Assignment 2

Numerical Computing

Rudolf C. Kischer

260956107

Floating point in C, Overflow and Underflow, numerical cancellation

---

Q2. For any $x_0 >= 1$, the sequence defined recursively by

$$x_{n+1} = 2^{n+1}(\sqrt{1 + 2^{-n}x_n} - 1), \qquad (n \geq 0)$$

converges to $ln(x_0 + 1)$

Question: a)Let $x_0 = 1$.Use the formula to compute $x_n - \ln(x_0+1)$ n=1,2,. . . ,60 in double precision. Explain your results

Answer:

Recursive Fuction Definition

```
double x0 = 1.0;


std::unordered_map<int, double> memo;
double f(int n) {
  if (memo.find(n) != memo.end()) {
    return memo[n];
  }

  if (n == 0) {
    memo[0] = x0;
    return x0;
  }
  float under_sqrt = 1.0 + pow(2.0, -(n-1)) * f(n-1);
  printf("under_sqrt = %.30f\n", under_sqrt);

  memo[n] = pow(2.0, n) * (sqrt(under_sqrt) -1.0);
  return memo[n];
}
```

Difference from convergent Value

```
double h(int n) {
  return f(n) - log(x0 + 1.0);
}
```

Input:

```
int main() {

    int m = 60;

    // Question a)
    for (int i = 0; i <= m; i++) {
        printf("h(%d) = %.8e\n", i, h(i));
    }
}
```

Output:

```
f(0) = 1.000000000000000000000000000000e+00
f(1) = 8.284270763397216796875000000000e-01
f(2) = 7.568283081054687500000000000000e-01
...
f(58) = 0.000000000000000000000000000000e+00
f(59) = 0.000000000000000000000000000000e+00
f(60) = 0.000000000000000000000000000000e+00
```

```
h(0) = 3.06852819e-01
h(1) = 1.35279896e-01
h(2) = 6.36811275e-02
h(3) = 3.09147854e-02
h(4) = 1.52325649e-02
...
h(57) = -6.93147181e-01
h(58) = -6.93147181e-01
h(59) = -6.93147181e-01
h(60) = -6.93147181e-01
```

See the attached file `q2a.txt` for the full output.

Explanation: - We can see clearly that we did not get the result we wanted. We can deduce by our formula for `h(x)` that our value for `f(x)` wen to zero at some point during our computation. The computed result for `f(x)` also demonstrates that.

- This is because our the value under the square root becomes a number that is very close to 1, but not exactly 1. The difference becomes so small, that it cannot be properly represented and is rounded down. This cancled out with the negative 1, which is why we get 0.
- Although the value for one intermediate part of our computed result is not representable in our floating point system, the final result should still be representable. This is because the final result is the product of a large number (2^{n+1}) and a very small number.
- This is not the case however, because the intermediate result is rounded down to 0, this means that the final result is also 0.

Question:

b) Improve the formula to avoid the difficulty you encountered in 2(a). Again compute $x_n - \ln(x_0 + 1)$ for n=1,2,...,60 in double precision.

Answer:

Solution: As seen in class , a technique to avoid this is by multiplying by the conjugate, also known as rationalization. This is done by multiplying by the conjugate of the numerator and denominator.

$$x_{n+1} = 2^{n+1}(\sqrt{1 + 2^{-n}x_n} - 1) \cdot \frac{\sqrt{1 + 2^{-n}x_n} + 1}{\sqrt{1 + 2^{-n}x_n} + 1}$$

$$\implies x_{n+1} = \frac{2 \cdot x_n}{\sqrt{1 + 2^{-n}x_n} + 1}$$

We can see that this formula is much more stable, because we dont have intermediate values that cancel out.

Below is the code to support that change.

```
std::unordered_map<int, double> memo_fixed;
double f_fixed(int n) {
  if (memo_fixed.find(n) != memo_fixed.end()) {
    return memo_fixed[n];
  }

  if (n == 0) {
    memo_fixed[0] = x0;
    return x0;
  }

  float x_n = f_fixed(n-1);

  float under_sqrt = 1.0 + pow(2.0, -(n-1)) * x_n;

  memo_fixed[n] = (2.0 * x_n) / (sqrt(under_sqrt) + 1.0);
  return memo_fixed[n];
}

double h_fixed(int n) {
  return f_fixed(n) - log(x0 + 1.0);
}
```

Input:

```
int main() {

    int m = 60;
    for (int i = 0; i <= m; i++) {
        printf("f_fixed(%d) = %.12e\n", i, f_fixed(i));
    }
    for (int i = 0; i <= m; i++) {
        printf("h_fixed(%d) = %.12e\n", i, h_fixed(i));
    }
    return 0;
}
```

Output:

```
f_fixed(0) = 1.000000000000e+00
f_fixed(1) = 8.284271330514e-01
f_fixed(2) = 7.568284833700e-01
...
f_fixed(58) = 6.931475400925e-01
f_fixed(59) = 6.931475400925e-01
f_fixed(60) = 6.931475400925e-01

h_fixed(0) = 3.068528194401e-01
h_fixed(1) = 1.352799524915e-01
h_fixed(2) = 6.368130281005e-02
...
h_fixed(58) = 3.595325229755e-07
h_fixed(59) = 3.595325229755e-07
h_fixed(60) = 3.595325229755e-07
```

See the attached file `q2b.txt` for the full output.

Explanation: Now the error stabilizes, and does not increase as we continue to compute. This is because by rationalizing we have avoided the intermediate values that cancel out. The final result of `f(x)` is also very close to the expected value of `ln(2)`.