Homework 3

ZHONG Yifan

April 2, 2018

1 Problem 1

In this problem, we are required to determine the roots of the quadratic equation

$$x^2 - bx + 1 = 0$$

to show the inaccuracy of floating point number arithmetic. The second root

$$x_2 = \frac{b - r}{2},\ r = \sqrt{b^2 - 4} \tag{1}$$

will be interpreted differently in floating point number system when using the rationalized form

$$x_2 = \frac{2}{b + r},\ r = \sqrt{b^2 - 4} \tag{2}$$

For demonstrative purpose, single precision is used in C++ to compute the answer because the limit of floating point number will be better illustrated with lower ε_m. The root computed in double precision using Equation 2 is regarded to be the exact root x_2 for the equation. The result for different b is listed in the table below. The last two columns are the relative error accordingly.

b	$x_{2,n}$	$x_{2,r}$	Exact x_2	Error 1	Error 2
100	0.010001	0.010001	0.010001	-1.83104e-06%	-1.83104e-06%
1000	0.001	0.001	0.001	-2.1182e-06%	-2.1182e-06%
10000	0	0.0001	0.0001	-100%	-3.52621e-06%
100000	0	1e-05	1e-05	-100%	-2.53621e-06%

Table 1: Roots for different coefficient b

We can clearly learn that the Equation (2) yields more accurate x_2 than Equation (1) when b is very large. This is a vivid example for cancellation error. When we compute

$$a = b - c$$

in floating number arithmetic, the final error of a is

$$\varepsilon_a = \frac{b}{a}\left(|\varepsilon_b| + |\varepsilon_c|\right)$$

In our case, it is possible that $b = 1000$ and $a = 0.00002$. The error of a, ε_a, could be 5×10^9 greater than that of b or c. Therefore, the precision is drastically less significant using Equation (1).

However, Equation (2) preempted us from subtraction of two approximately equivalent floating number and the corresponding result is tolerable even b is quite large.

Note: Run the *run.sh* in folder *Problem 1* and you will get the result.

2 Problem 2

Floating point number arithmetic contains error intrinsically and even worse, the error will propagate during the computing process. That is when you increase the steps of computing, the result will be less accurate. To enlucid this property, we choose three different forms of an expression, which gives the same result algebraically. The only distinguish in the number of computing. However, this can lead to intolerable error in floating number arithmetic.

The three different forms are

1. The contracted form

$$y = (x-1)^9 \tag{3}$$

   It requires 1 addition and 9 multiplications.

2. Expanded form

$$y = x^9 - 9x^8 + 36x^7 - 84x^6 + 126x^5 - 126x^4 + 84x^3 - 36x^2 + 9x - 1 \tag{4}$$

   This requires 9 additions and 45 multiplications to attain a value.

3. Horner's method

$$y = x(x(x(x(x(x(x((x-9)x+36)-84)+126)-126)+84)-36)+9)-1 \tag{5}$$

   Horner's method requires 9 additions and 8 multiplications to give a result.

We can predict that the most accurate algorithm is Equation (3) followed by Horner's algorithm in Equation (5). The worse method is to expand the expression as in Equation (4)

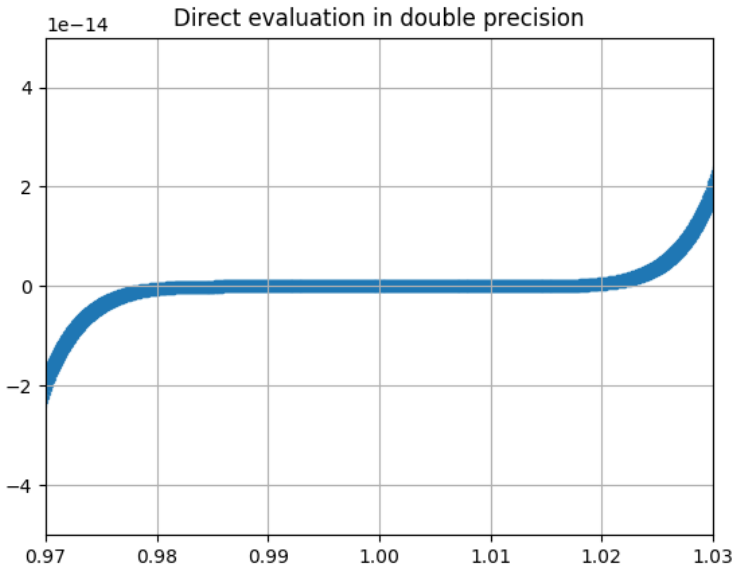
This following part of this section is dedicated to detailed result and discussion on some enthralling phenomena. The evaluation is performed on the interval $[0.7, 1.3]$

2.1 Double precision

We first divided the the interval $[0.7, 1.3]$ into 10^4 segments and computed the expression in different forms with single and double precision in C++.

This three figures Figure 1-3 are the result in double precision

Figure 1: Contracted form in double precision, 10^4 segments



As our anticipation, the contracted form gives the most accurate result for the least number of manipulation. The result from Horner' method is less nice but still relatively acceptable. However, when we expand the expression to evaluate, the numerical result divert notable from the exact value because because of the excessive times of manipulation on floating number.

2.2 Single precision

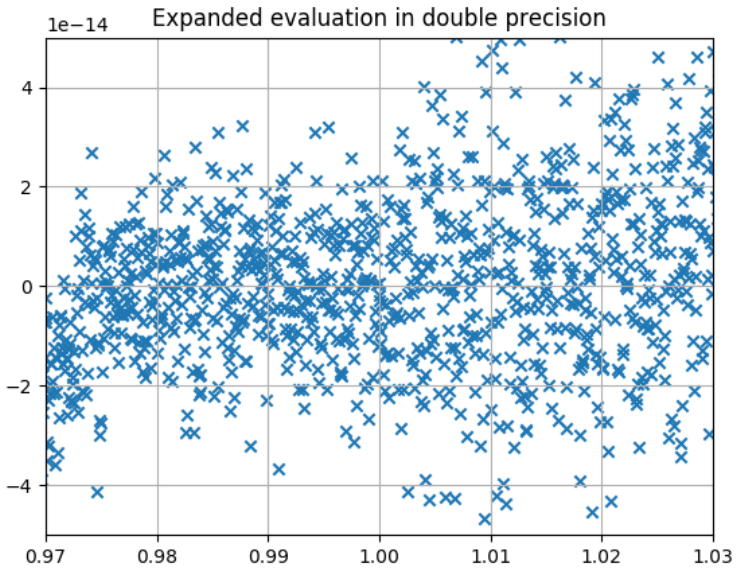
Interesting patterns emerge when we use single precision to repeat the process. Divide the interval into 10^4 fragments, we get the Figure 4-6 below.

Figure 1 and Figure 4 are quite similar to each other. They both demonstrate the accuracy of evaluating a expression without unnecessary computation. Cases are disparate for Figure 2 and 5.

In Figure 2, the points smear about the accurate value. Staggering, the points in Figure 2 forms a pattern. It seems like that the real line of $y = (x-1)^9$ is cut and shifted vertically to generate the figure. In other words, certain results share lost accuracy by the same degree.

In Figure 6, horizontally parallel lines can be observed. This also indicates that the loss of nicety is periodic with finite modes.

Figure 2: Expanded form in double precision, 10^4 segments



2.3 A closer look

In the previous part, we have already seen some mesmeric patterns in the figures corresponding to single precision while those of double precision looks nothing new. The question which is natural to raise is that is this inherent for floating number arithmetic or simply a coincidence on single precision. To validate the conjectures, we may include more sampling points in double precision evaluation to see if similar patter can show up.

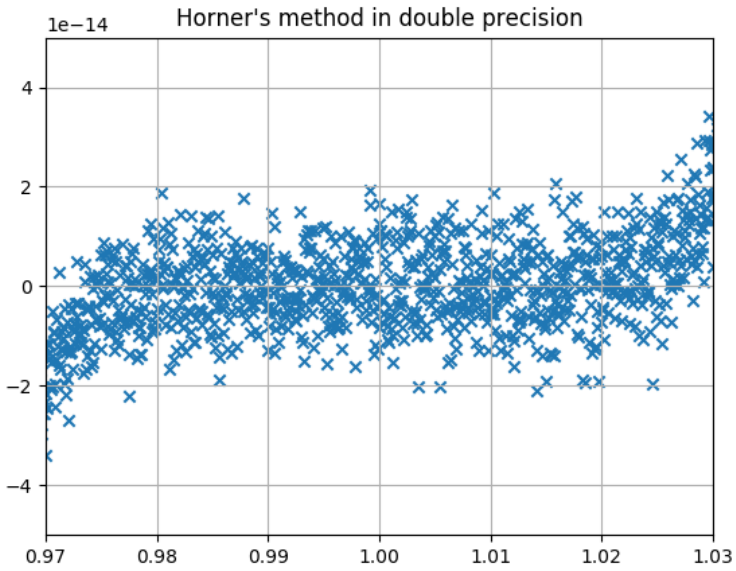
This time we use 10^6 point to divide the same interval. The result of single precision is unchanged when more points is added. You may compare Figure 4-6 with Figure 7-9 to attain the conclusion.

Figures for double precision may seem the same even more points are inserted as is shown in Figure 10-12. The points merely spread in a wider region on the xy plane.

However, when we zoom in to the 10^{-15} scale in y-axis, we finally encounter the pattern met in single precision with 10^4 points. This is shown in Figure 14 and 15. The pattern will **emerge again** as long as the number line is divided into segments small enough. We can generalize that it is an **intrinsic** result of floating number arithmetic because we found the pattern in two different scheme of precision.

The evaluation in double precision with 10^6 points provided us with more information about the error in floating number arithmetic. In Figure 11, we can

Figure 3: Horner's method in double precision, 10^4 segments



see that the points spreads wider in the right half plane divided by $x = 1$. The accurate result changes sign from negative to positive when crossing the vertical line. This may indicate that floating number arithmetic can be heterogeneous for positive number and positive numbers.

From Figure 9, 14 and 15 we can also assert that the distribution of error is uneven on the two sides of $x = 1$

2.4 Conclusion and Discussion

We can draw some conclusions from the text above and they will be briefly justified.

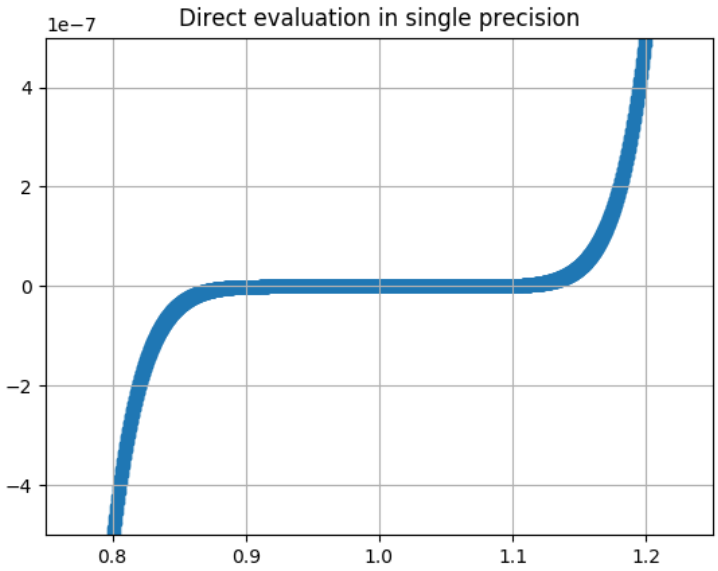
1. The error in floating number arithmetic will propagate

   We have confirmed that the best algorithm should involve least manipulation on floating number. In our case, evaluating $y = (x - 1)^9$ directly wins over other algorithms.

2. Double precision has higher accuracy than single precision

   If our evaluation only differ in the scheme of precision, we ca learn from the plot that dots in double precision is more concentrated. This can support the statement.

Figure 4: Contracted form in single precision, 10^4 segments



3. The error has certain discrete modes

   We first observed the phenomena, some definite patterns, in Figure 5 and 6. Though we suspected this is a unique consequence due to single precision, the same pattern in double precision as we increase sampling points, which indicates the phenomena is mutual among floating number arithmetic.

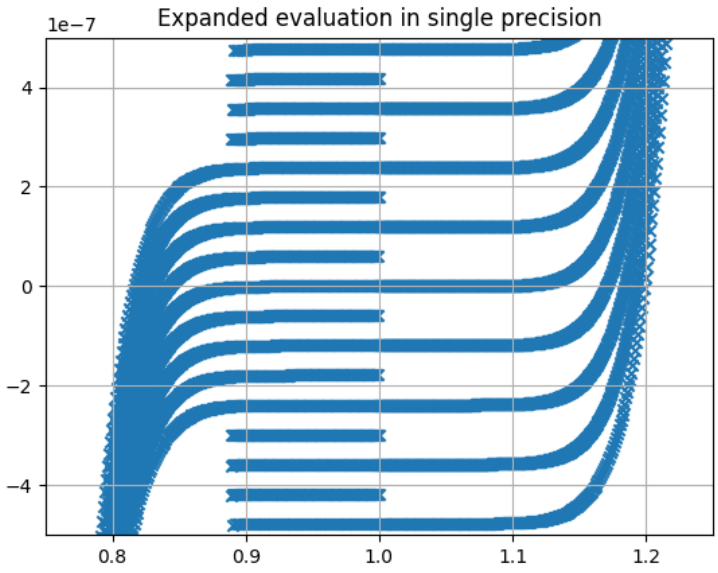
   The best explanation is that not all decimal fractions can be impeccably represented in binary number system. Only a specific set of them can be converted without any loss of precision. The numbers on the number line is nonuniform and periodic. This triggers the the patterns in the figures.

   As for the absence of pattern in evaluation in double precision with 10^4 points, higher nicety of double precision is the main reason. Double precision has smaller ε_m the discretization is achieve with finer details. Equivalently, the period of discretization error tends to be longer. 10^4 points may not be enough for notable pattern but as we increase the points to 10^6, the pattern emerge immediately.

   Thus, the pattern reflects mutual discretization error in floating number arithmetic.

Note: *run.sh* will generate the data and make some plots while *visual.py* only plots using currently generated data. Please feel free to reinterpret the

Figure 5: Expanded form in single precision, 10^4 segments



data in this problem.

Figure 6: Horner's method in single precision, 10^4 segments

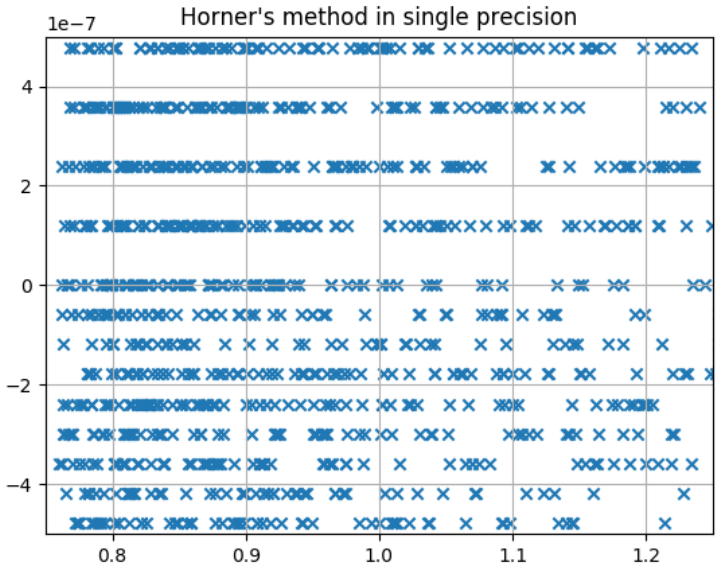


Figure 7: Contracted form in single precision, 10^6 segments

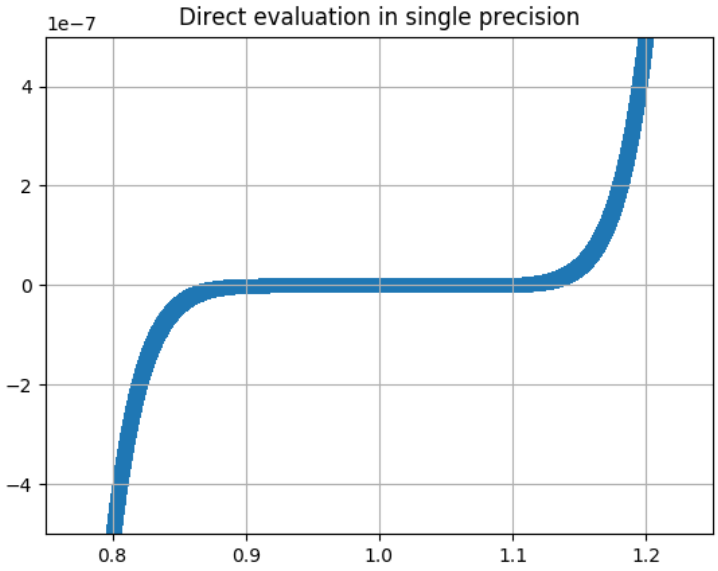


Figure 8: Expanded form in single precision, 10^6 segments

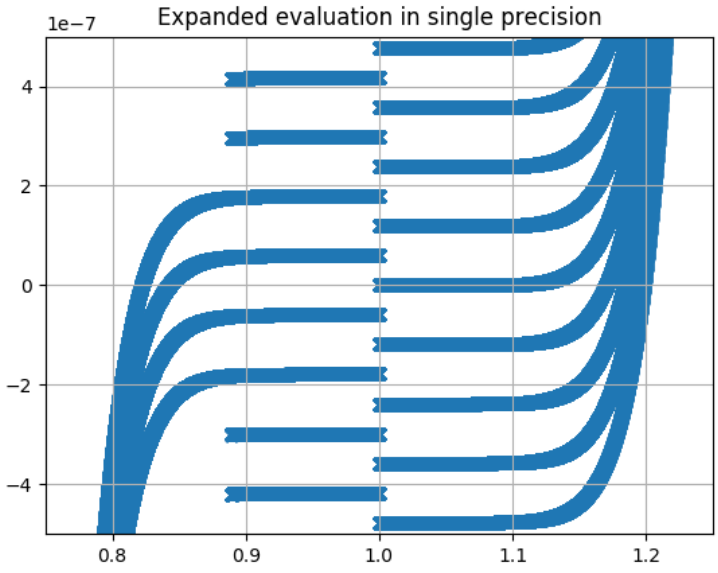


Figure 9: Horner's method in single precision, 10^6 segments

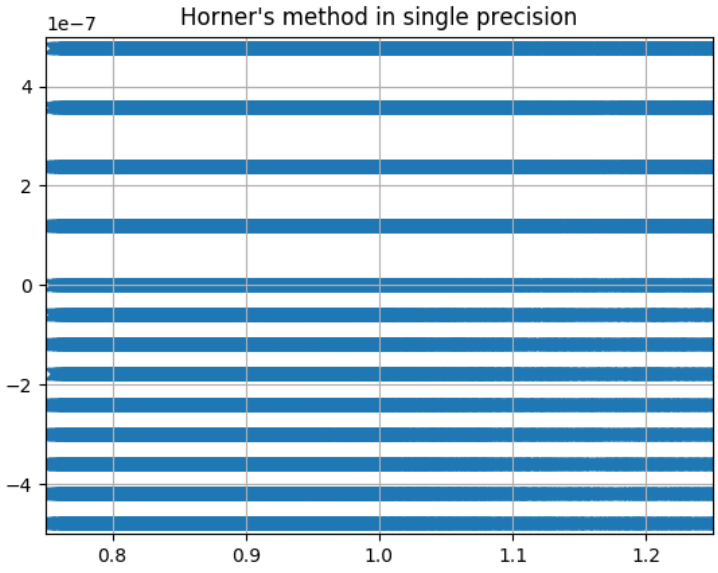


Figure 10: Contracted form in single precision, 10^6 segments

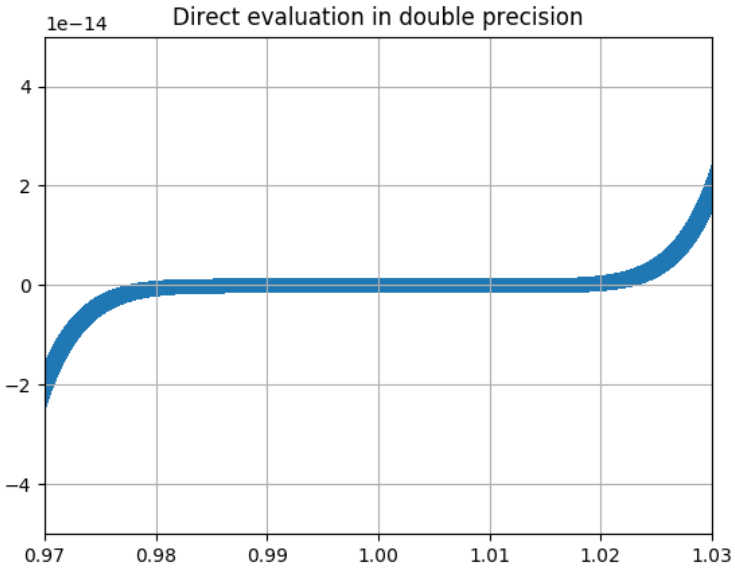


Figure 11: Expanded form in single precision, 10^6 segments

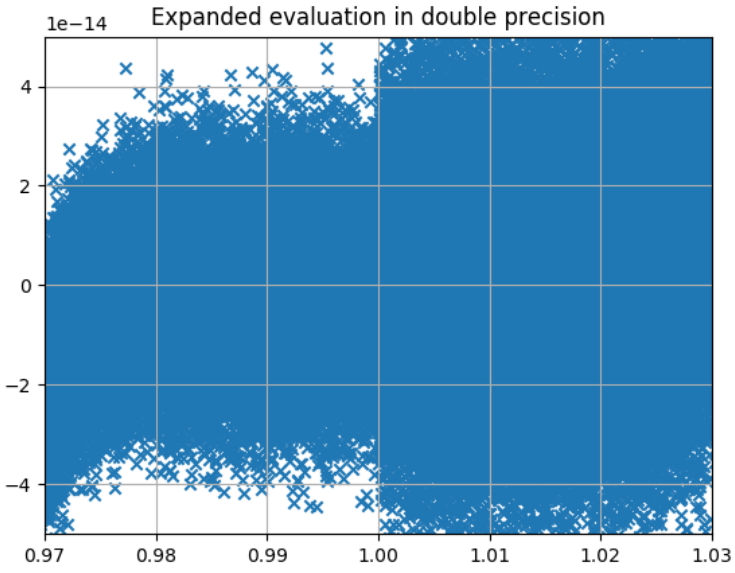


Figure 12: Horner's method in single precision, 10^6 segments

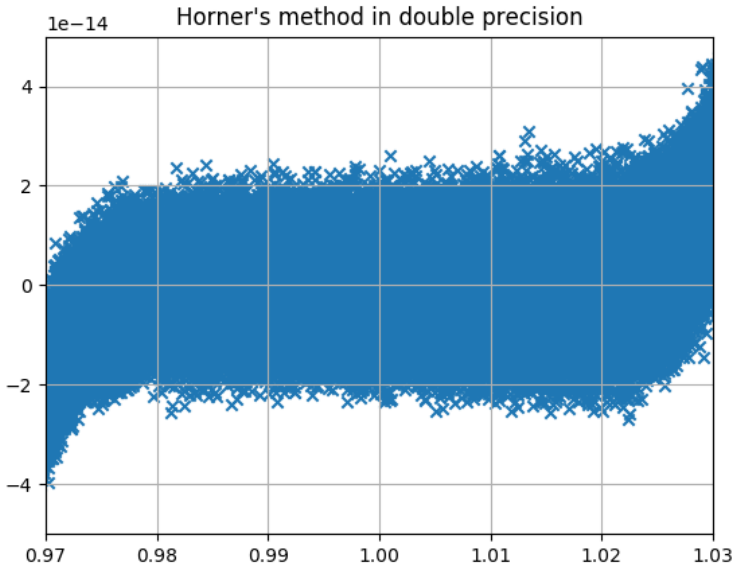


Figure 13: Contracted form in single precision, 10^6 segments

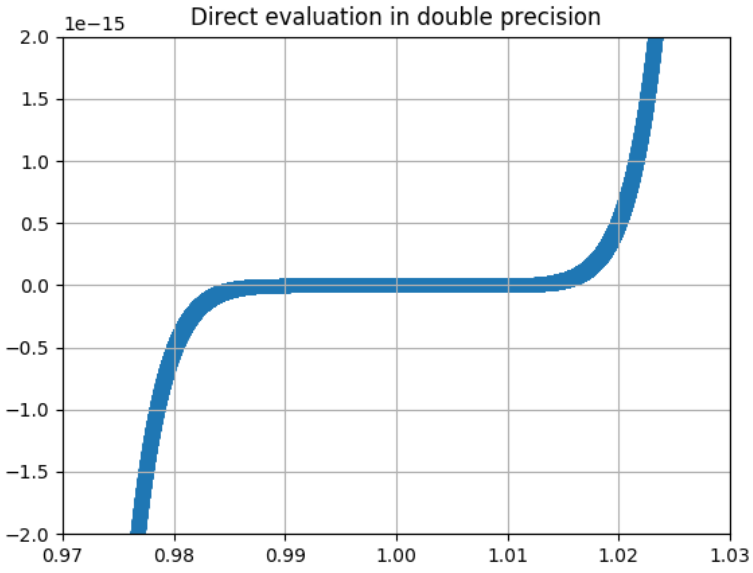


Figure 14: Expanded form in single precision, 10^6 segments

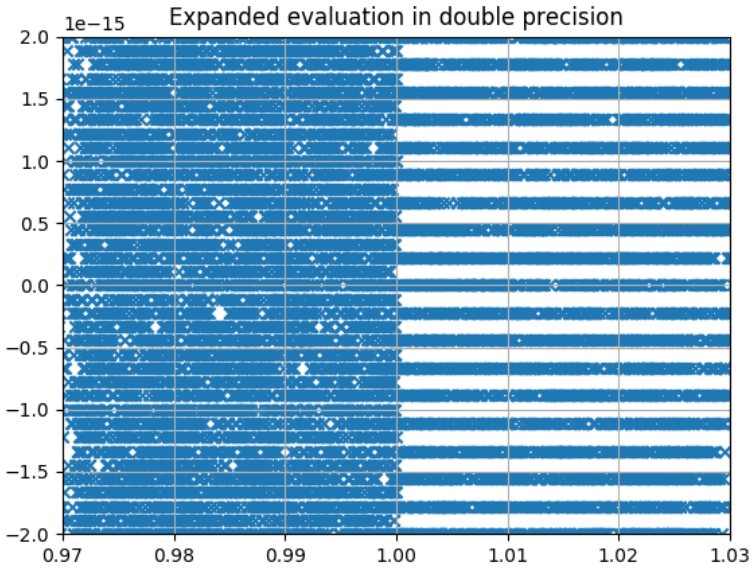


Figure 15: Horner's method in single precision, 10^6 segments