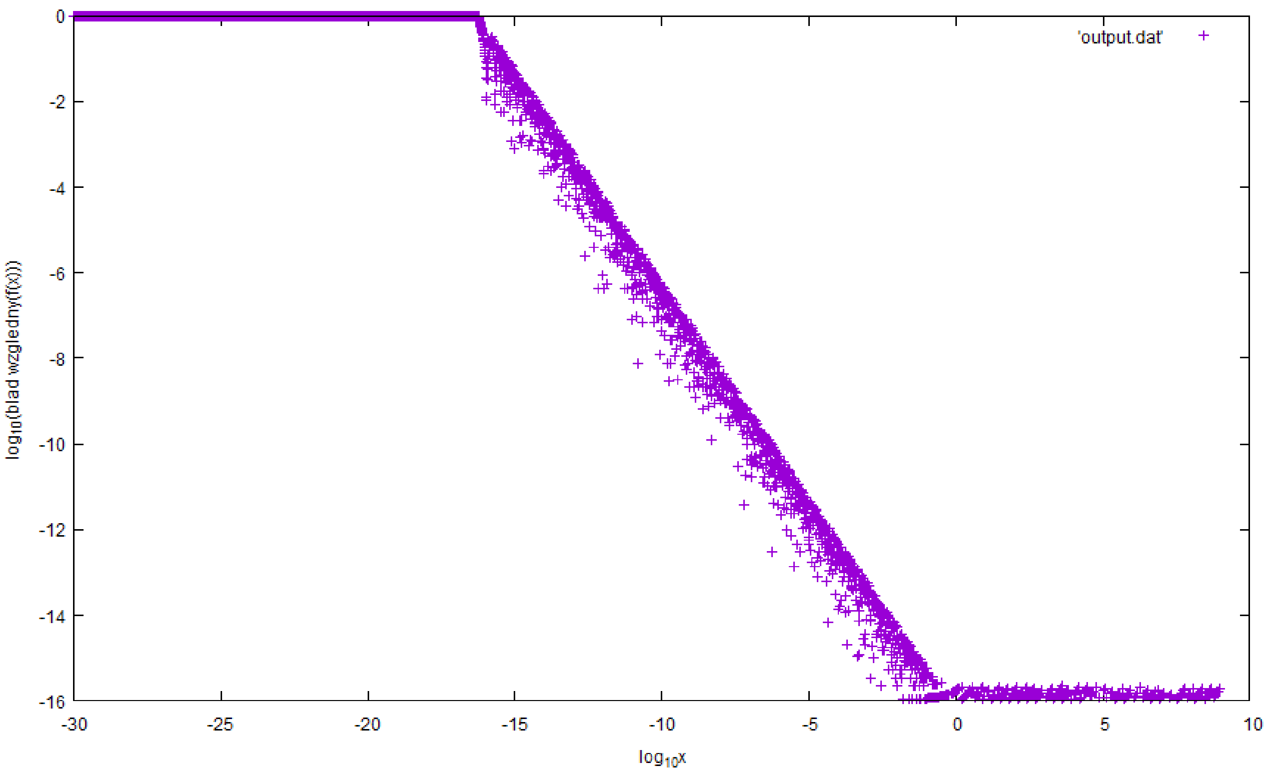
Laboratorium 2

Artem Buhera
GĆ01 135678

13.06.2021

Obliczając wartość funkcji $f(x)=(1-e^{-x})/x$ przy użyciu funkcji standardowej exp() i budując wykres błędu względnego między wartością obliczoną przez komputer a dokładną, możemy zaobserwować, że dla argumentów mniejszych od około $10^{-0.5}$ błąd jest tym większy, im mniejszy jest argument:

```
double f(double x) {
    return (1 - exp(-x)) / x;
}

int main() {
    ifstream plik;
    ofstream output;

    plik.open("dane.txt");
    output.open("output.dat");

    double log10x, x, exact, calculated;
    while(plik.good()) {
        plik >> log10x >> x >> exact;

        calculated = f(x);

        output << log10x << ' ' << log10(abs((exact - calculated) / exact)) << endl;
    }
}
```



Wykres logarytmiczny zależności błędu względnego miedzy wartością obliczoną a wartością dokładną funkcji f(x) od argumentu x

Obserwowane błędy wynikają m.in. z tego, że dla bardzo małych argumentów funkcja exp() daje niewystarczająco precyzyjne wyniki. Z tego powodu wyprowadzimy nowy wzór na f(x) używając rozwinięcia Taylora funkcji e^x wokół punktu $x_0 = 0$:

$$
\begin{aligned}
&(e^{-x})^{(0)} = e^{-x}\\
&(e^{-x})^{(1)} = (e^{-x})' = e^{-x}(-x)' = -e^{-x}\\
&(e^{-x})^{(2)} = (-e^{-x})' = e^{-x}\\
&\vdots\\
&(e^{-x})^{(n)} = (-1)^{n+1}e^{-x}
\end{aligned}
$$

$$(e^{-0})^{(n)} = (-1)^{n+1}e^{-0} = (-1)^{n+1}$$

$$
\begin{aligned}
g(x) = \sum_{n=0}^{\infty}\frac{g(x_0)^{(n)}}{n!}(x-x_0)^n \quad \Rightarrow \quad e^{-x} = \sum_{n=0}^{\infty}\frac{(e^{-0})^{(n)}}{n!}(x-0)^n = \sum_{n=0}^{\infty}(-1)^n\frac{x^n}{n!} = \\
= \frac{x^0}{0!} - \frac{x^1}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} ... = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots
\end{aligned}
$$

$$
\begin{aligned}
f(x) = \frac{1-e^{-x}}{x} = \frac{1-\left(1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - ...\right)}{x} = \frac{1- 1 + x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + ...}{x} \\
= \frac{x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + ...}{x} = 1 - \frac{x}{2!} + \frac{x^2}{3!} - \frac{x^3}{4!} + ... = \sum_{n=0}^{\infty}(-1)^n\frac{x^n}{(n+1)!}
\end{aligned}
$$

Dla dużego n obliczanie x^n oraz (n+1)! w każdej iteracji jest nieoptymalne, dlatego zaimplementujemy obliczanie f(x) poprzez dodawanie do wyniku części, którą w każdej iteracji będziemy domnażać przez x oraz dzielić przez n+1, efektywnie obliczając kolejne potęgi x oraz silnię bez zbędnych operacji.

Ponieważ dysponujemy wartościami dokładnymi, możemy liczyć dopóki różnica między wartością do tej pory obliczoną a dokładną nie jest równa lub mniejsza od epsilonu maszynowego, wartość którego możemy wziąć z biblioteki <cfloat>. W innym przypadku musielibyśmy najpierw wyznaczyć minimalny n, dla którego błąd względny wartości obliczonej a dokładnej byłby na poziomie epsilonu maszynowego dla argumentów z całego zakresu, na którym będziemy używać dany algorytm.

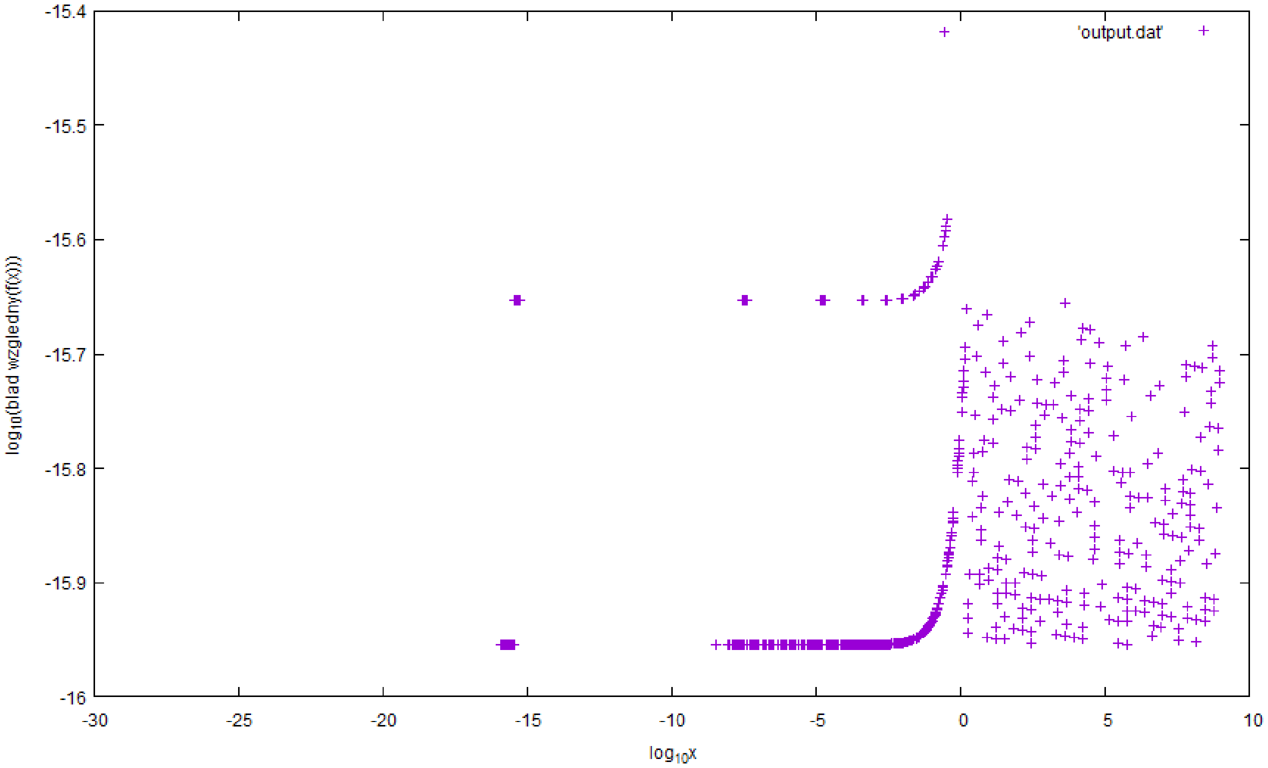
Musimy jednak przerywać obliczanie w momencie, gdy ta część, obliczana w każdej iteracji, nie staje się na tyle mała, że jest reprezentowana jako zero, przez co podalsze obliczanie nigdy się nie kończy.

```
double recalculated(double x, double exact) {
    double result = 0, iter_part = 1;
    int sign = 1;

    for(int n = 1; abs(exact - result) > DBL_EPSILON; n++) {
        result += sign * iter_part;

        iter_part *= x / (n + 1);
        if (iter_part == 0)
            break;

        sign *= -1;
    }

    return result;
}
```

Teraz możemy użyć funkcji `recalculated` dla zakresu, w którym poprzednia funkcja miała za duży błąd względny, czyli około [10^{-30}, $10^{-0.5}$]:

```
calculated = log10x < -0.5 ? recalculated(x, exact) : f(x);
```

Poniższy wykres pokazuje, że błąd względny nowej funkcji jest bardzo zbliżony do poziomu epsilonu maszynowego.



Wykres logarytmiczny zależności błędu względnego miedzy wartością obliczoną nową metodą a wartością dokładną funkcji f(x) od argumentu x