بسم الله الرحمن الرحيم

الاسم: محمد نايف أولاد محمد الرقم الجامعي: 211026 الشعبة: 2

Numerical Analysis Spring Project 2023

Given $f(x) = \frac{x^3 + \sqrt[3]{x} - 1}{2 - x}$ as x ∈ [0 , 1], I will compute the solution of the function $f(x) = 0$ to within a tolerance 10^{-3} using C++.

1) Bisection Method:

Applying Bolzano theorem:

1. $f(x)$ is continuous in the given interval.

2. $f(0) < 0$ and $f(1) > 0$. So $f(0).f(1) < 0$, and there exist a point c ∈ (0 , 1) such that $f(c) = 0$ where $a = 0$ and $b = 1$

The code of the algorithm:

```
#include <iostream>
#include <cmath>
using namespace std;

//This is the given function f(x)
double f(double x) {
	return (pow(x, 3) + cbrt(x) - 1) / (2 - x);
}

int main() {
	// (a) is the first guess, (b) is the second guess,
	// (c) is the bisector, (E) is the tolerance
	double a = 0, b = 1, c, E = 0.001, iterations = 0;

	while (f(a) * f(b) < 0) {
		iterations++;

		c = (a + b) / 2;

		if (fabs(f(c)) < E) // |f(c)| < E ?
			break;

		if (f(a) * f(c) < 0)
			b = c;

		else a = c;
	}

	cout << "Number of iterations = " << iterations << endl;
	cout << "The solution c = " << c << endl;
}
```

The output:

Number of iterations = 9

The solution c = 0.560547

The values of c :

$c_1 = 0.5$, $c_2 = 0.75$, $c_3 = 0.625$, $c_4 = 0.5625$, $c_5 = 0.53125$, $c_6 = 0.546875$, $c_7 = 0.554688$,

$c_8 = 0.558594$, $c_9 = 0.560547$

2) Fixed-Point Method:

Transforming $f(x)$ to $g(x) = x$:

$$f(x) = \frac{x^3 + \sqrt[3]{x} - 1}{2 - x} = 0$$

$$x^3 + \sqrt[3]{x} - 1 = 0$$

$$x^3 = 1 - \sqrt[3]{x}$$

$$\boldsymbol{x = \sqrt[3]{1 - \sqrt[3]{x}} = g(x)}$$

1. $g(x)$ is continuous in [1 , 2]
2. $g(x) \in [1 , 2]$

$$P_0 = \frac{a+b}{2} = \frac{0+1}{2} = 0.5$$

The code of the algorithm:

```
#include <iostream>
#include <cmath>
using namespace std;

//This is the given function f(x)
double f(double x) {
	return (pow(x, 3) + cbrt(x) - 1) / (2 - x);
}

double g(double x) {
	return cbrt(1 - cbrt(x));
}

int main() {
	// (E) is the tolerance, P is the current answer (Pn)
	double E = 0.001, P = 0.5, iterations = 0;

	while (fabs(P - g(P)) > E) { // |P - g(P)| > E ?
		iterations++;

		P = g(P);
	}

	cout << "Number of iterations = " << iterations << endl;
	cout << "The solution P = " << P << endl;
}
```

The output:

Number of iterations = 7

The solution P = 0.560711

The values of P:

$P_0 = 0.5$, $P_1 = 0.59088$, $P_2 = 0.543857$, $P_3 = 0.568506$, $P_4 = 0.555688$, $P_5 = 0.562379$, $P_6 = 0.558894$, $P_7 = 0.560711$

3) Newton's Method:

Finding the derivative of $f(x)$:

$$f'(x) = \frac{\left(3x^2 + \frac{1}{3}x^{-2/3}\right)(2-x) + x^3 + \sqrt[3]{x} - 1}{(2-x)^2}$$

$$P_0 = \frac{a+b}{2} = \frac{0+1}{2} = 0.5$$

The code of the algorithm:

```
#include <iostream>
#include <cmath>
using namespace std;

//This is the given function f(x)
double f(double x) {
	return (pow(x, 3) + cbrt(x) - 1) / (2 - x);
}

//This is the derivative of the funciton f(x)
double fPrime(double x) {
	return ((3 * pow(x, 2) + 1.0 / 3.0 * pow(x, -2.0 / 3.0)) * (2 - x)
		+ pow(x, 3) + cbrt(x) - 1) / pow((2 - x), 2);
}

int main() {
	// (E) is the tolerance, P is the current answer (Pn)
	// previousP is the previous answer (Pn-1)
	double E = 0.001, P = 0.5, previousP, iterations = 0;

	while (fabs(f(P)) > E) { // |f(P)| > E ?
		iterations++;
		previousP = P;

		P = previousP - f(previousP) / fPrime(previousP);
	}

	cout << "Number of iterations = " << iterations << endl;
	cout << "The solution P = " << P << endl;
}
```

The output:

Number of iterations = 3

The solution P = 0.560154

The values of P:

$P_0 = 0.5$, $P_1 = 0.56637$, $P_2 = 0.560154$

4) Secant Method:

We get the first two values of P which are $P_0 = 0.5$ and $P_1 = 0.75$ from the previous solution of Bisection Method.

The code of the algorithm:

```
#include <iostream>
#include <cmath>
using namespace std;

//This is the given function f(x)
double f(double x) {
	return (pow(x, 3) + cbrt(x) - 1) / (2 - x);
}

int main() {
	// (E) is the tolerance, P is the current answer (Pn)
	// P0 and P1 are the first two values of P
	double E = 0.001, P, P0 = 0.5, P1 = 0.75, iterations = 0;

	do {
		iterations++;

		// P0 is considered as Pn-2 and P1 as Pn-1
		P = P1 - (f(P1) * (P1 - P0)) / (f(P1) - f(P0));

		// Update the values of P0 and P1
		P0 = P1;
		P1 = P;

	} while (fabs(f(P)) > E); // |f(P)| > E ?

	cout << "Number of iterations = " << iterations << endl;
	cout << "The solution P = " << P << endl;
}
```

The output:

Number of iterations = 3

The solution P = 0.560089

The values of P:

$P_1 = 0.542537$, $P_2 = 0.55504$, $P_3 = 0.560238$

Summary:

Newton's Method and Secant Method have found the answer using the least number of iterations (3) compared to Bisection Method (9) and Fixed-Point Method (7)