EX.0.4.3, Sauer

Explain how to most accurately compute the two roots of the equation $x^2 + bx - 10^{-12} = 0$, where b is a number greater than 100.

EX.0.4.3, Sauer, solution, Langou

- Only turning the Python code is not a good answer.
- The copy-paste from this PDF to python code does not work great. It is better to copy-paste from colab.
- The Colab Jupyter Notebook is available at: `https://colab.research.google.com/drive/19c-eoifrUWtfhXvYtjqMIsPeFEXW32CP`.
- The Python code and its ouput is at the end of this document.

We can use the standard formulae for the roots for a polynomial of degree 2:

$$x_1 = \frac{1}{2}(-b + \sqrt{b^2 + 4 \cdot 10^{-12}})$$
$$x_2 = \frac{1}{2}(-b - \sqrt{b^2 + 4 \cdot 10^{-12}}).$$

However, for large b, say b larger than 100, we see that the formula for the first root, x_1, we lead to something like this:
When computed $\Delta = b^2 + 4 \cdot 10^{-12}$ will be approximately b^2, then $\sqrt{\Delta}$ will be approximately b, and so x_1 will be be approximately $b - b$ so 0.
The issue is that there is no room in the 52-bit mantissa of Δ to store $4 \cdot 10^{-12}$. b is about 10^2, so b^2 is about 10^4, so larger than 2^{13}. Then $4 \cdot 10^{-12}$ is smaller than 2^{-37}. Because the gap between 2^{13} and 2^{-37} is 2^{50}, this means that there are at least 50 bits in the 52-bit representation of $\Delta = b^2 + 4 \cdot 10^{-12}$ that are zeros, and then you can start representing $4 \cdot 10^{-12}$, so you have a very poor accuracy. Even worse, for $b = 10^3$, then $4 \cdot 10^{-12}$ is washed away and we actually have

$$fl(b^2 + 4 \cdot 10^{-12}) = b.$$

A better formula when b is larger than 100, (and a is 1, and c is -10^{-12}), for the first root, x_1 is

$$x_1 = -\frac{2 \cdot 10^{-12}}{b + \sqrt{b^2 + 4 \cdot 10^{-12}}}$$

Here is a python code to show this. Here we set $b = 10^3$. We first compute a "trusted" answer using `numpy.root`, and we find that x_1 should be around 10^{-15}. Then we use the "bad" formula, and we find that $x_1 = 0$. Given that x_1 should be around 10^{-15}, the relative error by returning $x_1 = 0$ infinitely large. Then we use the "good" formula, and we find $x_1 = 10^{-15}$, which is much more in accordance with `numpy.root`.

```
from math import sqrt
import numpy as np
np.set_printoptions(precision=16,suppress=False)
```

```
a = 1.
b = 1e3
c = - 1e-12
```

```
r = np.roots( np.array( [ a, b, c ]) )
print( r[0], r[1] )
```

```
-1000.0 1e-15
```

```
x0 = ( - b - sqrt( b**2 - 4 * a * c ) ) / ( 2 * a )
x1 = ( - b + sqrt( b**2 - 4 * a * c ) ) / ( 2 * a )
print( x0, x1 )
```

```
-1000.0 0.0
```

```
x0 = ( - b - sqrt( b**2 - 4 * a * c ) ) / ( 2 * a )
x1 = - ( 2 * c ) / ( b + sqrt( b**2 + 4 * a * c ) )
print( x0, x1 )
```

```
-1000.0 1e-15
```