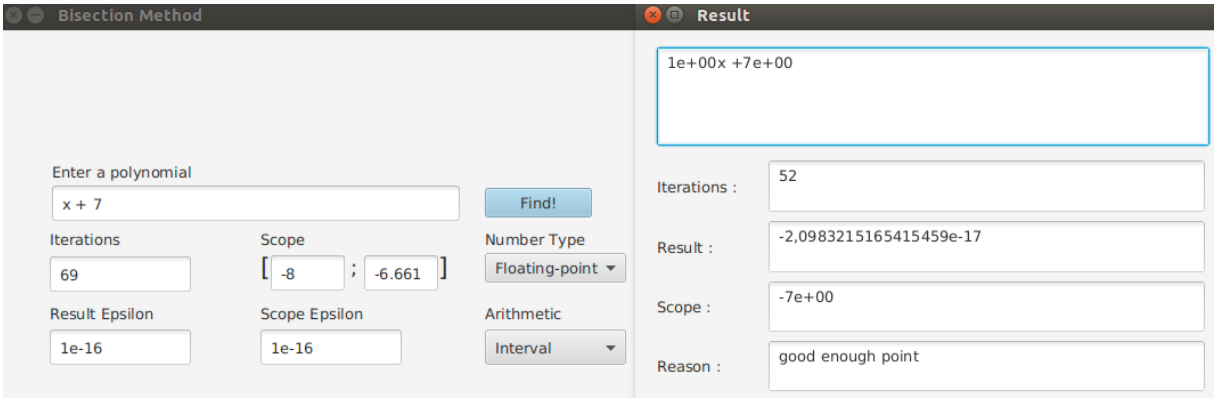


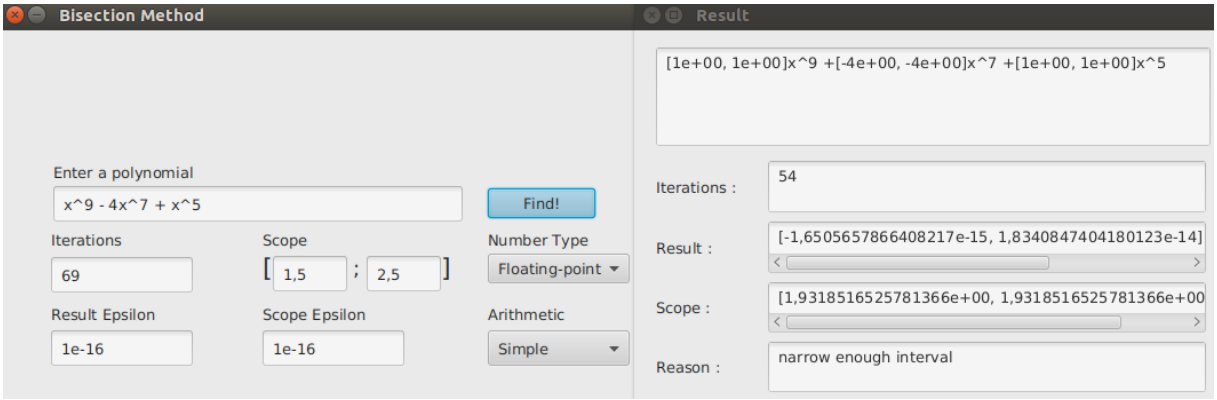


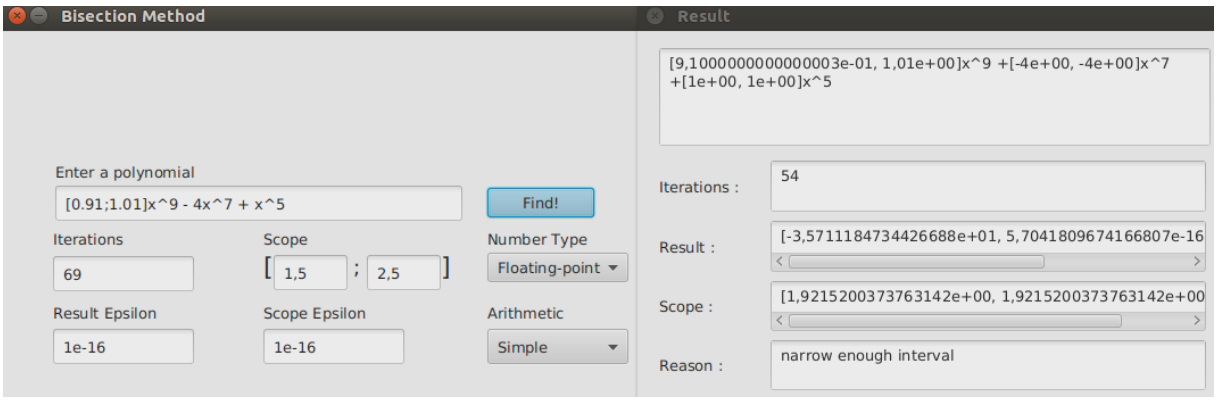


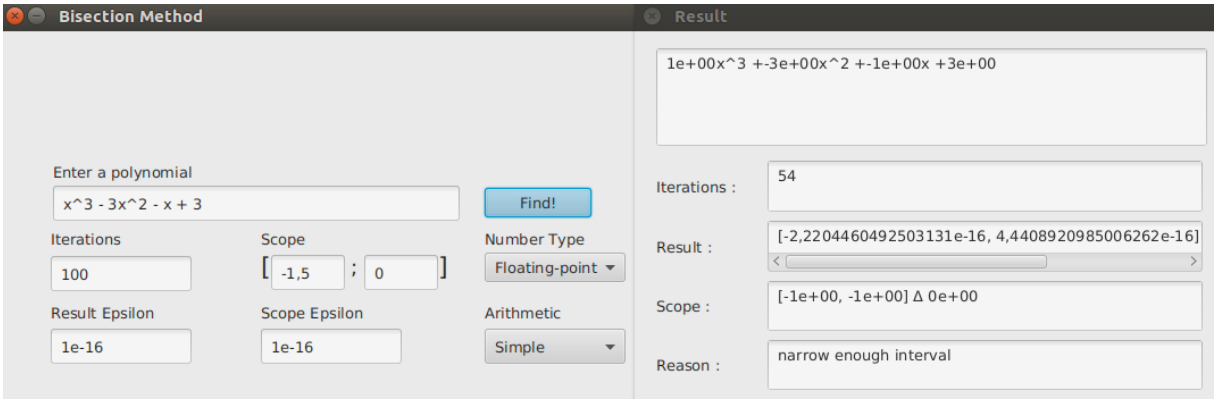


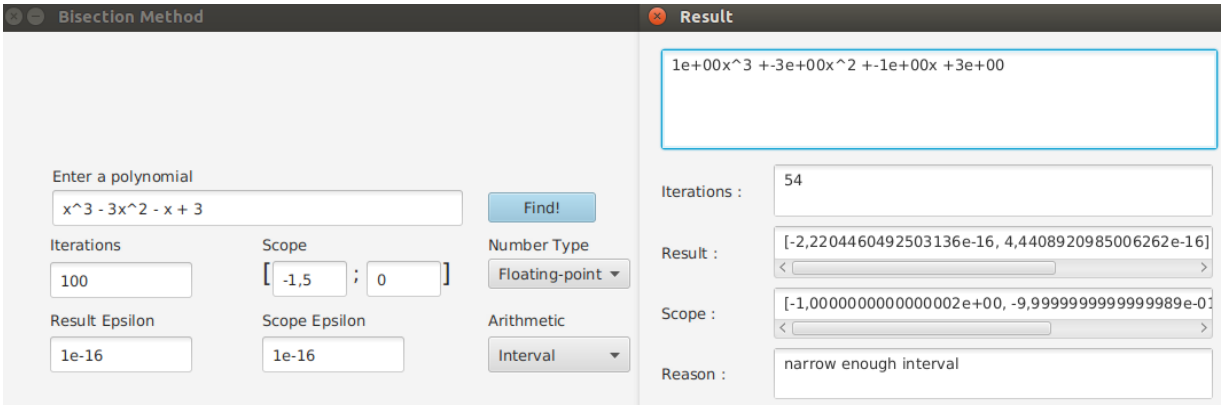

Bisection Method
Enter a polynomial
x^2 - 2
Find!
Iterations
60
Scope
[1 ; 3]
Number Type
Floating-point
Result Epsilon
1e-20
Scope Epsilon
1e-16
Arithmetic
Interval
Result
1e+00x^2 +-2e+00
Iterations :
55
Result :
[-4,0586145430278093e-17, 1,1642310043809944e-16]
Scope :
[1,4142135623730949e+00, 1,4142135623730951e+00
Reason :
narrow enough interval


Bisection Method
Enter a polynomial
x + 7
Find!
Iterations
69
Scope
[-8 ; -6.661]
Number Type
Floating-point
Result Epsilon
1e-16
Scope Epsilon
1e-16
Arithmetic
Interval
Result
1e+00x +7e+00
Iterations :
52
Result :
-2,0983215165415459e-17
Scope :
-7e+00
Reason :
good enough point


Bisection Method
Enter a polynomial
x^9 - 4x^7 + x^5
Find!
Iterations
69
Scope
[1,5 ; 2,5]
Number Type
Floating-point
Result Epsilon
1e-16
Scope Epsilon
1e-16
Arithmetic
Simple
Result
[1e+00, 1e+00]x^9 +[-4e+00, -4e+00]x^7 +[1e+00, 1e+00]x^5
Iterations :
54
Result :
[-1,6505657866408217e-15, 1,8340847404180123e-14]
Scope :
[1,9318516525781366e+00, 1,9318516525781366e+00
Reason :
narrow enough interval


Bisection Method
Enter a polynomial
[0.91;1.01]x^9 - 4x^7 + x^5
Find!
Iterations
69
Scope
[1,5 ; 2,5]
Number Type
Floating-point
Result Epsilon
1e-16
Scope Epsilon
1e-16
Arithmetic
Simple
Result
[9,1000000000000003e-01, 1,01e+00]x^9 +[-4e+00, -4e+00]x^7
+[1e+00, 1e+00]x^5
Iterations :
54
Result :
[-3,5711184734426688e+01, 5,7041809674166807e-16
Scope :
[1,9215200373763142e+00, 1,9215200373763142e+00
Reason :
narrow enough interval


Bisection Method
Enter a polynomial
x^3 - 3x^2 - x + 3
Find!
Iterations
100
Scope
[-1,5 ; 0]
Number Type
Floating-point
Result Epsilon
1e-16
Scope Epsilon
1e-16
Arithmetic
Simple
Result
1e+00x^3 +-3e+00x^2 +-1e+00x +3e+00
Iterations :
54
Result :
[-2,2204460492503131e-16, 4,4408920985006262e-16]
Scope :
[-1e+00, -1e+00] Δ 0e+00
Reason :
narrow enough interval


Bisection Method
Enter a polynomial
x^3 - 3x^2 - x + 3
Find!
Iterations
100
Scope
[-1,5 ; 0]
Number Type
Floating-point
Result Epsilon
1e-16
Scope Epsilon
1e-16
Arithmetic
Interval
Result
1e+00x^3 +-3e+00x^2 +-1e+00x +3e+00
Iterations :
54
Result :
[-2,2204460492503136e-16, 4,4408920985006262e-16]
Scope :
[-1,0000000000000002e+00, -9,9999999999999989e-01
Reason :
narrow enough interval