DATA 1010
In-class exercises
Samuel S. Watson
14 September 2018

Problem 1

Suppose that A is an $m \times n$ matrix, that $\mathbf{b} \in \mathbb{R}^m$, and that $\lambda > 0$. Find a formula for the value of $\mathbf{x}$ which minimizes

$$|A\mathbf{x} - \mathbf{b}|^2 + \lambda|\mathbf{x}|^2.$$

(For any square matrices that you would like to invert, you may assume they are invertible). Describe what happens (either to the minimizer, or to the original optimization problem) as λ increases from 0.

Problem 2

Show that the difference between any two consecutive positive `Float64` values is exactly representable as a `Float64`.

Problem 3

Select the numbers which are exactly representable as a `Float64`.

$$2^{1024} \quad \frac{4}{3} \quad -2^{-1074} \quad \frac{3}{8} \quad 0.0 \quad 1.5 \quad 0.8$$

Problem 4

In Julia, the function `nextfloat` returns the next largest representable value. Predict the values returned by the following lines of code, and then run them to confirm your predictions.

```
log2(nextfloat(15.0)-15.0)
log2(nextfloat(0.0))
log2(1.0 - prevfloat(1.0))
```

Problem 5

Investigate the rounding behavior when the result of a calculation is exactly between two representable values. Define `ϵ = 1/2^52` and check whether `1.0 + 0.5ϵ == 1.0`. Repeat with 1.5 in place of 0.5 (and appropriate changes made to the right-hand side). What does the rounding rule appear to be?

Bonus

Calculating inverse square roots is a very common task in graphics-intensive settings like video games. In the late 1990's, the following algorithm for approximating the inverse square root function appeared in the source code of the game *Quake III Arena*. The operator `>>` shifts the bits in the underlying representation over by 1 position, and `reinterpret` creates a new instance of the given type whose bits are the same as the bits of the given value.

```
function invsquareroot(x::Float32)
    y = 0x5f3759df - (reinterpret(Int32,x) >> 1)
    z = reinterpret(Float32,y)
    z * (1.5f0 - (0.5f0*x)*z*z)
end
```

this is syntax for an unsigned, 32-bit integer

If you are amazed by the appearance of the magic constant `0x5f3759df`* so was the original author. You can see their code comments on the Wikipedia entry for Fast Inverse Square Root.) Evaluate this function with a few input values and determine its relative error on each. Define another function which calculates the inverse square root in the obvious way (`1/sqrt(x)`) and check that the one above actually does run faster. As a double extra bonus, figure out why this code works.