Finite-Precision Effects on Iterative Square Root and Squaring in MATLAB

Colton Blackwell

****Assuming f(x) = nthroot(y, 2); g(x) = y.^2, output function = Refer to figures below**

Purpose

This report investigates the effects of finite-precision arithmetic on a computational sequence involving square roots and squaring operations. Implemented in Matlab, the sequence theoretically leaves a number x unchanged for any nonnegative integer n, but finite-precision arithmetic can cause deviations/errors, particularly for large n. By examining outputs for x in the range $0 \leq x \leq 5$, the report presents plots, identifies the smallest n at which deviations occur, and discusses the behavior as n increases.

In real-world applications, rounding errors in finite-precision arithmetic can have significant impacts. This experiment demonstrates how some algorithms can become unstable due to error accumulation. By understanding the limitations of numerical computations, we can design better algorithms to mitigate such issues.

Observations

The parameter n was increased throughout the experiment to observe visual transformations in the output plots. For lower values of n (Ex. 1,2,3….), the graphs exhibited the expected linear characteristic resembling y = *x*. However, a notable change occurred around n=46, where distinct jagged "steps" became noticeable. Subsequently, as n continued to increase, these "steps" near/after x =1 became significantly pronounced, while the region [0, 1] retained a relatively consistent appearance, as depicted in Figure 1. Upon surpassing n=55, the observations indicated a convergence of the function toward 1 (fig. 2).

Understanding

What was observed in this experiment can be attributed to the fact that some arithmetic operation in a finite-precision system introduces a small rounding error because the exact result cannot always be represented within the limited precision of the floating-point format (MATLAB's default is Double-precision/64 bits). When iterative operations are performed (Ex. repeatedly taking the square root and then squaring) these errors accumulate.

For smaller n, the issues arising from finite-precision arithmetic are not as pronounced, due to the error introduced in one operation becoming the input for the next operation. For a small number of operations, the error introduced in each step does not propagate enough to cause a significant deviation. Additionally, when n is small, the individual changes to each y element are relatively minor. The effects of finite-precision arithmetic on these small changes are therefore less significant.

The contrast in "step" sizes between values smaller and larger than x = 1 in Fig 1. can be described via the original functions f(x) and g(x) (see function meanings above) exhibiting values within the interval [0,1] that closely align with the original $y=x$ function. However, beyond $x=1$ or higher x values, these functions deviate significantly from each other. The deviation occurs because of the fundamental properties of the square root and squaring operations. Specifically, f(x) plateaus as x increases, whereas g(x) experiences exponential growth with increasing x.

As shown in Fig. 2 and given n >= 53, the convergence of the output function to 1 becomes increasingly evident. As the number of loops (n) increases, the y-values of the output function become extremely close to 1 due to the property of repeatedly square-rooting. Some y-values may be so close to 1 that they exceed MATLAB's finite-precision arithmetic and get rounded (Ex. 1.00…023 => 1.0). Then, in the subsequent loop, squaring 1.0 remains unchanged, regardless of the number of iterations.

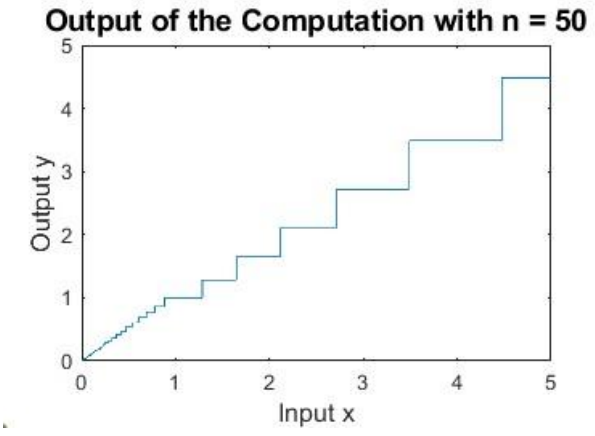

fig. 1

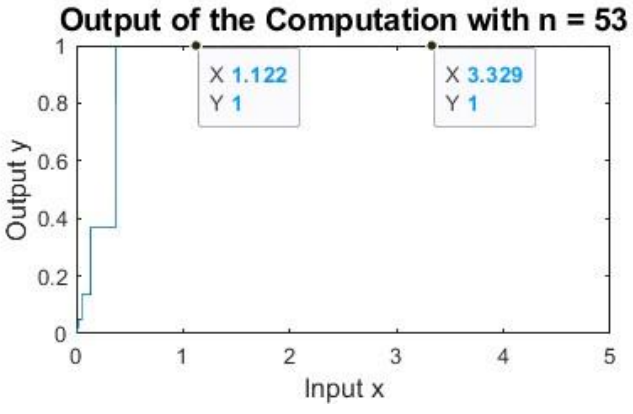

fig. 2

FloatPt.m (remain unchanged)

FloatPt.m

```
% MACM 316 - Homework 1
% Floating Point Arithmetic
% Description: Performs n-fold square-rooting following by squaring of
% the number x
% File name: FloatPt.m

clear
format long
n=50;
st=0.001; % Define a stepsize
x=0:st:5; % x is a row vector of numbers between 0 and 1 of increments st
y=x;

for i=1:n
    y=nthroot(y,2);
end

for i=1:n
    y=y.^2; % The '.' here means the squaring is carried out on each element of y
end

% Plot the output y versus the input x
plot(x,y)
title(['Output of the Computation with n = ' num2str(n)],'fontsize',14)
xlabel(['Input x'],'fontsize',12)
ylabel(['Output y'],'fontsize',12)
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> FloatPt
>> FloatPt
>> FloatPt
>> FloatPt
>> FloatPt
>> FloatPt
```