Floating Point Arithmetic Notes

Travis Askham

October 14, 2016

1 Floating point

1.1 Floating point numbers

A double-format number is represented by 64 bits, which we label as

$$\pm \qquad a_1 a_2 \dots a_{11} \qquad b_1 b_2 \dots b_{52}$$

One of the bits is used to determine the sign of the number (in the standard 0 corresponds to positive, 1 to negative), eleven bits are used to determine the exponent, and the remaining fifty two bits determine what's called the mantissa. Assuming the exponent is right, the number of correct bits in the mantissa for any given calculation gives a sense of the relative accuracy of that calculation (we also often speak of significant digits or digits of accuracy, meaning the number of correct digits when the number is converted to base 10). For normal floating point numbers, all fifty two bits in the mantissa contain information. This is no longer true for "subnormal" numbers, which we will define below.

The value corresponding to a set of bits can be figured out from the following table

If exponent string is:	Then the value is:
$(00000000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 \dots b_{52})_2 \times 2^{-1022}$
$(00000000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{-1022}$
$(00000000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{-1021}$
$\vdots$	$\vdots$
$(01111111111)_2 = (1023)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{0}$
$(10000000000)_2 = (1024)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111111101)_2 = (2045)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1022}$
$(11111111110)_2 = (2046)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1023}$
$(11111111111)_2 = (2047)_{10}$	$\pm\infty$ if $b_i = 0\ \forall i$, NaN otherwise

A single-format number is represented using 32 bits, which we label as

$$\pm \qquad a_1 a_2 \dots a_8 \qquad b_1 b_2 \dots b_{23}$$

The value corresponding to a set of bits can be figured out from the following table

If exponent string is:	Then the value is:
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 \dots b_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 \dots b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 \dots b_{23})_2 \times 2^{-125}$
$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1 b_2 \dots b_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1 b_2 \dots b_{23})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111101)_2 = (253)_{10}$	$\pm(1.b_1 b_2 \dots b_{23})_2 \times 2^{126}$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1 b_2 \dots b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ if $b_i = 0\ \forall i$, NaN otherwise

Note that for either format, the numbers corresponding to the exponent zero are interpreted differently. The leading zero allows these "subnormal" numbers to take very small values, but at the cost of significant digits (i.e. we are "wasting" bits of the mantissa to make a smaller exponent).

1.2 Machine epsilon

The value of machine epsilon is determined by the next largest floating point number after the number 1. For normal numbers, this gives a sense of the relative error in rounding a real number to the nearest floating point number. If y is a real number in the range of the normal floating point numbers, then its floating point representation $\hat{y}$ satisfies $\hat{y} = y(1+\delta)$ for some δ with $|\delta| < \epsilon$.

- Double-format: $\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$
- Single-format: $\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$

1.3 Floating point arithmetic

With IEEE correctly rounded arithmetic, the results of arithmetical operations have nice error bounds and behavior. Let round be the function that represents a number as its nearest floating point approximation and a circled operation signify its floating point equivalent. Assume x, y are floating point numbers.

- $x \oplus y = \text{round } (x+y) = (x+y)(1+\delta)$
- $x \ominus y = \text{round } (x-y) = (x-y)(1+\delta)$

- $x \otimes y = \text{round } (x \times y) = (x \times y)(1 + \delta)$
- $x \oslash y = \text{round } (x/y) = (x/y)(1 + \delta)$
- $x \otimes 1 = x$
- $x \ominus y = 0 \Rightarrow x = y$

where $|\delta| < \epsilon$ (assuming that the result is a normal floating point number).

However, it's not all good. Indeed, it is possible that

$$(x \oplus y) \ominus z \neq \text{round } (x + y - z)$$

for x, y, and z floating-point numbers. Consider $x = z = 1$ and $y = (1 + \text{rand}())2^{-52}$ in double-format.

1.4 Some other important terms

- Overflow: when, in the course of a computation, numbers become too large to be represented and are replaced by infinity.
- Underflow: when, in the course of a computation, numbers become to small to be represented with normal numbers (so that they are replaced by subnormals or zero) and significant digits may be lost.

1.5 Qualifying exam type questions

- Find three double precision IEEE floating point numbers a, b, and c for which the relative error of $a + b + c$ is very large. Try to make it as bad as possible and explain your reasoning. Try where all numbers are normal and where you allow subnormal numbers.
- What is the smallest positive integer which is not exactly represented as a single precision IEEE floating point number? What is the largest finite integer which is part of the double precision IEEE floating point system?
- Find the IEEE single and double precision floating point representations (values of a_i and b_i of the numbers $4, 100, 1/100, 2^{100}, 2^{200}$, and 2^{1050} .
- What is the approximate value of machine epsilon in the IEEE double-format floating-point standard?

2 Numerical Stability

Because of the possible sources of error in floating point calculations (catastrophic cancellation, accumulation of errors, under/overflow, etc.) it is important to study what we call the numerical stability of any given algorithm. This is a broad subject; for now, we will concern ourselves with the basic definitions and some results.

2.1 Notation

First, some notation. Let $f : X \to Y$ represent our "problem". For a given input $x \in X$ we desire the output $f(x) \in Y$. In order to (approximately) solve the problem on the computer, we have any algorithm $\tilde{f}$ which produces an approximation $\tilde{f}(x)$ of $f(x)$. Let $\|\cdot\|$ be a given norm (in an abuse of notation, we'll use the same norm for points in X and for points in Y, even though, generally speaking, the norms could be different for these spaces). In what follows, ϵ will refer to the machine precision.

2.2 Definitions

- Stability: we say that an algorithm $\tilde{f}$ is stable if for any given $x \in X$ there exists $\tilde{x}$ such that

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(\epsilon)$$

and

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = \mathcal{O}(\epsilon)\,.$$

One might say that a stable algorithm gives "almost the right answer to almost the right problem". Here the $\mathcal{O}(\epsilon)$ is typically taken to mean some small multiple of the machine precision (while you can interpret it asymptotically, i.e. as $\epsilon \to 0$, we typically work with a fixed precision).

Stability is a reasonable requirement of any algorithm for solving a specific problem. Indeed, does your algorithm work if it's not stable? An example of an unstable algorithm is given by using a high order Taylor series approximation of e^x to evaluate e^x for large, negative x. This algorithm experiences catastrophic cancellation (and possibly overflow for very large x).

- Backward stability: we say that an algorithm $\tilde{f}$ is backward stable if for any given x there exists $\tilde{x}$ such that

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(\epsilon)$$

and $\tilde{f}(x) = f(\tilde{x})$, i.e. a backward stable algorithm is one for which the relative backward error is small.

This is a stricter requirement than stability (it is easy to see that backward stability implies stability). An example of a backward stable algorithm is given by using a robust implementation of a QR decomposition algorithm (e.g. the QR decomposition in MatLab) to solve the linear system $Ax = b$. The famous Gaussian elimination method with partial pivoting is known to not be backward stable for certain matrices. However, these matrices are rare (in some sense) and Gaussian elimination with partial pivoting is generally observed to be backward stable in practice.

- Condition number: the condition number κ of a problem f for a given input x is defined by

$$\kappa = \lim_{\delta \to 0} \sup_{\|\tilde{x} - x\| \leq \delta} \frac{\frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|}}{\frac{\|\tilde{x} - x\|}{\|x\|}} .$$

  If f is differentiable, then

$$\kappa = \|x\| \frac{\|J(x)\|}{\|f(x)\|} ,$$

  where J is the Jacobian of f.

  To see why the condition number is a useful concept, let's consider the relative error of a backward stable algorithm. We have

$$\begin{aligned}
\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} &= \frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|} && (1) \\
&= \frac{\frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|}}{\frac{\|\tilde{x} - x\|}{\|x\|}} \frac{\|\tilde{x} - x\|}{\|x\|} && (2) \\
&\leq \kappa \frac{\|\tilde{x} - x\|}{\|x\|} && (3) \\
&= \kappa \mathcal{O}(\epsilon) , && (4)
\end{aligned}$$

  where we have used the definitions of backward stability and the condition number. From the last line, we can reasonably say that the relative error in the solution of a well conditioned problem (well conditioned means small κ) using a backward stable algorithm is small (near machine precision).

  In some cases, we can use the second to last line above to be more quantitative. Suppose that $f(x) = A^{-1}x$, i.e. f is the solution operator for $Ay = x$. Then, $\tilde{x} = AA^{-1}\tilde{x} = Af(\tilde{x}) = A\tilde{f}(x)$.

Therefore, we can evaluate $\tilde{x}$ by multiplying the result of the algorithm by A. In particular, we know the error

$$\frac{\|\tilde{x} - x\|}{\|x\|} ,$$

in this case (called the residual error here). Therefore, given the residual and the condition number, we can use (3) to provide a more quantitative bound on the solution error $\|\tilde{f}(x) - f(x)\|$. (Note that for this bound backward stability is unnecessary).

What is the condition number for the problem $f(x) = A^{-1}x$? We have

$$\frac{\frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|}}{\frac{\|\tilde{x} - x\|}{\|x\|}} = \frac{\|A^{-1}\tilde{x} - A^{-1}x\|/\|A^{-1}x\|}{\|\tilde{x} - x\|/\|x\|} \tag{5}$$

$$= \frac{\|A^{-1}(\tilde{x} - x)\|}{\|\tilde{x} - x\|}\frac{\|x\|}{\|A^{-1}x\|} \qquad = \frac{\|A^{-1}(\tilde{x} - x)\|}{\|\tilde{x} - x\|}\frac{\|A(A^{-1}x)\|}{\|A^{-1}x\|} \leq \|A^{-1}\|\|A\| \tag{6}$$

so that $\kappa \leq \|A^{-1}\|\|A\|$. We define the condition number of a matrix to be $\kappa(A) = \|A^{-1}\|\|A\|$ based on this bound.

Therefore, if we solve $Ax = b$ (using any method), we have

$$\|\tilde{x} - x\| \leq \kappa(A)\|b - A\tilde{x}\| .$$

For a backward stable method, we should have

$$\|\tilde{x} - x\| \leq \kappa(A)\mathcal{O}(\epsilon) .$$