Comparing Reals vs. Comparing Floats

When programming with floats, we know that the assignment statement
\[ m = x \]
isn’t to be interpreted as an equation, but as
find a place in memory we will call \( m \), and store \( x \) there.

Some languages use other symbols, like ‘:=’ or ‘<-' (instead of ‘=’) to make it clear that this is assignment, not an equation. But sometimes we want “equals” as in “equation”, and programming languages need such a mechanism. For example, in the Matlab language
\[ m == 1 \]
returns TRUE if (the value in) \( m \) is 1, and FALSE otherwise.

So how are we to test the real variable equation \( x = y \) in floating point? The short answer is: we cannot! We first have to represent \( x \) and \( y \) as floats, say \( fx = \text{fl}(x) \) and \( fy = \text{fl}(y) \).

If \( x \) and \( y \) are in the floating point range, then the test
\[ fx == fy \]
will return TRUE iff their floating point representations are the same. What we are testing is whether or not there exists \( dx \) and \( dy \), with \( |dx|, |dy| \leq \mu \), for which \( x(1 + dx) = y(1 + dy) \) is a float. This implies \( |x - y| \leq \mu(|x| + |y|) \). But the converse doesn’t hold: for example, if \( x \in \mathbb{R} \) is exactly halfway between 2 neighboring floats, then for any \( \epsilon > 0 \), \( \text{fl}(x - \epsilon) \) and \( \text{fl}(x + \epsilon) \) are different floats. For example, there are \( x, y \in \mathbb{R} \) that do not overflow, which differ by \( 10^{290} \) for which \( \text{fl}(x) == \text{fl}(y) \) is TRUE (exponential spacing), and there are those that differ by \( 10^{-290} \) and return FALSE (binning). To test \( x == y \) in this case, I very rarely use anything more stringent than \( |fx - fy| \leq 2\mu \ast \max\{|fx|, |fy|\} \).

If \( fx \) and \( fy \) both underflow, the situation is different. We cannot give a relative bound like above, and subnormals make the situation complicated to talk about: The number \text{realmin} \ is the smallest positive normalized float, and in Matlab \text{realmin} \ is about \( 10^{-308} \). The floating point statement
\[ fx == 0 \]
is testing \( \text{fl}(x) \) against \( \pm 0 \), and depends on whether or not subnormals are used: if underflow is set to zero, then \( |x| < \text{realmin} \) means \( fx \) is set to \( \pm 0 \), while if subnormals are in effect, then \( |x| < \mu \ast \text{realmin} \) means \( fx \) is set to \( \pm 0 \). [Subnormals are the denormalized floats \( fx \), with \( |fx| \in [\mu \ast \text{realmin}, \text{realmin}] \); Matlab uses subnormals.]

Now the equations \( x = 0 \) and \( 1 + x = 1 \) are equivalent over \( \mathbb{R} \); they have the same solution set: \{0\}. But the real numbers \( x \) for which
\[ fx == 0 \]
is TRUE live in the interval \( (-\text{realmin}, \text{realmin}) \), while those for which
\[ 1 + fx == 1 \]
is TRUE are the real interval \( (-\mu, \mu) \). Since \( (-\text{realmin}, \text{realmin}) \subset (-\mu, \mu) \), we can say
\[ fx == 0 \implies 1 + fx == 1 \quad \text{but} \quad 1 + fx == 1 \nRightarrow fx == 0. \]

There are many floats for which \( 1 + fx == 1 \) is TRUE, but \( fx == 0 \) is FALSE. No normalized floats satisfy \( fx == 0 \), but (in double precision) almost 0.4 percent of all floats satisfy \( 1 + fx == 1 \). Another way of saying this (in double precision) is that about \( 7 \times 10^{16} \) of the about \( 2 \times 10^{19} \) floats are \( |\text{less than}| \mu \). How we test for “small” depends on why we are testing. Whether to use a relative measure, like \( \mu \), or an absolute, like \text{realmin}, is a problem-dependent – but fundamental – decision.