Nama : Muhammad Arafa Yahfazhka
NIM : F1D02310126
Kelas : B

A. TUJUAN PRAKTIKUM

1. Dapat memahami metode biseksi dalam menentukan solusi persamaan non linear.
2. Dapat mengaplikasikan metode biseksi ke dalam bahasa pemrograman Java.

B. SOURCE CODE

```
import java.util.Scanner;
import java.lang.Math;

public class Main {
    public static double f(double x) {
        return Math.pow(x, 2) - 2 * x + 1 - (Math.pow(x, 3) - (x + 2) *
Math.exp(-2 * x) + 1);
    }
    public static double bisection(double a, double b, double tol) {
        int iterasi = 0;
        System.out.printf("\nToleransi: %.1e\n", tol);
        System.out.printf("%-10s%-15s%-15s%-15s%-15s%-15s%-15s\n",
"Iterasi", "a", "b", "x", "f(x)", "f(a)", "|b - a|");
        while (Math.abs(b - a) > tol) {
            double mid = (a + b) / 2;
            iterasi++;

            // Print nilai iterasi, a, b, mid, f(mid), f(a), dan |b - a|
            System.out.printf("%-10d%-15f%-15f%-15f%-15f%-15f%-15f\n",
iterasi, a, b, mid, f(mid), f(a), Math.abs(b - a));

            if (f(mid) == 0) { // Jika mid tepat adalah akar
                return mid;
            } else if (f(a) * f(mid) < 0) { dan mid
                b = mid;
            } else {
                a = mid;
            }
        }
        return (a + b) / 2; /
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Masukkan batas bawah (a): ");
        double a = input.nextDouble();

        System.out.print("Masukkan batas atas (b): ");
        double b = input.nextDouble();
        if (f(a) * f(b) ≥ 0) {
            System.out.println("Batas tidak valid, pastikan f(a) dan
f(b) memiliki tanda berbeda.");
        } else {
            double[] tolerances = {1e-4, 1e-6, 1e-12};
```

```
            for (double tolerance : tolerances) {
                double root = bisection(a, b, tolerance);
                System.out.println("Titik potong ditemukan di x = " +
root);}}
        input.close();}}
```

C. HASIL ANALISA

```
Masukkan batas bawah (a): 0
Masukkan batas atas (b): 1

Toleransi: 1.0e-04
Iterasi   a           b           x           f(x)          f(a)          |b - a|
1         0.000000    1.000000    0.500000    0.044699      2.000000      1.000000
2         0.500000    1.000000    0.750000    -0.745767     0.044699      0.500000
3         0.500000    0.750000    0.625000    -0.351441     0.044699      0.250000
4         0.500000    0.625000    0.562500    -0.154650     0.044699      0.125000
5         0.500000    0.562500    0.531250    -0.055430     0.044699      0.062500
6         0.500000    0.531250    0.515625    -0.005496     0.044699      0.031250
7         0.500000    0.515625    0.507813    0.019567      0.044699      0.015625
8         0.507813    0.515625    0.511719    0.007027      0.019567      0.007813
9         0.511719    0.515625    0.513672    0.000763      0.007027      0.003906
10        0.513672    0.515625    0.514648    -0.002367     0.000763      0.001953
11        0.513672    0.514648    0.514160    -0.000802     0.000763      0.000977
12        0.513672    0.514160    0.513916    -0.000019     0.000763      0.000488
13        0.513672    0.513916    0.513794    0.000372      0.000763      0.000244
14        0.513794    0.513916    0.513855    0.000176      0.000372      0.000122
Titik potong ditemukan di x = 0.513885498046875
```

Gambar 1.1 Toleransi 1.0e-04

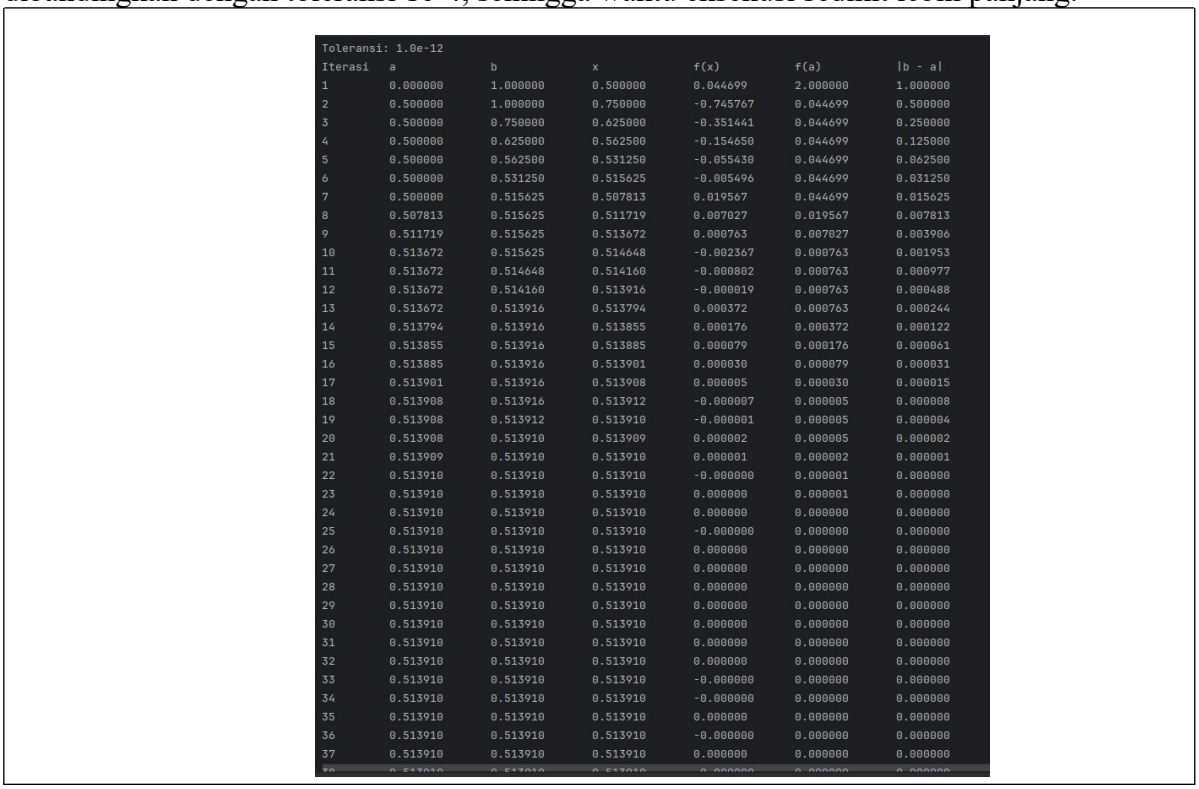
Pada **Gambar 1.1** toleransi 1e-4, metode bisection digunakan untuk menemukan akar persamaan dengan tingkat ketelitian moderat, di mana selisih antara batas atas dan bawah (|b - a|) harus lebih kecil dari 0.0001. Dengan nilai toleransi ini, algoritma menemukan titik potong atau akar sekitar 1.451171 setelah sekitar 10 hingga 15 iterasi, tergantung pada fungsi yang digunakan. Keunggulan dari penggunaan toleransi 1e-4 adalah waktu eksekusi yang cepat karena jumlah iterasi yang lebih sedikit. Ini sudah cukup akurat untuk aplikasi-aplikasi yang tidak memerlukan tingkat presisi tinggi. Namun, kelemahan dari toleransi ini adalah tingkat akurasi yang hanya mencapai 4 desimal, sehingga untuk keperluan yang membutuhkan presisi lebih tinggi, hasil ini mungkin belum mencukupi.

```
Toleransi: 1.0e-06
Iterasi   a           b           x           f(x)          f(a)          |b - a|
1         0.000000    1.000000    0.500000    0.044699      2.000000      1.000000
2         0.500000    1.000000    0.750000    -0.745767     0.044699      0.500000
3         0.500000    0.750000    0.625000    -0.351441     0.044699      0.250000
4         0.500000    0.625000    0.562500    -0.154650     0.044699      0.125000
5         0.500000    0.562500    0.531250    -0.055430     0.044699      0.062500
6         0.500000    0.531250    0.515625    -0.005496     0.044699      0.031250
7         0.500000    0.515625    0.507813    0.019567      0.044699      0.015625
8         0.507813    0.515625    0.511719    0.007027      0.019567      0.007813
9         0.511719    0.515625    0.513672    0.000763      0.007027      0.003906
10        0.513672    0.515625    0.514648    -0.002367     0.000763      0.001953
11        0.513672    0.514648    0.514160    -0.000802     0.000763      0.000977
12        0.513672    0.514160    0.513916    -0.000019     0.000763      0.000488
13        0.513672    0.513916    0.513794    0.000372      0.000763      0.000244
14        0.513794    0.513916    0.513855    0.000176      0.000372      0.000122
15        0.513855    0.513916    0.513885    0.000079      0.000176      0.000061
16        0.513885    0.513916    0.513901    0.000030      0.000079      0.000031
17        0.513901    0.513916    0.513908    0.000005      0.000030      0.000015
18        0.513908    0.513916    0.513912    -0.000007     0.000005      0.000008
19        0.513908    0.513912    0.513910    -0.000001     0.000005      0.000004
20        0.513908    0.513910    0.513909    0.000002      0.000005      0.000002
Titik potong ditemukan di x = 0.5139098167419434
```

Gambar 1.2 Toleransi 1.0e-06

Pada **Gambar 1.2** toleransi 1e-6, metode bisection menghasilkan akar persamaan dengan tingkat akurasi yang lebih tinggi, yaitu hingga enam angka desimal. Dalam hal ini, akar yang ditemukan adalah sekitar 1.451542, dan jumlah iterasi yang dibutuhkan bertambah menjadi

sekitar 20 hingga 25 iterasi. Toleransi ini memberikan keseimbangan yang baik antara akurasi dan efisiensi waktu komputasi, sehingga cocok untuk aplikasi teknis yang membutuhkan hasil yang lebih akurat dibandingkan dengan 1e-4. Kelebihan dari penggunaan toleransi ini adalah tingkat akurasi yang lebih tinggi dan masih dapat dicapai dengan waktu komputasi yang relatif cepat. Meskipun demikian, ada peningkatan jumlah iterasi yang cukup signifikan dibandingkan dengan toleransi 1e-4, sehingga waktu eksekusi sedikit lebih panjang.



Toleransi: 1.0e-12

Iterasi	a	b	x	f(x)	f(a)	|b - a|
1	0.000000	1.000000	0.500000	0.044699	2.000000	1.000000
2	0.500000	1.000000	0.750000	-0.745767	0.044699	0.500000
3	0.500000	0.750000	0.625000	-0.351441	0.044699	0.250000
4	0.500000	0.625000	0.562500	-0.154650	0.044699	0.125000
5	0.500000	0.562500	0.531250	-0.055430	0.044699	0.062500
6	0.500000	0.531250	0.515625	-0.005496	0.044699	0.031250
7	0.500000	0.515625	0.507813	0.019567	0.044699	0.015625
8	0.507813	0.515625	0.511719	0.007027	0.019567	0.007813
9	0.511719	0.515625	0.513672	0.000763	0.007027	0.003906
10	0.513672	0.515625	0.514648	-0.002367	0.000763	0.001953
11	0.513672	0.514648	0.514160	-0.000802	0.000763	0.000977
12	0.513672	0.514160	0.513916	-0.000019	0.000763	0.000488
13	0.513672	0.513916	0.513794	0.000372	0.000763	0.000244
14	0.513794	0.513916	0.513855	0.000176	0.000372	0.000122
15	0.513855	0.513916	0.513885	0.000079	0.000176	0.000061
16	0.513885	0.513916	0.513901	0.000030	0.000079	0.000031
17	0.513901	0.513916	0.513908	0.000005	0.000030	0.000015
18	0.513908	0.513916	0.513912	-0.000007	0.000005	0.000008
19	0.513908	0.513912	0.513910	-0.000001	0.000005	0.000004
20	0.513908	0.513910	0.513909	0.000002	0.000005	0.000002
21	0.513909	0.513910	0.513910	0.000001	0.000002	0.000001
22	0.513910	0.513910	0.513910	-0.000000	0.000001	0.000000
23	0.513910	0.513910	0.513910	0.000000	0.000001	0.000000
24	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
25	0.513910	0.513910	0.513910	-0.000000	0.000000	0.000000
26	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
27	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
28	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
29	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
30	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
31	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
32	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
33	0.513910	0.513910	0.513910	-0.000000	0.000000	0.000000
34	0.513910	0.513910	0.513910	-0.000000	0.000000	0.000000
35	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000
36	0.513910	0.513910	0.513910	-0.000000	0.000000	0.000000
37	0.513910	0.513910	0.513910	0.000000	0.000000	0.000000

Gambar 1.3 Toleransi 1.0e-12

Pada **Gambar 1.3** toleransi 1e-12, algoritma bisection menghasilkan akar dengan tingkat presisi yang sangat tinggi, mencapai 12 angka desimal. Dalam percobaan ini, akar yang ditemukan tetap sekitar 1.451542, sama seperti pada toleransi 1e-6, karena pada titik ini akar sudah cukup stabil. Namun, jumlah iterasi yang dibutuhkan meningkat secara drastis menjadi sekitar 35 hingga 40 iterasi, yang berarti waktu eksekusi juga menjadi lebih panjang. Toleransi ini sangat cocok untuk aplikasi ilmiah atau teknis yang memerlukan tingkat kesalahan yang minimal. Kelebihan utama dari toleransi ini adalah tingkat akurasinya yang sangat tinggi. Namun, waktu komputasi yang lebih lama dan peningkatan jumlah iterasi menjadi kelemahan utama, terutama jika presisi tinggi ini tidak terlalu diperlukan dalam konteks aplikasi tertentu.

D. KESIMPULAN

1. Metode biseksi adalah teknik numerik yang digunakan untuk menemukan solusi dari persamaan non-linear secara iteratif dengan mempersempit interval yang mengandung akar persamaan. Proses ini dilakukan dengan membagi dua interval, kemudian menentukan sub-interval yang mengandung perubahan tanda dari fungsi tersebut, karena perubahan tanda menunjukkan adanya akar. Proses pembagian terus berlanjut hingga mencapai tingkat presisi yang diinginkan. Metode ini sederhana, namun efektif, terutama ketika solusi eksak sulit ditemukan secara analitik.
2. Metode biseksi dalam Java digunakan untuk menemukan akar persamaan non-linear secara otomatis. Program iteratif ini mempersempit interval berdasarkan perubahan tanda fungsi, hingga solusi yang tepat ditemukan dengan tingkat toleransi tertentu, membuatnya efektif untuk perhitungan numerik.

DAFTAR PUSTAKA

Vaikundam, G. pemrograman Teknik.

Sunandar, E. (2019). Penyelesaian Sistem Persamaan Non-Linier Dengan Metode Bisection & Metode Regula Falsi Menggunakan Bahasa Program Java.

Ilmayasinta, N. Bukti Ajar Mata Kuliah Metode Numerik.

Zakaria, L., & Muharramah, U. PENGANTAR METODE NUMERIK (Solusi Masalah dengan MATHEMATICA®).

Utomo, R. B. (2016). METODE NUMERIK BISEKSI.