Gabriel Rios
AOS 575
September 8, 2022

Problem Set 0. Quadratic Equation

1. **Explain why:**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

is a very inaccurate expression for using floating point numbers to find one of the roots of:

$$x^2 - (1 + 10^{-20})x + 10^{-20} = 0 \tag{2}$$

and demonstrate in code an approach that accurately yields both roots.

Equation 1 is inaccurate for using floating point numbers because of the discrepancy in orders of magnitude between the coefficients, specifically a and c. The term $b^2 - 4ac$ is a source of this inaccuracy, because $-b \pm \sqrt{b^2 - 4ac} \rightarrow -b \pm \sqrt{b^2} \rightarrow 0$ for one of the cases when $a >> c$ or $c >> a$, due to round-off error.

To try and address this, normalizing the coefficient values (a, b, c) is attempted in the code attached below. The point of normalizing the coefficient values is to reduce the disparity in magnitudes of the coefficient values to try and avoid the round-off error. The results are adequate for positive coefficient values (see the test cases in the code), although negative values produce real and imaginary components. It is worth noting that the real components are also adequate, but this method needs improvements.

Appendix

notebook

September 8, 2022

[324]:
```
import numpy as np
```

Define the quadratic function with customization to handle floating points.

[496]:
```
def quadratic(a, b, c, norm = 1/2):
    # 'norm' is the exponent to normalize by (for example, '1/2' = square root)

    # Normalize all coefficients
    a = a**(norm)
    b = -(abs(b)**(norm))
    c = c**(norm)
    print('Normalized a = {0}, b = {1}, c = {2}'.format(a, b, c))
    # Use the quadratic on the normalized coefficients and reverse the␣
 ↪normalization to get roots
    return [((-b + np.sqrt(b**2 - 4*a*c))/(2*a))**(1/norm),
            ((-b - np.sqrt(b**2 - 4*a*c))/(2*a))**(1/norm)]
```

Test case 1: nominal case (a = 1, b = -(1 + 1e-20), c = 1e-20)

[487]:
```
a = 1
c = 1e-20
b = -(a + c)
quadratic(a, b, c)
```

```
Normalized a = 1.0, b = -1.0, c = 1e-10
```

[487]:
```
[0.9999999998, 1.0000001654807488e-20]
```

Test case 2: random coefficients with similar scales to nominal

[488]:
```
a = 134
c = 4e-19
b = -(a + c)
quadratic(a, b, c)
```

```
Normalized a = 11.575836902790225, b = -11.575836902790225, c =
6.324555320336759e-10
```

[488]:
```
[0.9999999998907283, 2.9850777742600163e-21]
```

Test case 3: very small **c** value and different normalization exponent

[489]:
```
a = 4
c = 3e-36
b = -(a + c)
quadratic(a, b, c, norm = 1/4)
```

```
Normalized a = 1.4142135623730951, b = -1.4142135623730951, c =
1.3160740129524924e-09
```

[489]:
```
[0.9999999962775803, 7.500000332416309e-37]
```

Test case 4: negative coefficients Note: negative coefficients yield imaginary components, since normalization is performed by getting a square, cube, or other root. Another normalization process should be used in the future.

[501]:
```
a = -1
c = 1e-20
b = -(a + c)
quadratic(a, b, c, norm=1/2)
```

```
Normalized a = (6.123233995736766e-17+1j), b = -1.0, c = 1e-10
```

[501]:
```
[(-1+1.999998775353201e-10j), (1.0000000000000001e-20+1.2246467991473533e-36j)]
```