Recitation 11

Recitation Instructor: Shivam Verma
Email: shivamverma@nyu.edu
Ph: 718-362-7836
Office hours: WWH 605 (2.50 - 4.50 pm, Tuesdays)

Brief Overview

- Practice problems
- Floating point system
- Two's complement

Binary system

- Rational numbers have terminating or non-terminating repeating expansion in any number system (decimal or binary). Irrationals are non-repeating non-terminating.
- $(71)_{10} = 7 \times 10^1 + 1 \times 10^0 = 1 \times 64 + 0 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = (1000111)_2$
- Negative numbers represented using the 2's complement system, which means that a negative number $-y$ is represented as $2^{32} - y$ in single precision. This is because $y + (2^{32} - y) = 2^{32}$, which is just 0 since the overflow bit is ignored.
- You should know how to convert binary to decimal, decimal to binary as well as adding two binary numbers! See [18-21] for nice references. Wikipedia also has lots of great articles on the topic, such as [22, 23, 24].

Homework 5 tips

To illustrate how to do these problems, i'll do ex. 3.2, 3.3 on blackboard.

- 3.1, 3.4, 3.5: Read [22-24].
- 3.8
- 3.10
- 4.1
- 4.2
- Q8
- Q9

Floating Point System

- Due to finite precision in floating point number representation, there are gaps between consecutive numbers.
- Size of these gaps depends on the size of the number and on the precision (e.g., double or single precision).
- MATLAB has the function eps(), which returns, for a number, the distance to the next floating point number in the same precision.
- A double precision (64-bit) float is represented by 64 bits, with 52 bits in significand, 11 bits in exponent and 1 bit for sign, and implicit integer bit of value 1. 53 bits in significand is equivalent to roughly 16 decimal digits, since $log_{10}(2^{53}) \approx 15.955.$
- A single precision (32-bit) float is represented by 32 bits, with 23 bits in significand, 8 bits in exponent and 1 bit for sign, and implicit integer bit of value 1.
- 64 bit: Emin = -1022, Emax = 1023, 32 bit: Emin = -126, Emax = 127. Comes from two-complement (see 6,7), $-2^{N-1} to (2^{N-1}-1)$.

Examples:

- eps(1)
  - $eps(1) = 2.2204e-16$
  - 1 is represented as $1.00..000 \times 2^0$ where there are 52 zeros in significand. The next largest number is thus $1+2^{-52}$, with $2^{-52} \approx 2.2204e-16$.
- eps(single(1))
  - $eps(single(1)) = 1.1921e-07$
  - For single precision format, there are only 23 bits in the significand (after the implicit 1). This means the next largest representable number is $1+2^{-23}$ with $2^{-23} \approx 1.1921e-07$.
- eps(2^(40))
  - $eps(2^{40}) = 2.4414e-04$
  - In the 64-bit representation the number 2^{40} is given as $1.00..0 \cdot \times 2^{40}$ so the next number is $2^{40} + 2^{-12}$ with $2^{-12} \approx 2.4414e-04$.
- eps(single(2^(40)))
  - $eps(single(2^{40})) = 131072$
  - In the 32-bit representation there are only 23 zeros so the next number is $2^{40}+2^{17}$ with $2^{17} = 131072$ which is what single(eps(2^40)) gives.

Try the following in MATLAB:

```
>> a = 0.8;
>> b = 0.7;
>> a - b == 0.1
ans = 0
>> a - b - 0.1
ans = 8.3267e-17

>> a = 0.8;
>> b = 0.4;
>> a - b == 0.4
ans = 1

>> a - b - 0.4
ans = 0
```

Can you explain this behavior?

We'll use the concept of a **dyadic rational,** which is a number of the form $a/2^b$, i.e. its denominator is a power of 2. Such numbers have a finite binary expansion. See [5].

Since neither 0.8 nor 0.7 is a dyadic rational, they cannot be expressed exactly using the 64-bit floating point format. More precisely, we have

$$0.8 \approx a = \frac{3602879701896397}{4503599627370496},\ 0.7 \approx b = \frac{3152519739159347}{4503599627370496},$$

where a, b are the 64-bit floating point representations. The result of a − b is the closest 64-bit representation of the difference of these 2 approximating dyadic rationals. The exact difference of these fractions is
$a - b = \frac{225179981368525}{2251799813685248}$, which is exactly representable as a 64-bit value.

As a result you get:

```
>> a = 0.8;
>> b = 0.7;
>>  a-b == 225179981368525/2251799813685248
ans = 1
```

The value 0.1 is also not dyadic and is approximated as:

$0.1 \approx c = \frac{3602879701896397}{36028797018963968}$.

We note that $c + (2^{-51} - 2^{-52} - 2^{-53} - 2^{-55}) = a - b.$

These are the bits of 0.1 that were lost in the subtraction of 0.8 − 0.7 since their difference is nearly 8 times smaller than either of them individually. In other words, the errors in representing 0.8 and 0.7 were magnified by the cancellation. This shows that a − b will not equal c when compared as 64-bit values. Note that

```
>> 2^(-51) - 2^(-52) - 2^(-53) - 2^(-55)
ans = 8.3267e-17
```

which is what you obtain from a − b − 0.1.

For the second part:

The value 0.4 is also not a dyadic rational, so the 64-bit approximation gives

$0.4 \approx d = \frac{3602879701896397}{9007199254740992}$,

and using exact arithmetic we have

$0.8 - 0.4 \approx a - d = \frac{3602879701896397}{9007199254740992}$,

which is exactly representable with 64-bits and is equal to d. The reason we don't have the same issue as the previous part is that, ignoring overflow and underflow, x and x/2 have the same significand when approximated in 64-bit arithmetic. The subtraction (typically performed in 80-bit arithmetic) then gives the 64-bit approximation to x/2. It is tempting to think the lack of error here is fully explained by the fact that result of the subtraction is close to the order of the operands, but note that .7-.3==.4 gives an answer of 0 (false) debunking that theory.

Helpful links

1. https://en.wikipedia.org/wiki/IEEE_floating_point
2. Numerical Computing with IEEE Floating Point Arithmetic, Michael L. Overton (NYU)
3. Sec. 2.5, Numerical Mathematics, Alfio Quarteroni et al.
4. Sec. 2.1, Numerical Analysis in Modern Scientific Computing, Peter Deuflhard and Andreas Hohmann.
5. https://en.wikipedia.org/wiki/Dyadic_rational

6. https://en.wikipedia.org/wiki/Two%27s_complement
7. https://en.wikipedia.org/wiki/Double-precision_floating-point_format
8. https://www3.nd.edu/~coast/jjwteach/www/www/30125/pdfnotes/lecture5_9v14.pdf
9. http://math.nyu.edu/~stadler/num1/material/num1_interpolation
10. http://www.math.vt.edu/people/adjerids/homepage/teaching/S05/Math5466/interpolation.pdf
11. https://www.math.ohiou.edu/courses/math3600/lecture19.pdf
12. http://nikolavitas.blogspot.com/2013/09/cubic-spline-interpolation-periodic.html
13. http://vis.uni-kl.de/~alggeom/pdf/ws1213/alggeom_script_ws12_02.pdf
14. http://www.math.uconn.edu/~leykekhman/courses/MATH3795/Lectures/Lecture_15_poly_interp_splines.pdf
15. http://www.maths.lth.se/na/courses/FMN081/FMN081-06/lecture11.pdf
16. http://math.nyu.edu/~stadler/num1/material/assignment5.pdf
17. https://s3.amazonaws.com/piazza-resources/ie2ucbo1ftp4kb/ihz8liwjpvwpn/AA_hw5sol.pdf?AWSAccessKeyId=AKIAIEDNRLJ4AZKBW6HA&Expires=1460542999&Signature=SULn2odXi81HYtrMtl3SwZO9XRs%3D
18. http://www.wikihow.com/Convert-from-Decimal-to-Binary
19. http://www.wikihow.com/Convert-from-Binary-to-Decimal
20. http://www.wikihow.com/Add-Binary-Numbers
21. http://www.wikihow.com/Subtract-Binary-Numbers
22. https://en.wikipedia.org/wiki/Signed_number_representations#Sign-and-magnitude_method
23. https://en.wikipedia.org/wiki/Signed_zero
24. https://en.wikipedia.org/wiki/Two%27s_complement