Assignment 2

Floating Point Number Systems

Otto Hermann
otto.hermann@ucdconnect.ie
16203034

ACM40290

University College Dublin
9th October 2017

1 MachEps()

```
function [me_estimate, precision] = MachEps()

    % instantiate the estimate for machine epsilon
    me_estimate = 1.0;
    precision = 0;

    % use a while loop to refine the estimate
    while (1.0 + me_estimate) > 1.0
      me_estimate = me_estimate / 2;
      precision = precision + 1;
    end

    % return accounting for e/2 in the last step
    me_estimate = me_estimate * 2.0;
    precision = precision;

end
```

This solution used a while loop to iteratively determine ϵ_m, the difference between 1.0 and the next highest floating point number. If for a given value of ϵ_m, Matlab was able to distinguish between 1.0 and $1.0 + \epsilon_m$, I would divide ϵ_m by 2 and repeat. Once 1.0 and $1.0 + \epsilon_m$ were indistinguishable, I returned the penultimate value of ϵ_m i.e. $2\epsilon_m$, as it was the last value that allowed Matlab to delineate 1.0 and $1.0 + \epsilon_m$.

My results were identical to *eps* in Matlab, the built-in value of ϵ_m, so my method seems to work well. Also, knowing that our base is 2 and seeing that precision is estimated to be 53, we can check that $2^{-52} = 2.220446049250313e - 16 = b^{1-p}$.

2 Built-Ins

- eps
  - Floating-point relative accuracy
  - eps returns the distance from 1.0 to the next larger double-precision number, that is, eps = 2^-52.
- realmax
  - Largest positive floating-point number
  - realmax returns the largest finite floating-point number in IEEE® double precision.
- realmin
  - Smallest positive normalized floating-point number
  - realmin returns the smallest positive normalized floating-point number in IEEE® double precision.
- Inf
  - Infinity

  - Inf returns the IEEE® arithmetic representation for positive infinity. Infinity values result from operations like division by zero and overflow, which lead to results too large to represent as conventional floating-point values.

- pi
  - Ratio of circle's circumference to its diameter
  - pi returns the floating-point number nearest the value of π. The expressions 4atan(1) and imag(log(-1)) provide the same value.

- date
  - Current date string
  - date returns a character vector containing the date in the format, day-month-year, for example, 01-Jan-2014.

- ver
  - Version information for MathWorks products
  - A header containing the current MATLAB® product family version number, license number, operating system, and version of Java® software for the MATLAB product. The version numbers for MATLAB and all other installed MathWorks® products.

- version
  - Version number for MATLAB and libraries
  - version returns in ans the version and release number for the MATLAB® software currently running.

3 Alternating Infinite Series

An alternating infinite series is difficult to evaluate in floating point arithmetic because of the potential for catastrophic cancellation in subtraction and the resulting loss of precision. There is a significant potential for loss of precision if the numbers have the same exponent and the significant agrees to k digits.

This means that for a numeric approximation of say, the Maclaurin series of e^x with $x < 0$, you could try to minimise catastrophic cancellation in subtraction by rewriting the formula in a numerically suitable form; however, before embarking on this endeavour it would be sensible to make sure that the function isn't ill-conditioned, as no method can compensate for an ill-conditioned problem.

$cond(e^x) \approx |x|e^x/e^x = |x|$, so if $x < -1$ or $x > 1$, even a rewriting of the formula won't help.

4 Harmonic Series

With exact arithmetic, the evaluation of the Harmonic Series suggests it diverges:

$$\Sigma_{n=1}^{\infty} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} \ldots > \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \ldots = 1 + \frac{1}{2} + \frac{1}{2} + \ldots = \infty$$

In floating point arithmetic, a finite sum is achieved because of unit roundoff, which determines the maximum possible relative error in representing a nonzero real number in a floating point system. A system of floating point numbers has a bound of min and max. This means that in any floating point system, the approximation of $1/n$ as n grows large eventually reaches min ($realmin$ in Matlab); it also means that eventually the approximation of the series will reach max ($realmax$ in Matlab). Once at max, the system is unable to represent a higher number, so an approximation of the Harmonic Series will reach max or Inf. But because $1/n$ approaches min quicker than the approximation of the sum approaches max, a finite sum will be achieved because adding min will not be able to increase the approximation.

5 Matlab Response

- $\frac{0}{0} : NaN$, because you cannot divide zero by zero
- $\frac{Inf}{Inf} : NaN$, because both instances of Inf are too large to be represented in the floating point system, so we don't know what to divide
- $Inf \times 0 : 0$, because any number multiplied by zero is zero
- $Inf - Inf : NaN$, because both instances of Inf are too large to be represented in the floating point system, so we don't know what to subtract
- $0^0 : 1$, because any number to zero is one
- $Inf^0 : 0$, because any number to zero is one

Where NaN returns the IEEE® arithmetic representation for Not-a-Number. These values result from operations which have undefined numerical results. Inf is taken to be a number too large to be represented as a double (or whatever floating point system is employed).

6 Floating Point Arithmetic

By Hand
$a = 4/3 = 3/3 + 1/3 = 1 + 1/3 = 1.333333333\ldots$
$b = a - 1 = 1/3 = 0.333333333\ldots$
$c = b + b + b = 3b = 3(1/3) = 3/3 = 1$
$d = 1 - c = 1 - 1 = 0$

Matlab

```
function [a, b, c, d] = q6()
  format long;
  a = 4 / 3 ;
  b = a - 1 ;
  c = b + b + b ;
  d = 1 - c ;
end
```

$a = 1.333333333333333$
$b = 0.333333333333333$
$c = 1.000000000000000$
$d = 2.220446049250313e - 16$

Explanation
a is a double precision floating point approximation of $4/3$ and b of $1/3$; we can expect accuracy up to 15 digits and the maximum value of our relative rounding error, the unit roundoff u, to be ϵ_m (established earlier and also the value of d). Given that each of the first fifteen elements of the approximation of the infinite expansions of a and b is greater than ϵ_m (e.g. $3.0e - 16 > \epsilon_m$), we can see that Matlab presents to us values that match the hand calculations faithfully out to 15 digits.

Matlabs calculation of c also matches our intuition, but this occurs because of rounding error: 0.333333333333333 + 0.333333333333333 + 0.333333333333333 = 0.999999999999999, but Matlab returns 1.000000000000000 because of the errors in addition using Matlab (or any floating point system). EAS, in this case, is at most $(b + b + b) * u$ and $u <= \epsilon_m$, so EAS is possibly ϵ_m. Given that we are dealing with the sum of three numbers that each have maximum precision and sum to a number whose difference from 1.000000000000000 is ϵ_m, u will cause the sum to round up.

d returns ϵ_m because we are subtracting from an exact value of 1 an approximation of 1, which will, by its construction, differ from 1 by ϵ_m.