APC 523 – Homework #1

5.

The value of n-stop is 1x10^14 in order to get an answer that agrees to 13 significant figures. This seems to suggest that for every additional digit of convergence in the solution you need an additional multiply of 10 number of iterations.

The final value I converged to is   
S = 2.71611003408702

Here is a table of the n value and the estimate of e

n S

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 2

10 2.5937424601

100 2.704813829422

1000 2.716923932236

10000 2.718145926825

100000 2.718268237192

1000000 2.718280469096

10000000 2.718281694132

100000000 2.718281798347

1000000000 2.718282052012

10000000000 2.718282053235

100000000000 2.718282053357

1000000000000 2.718523496037

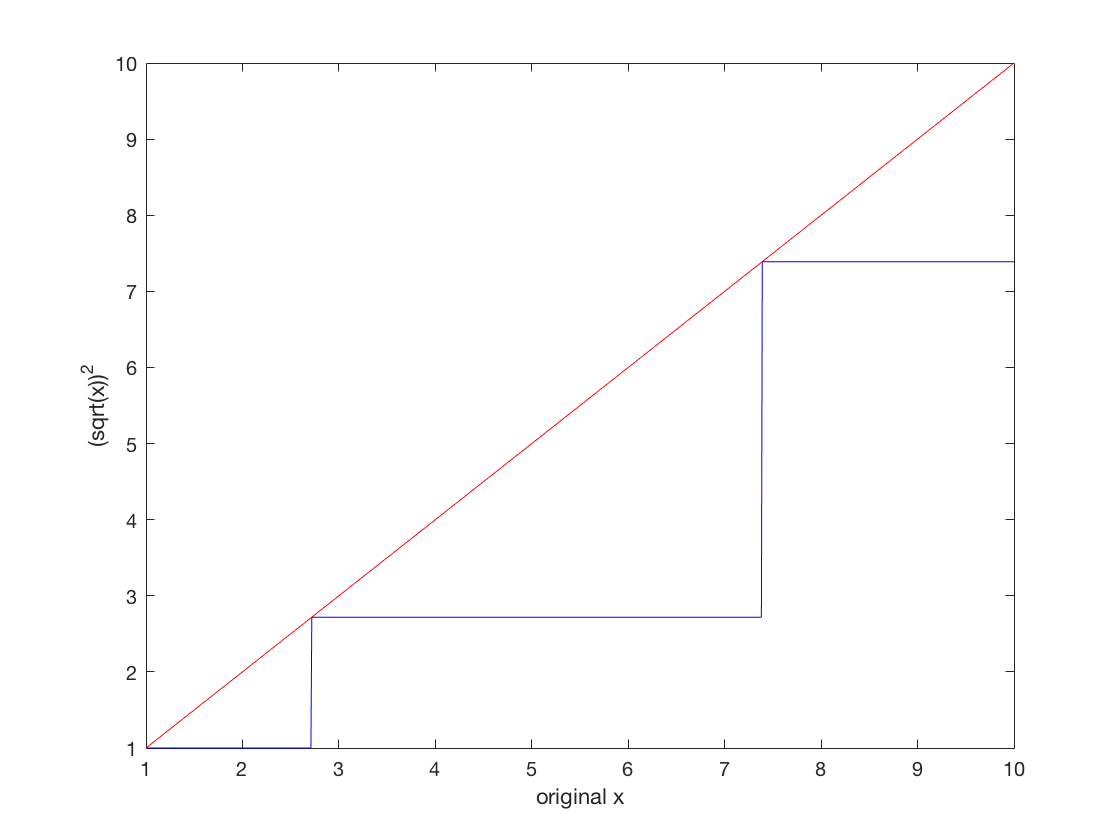
10000000000000 2.716110034087

100000000000000 2.716110034087

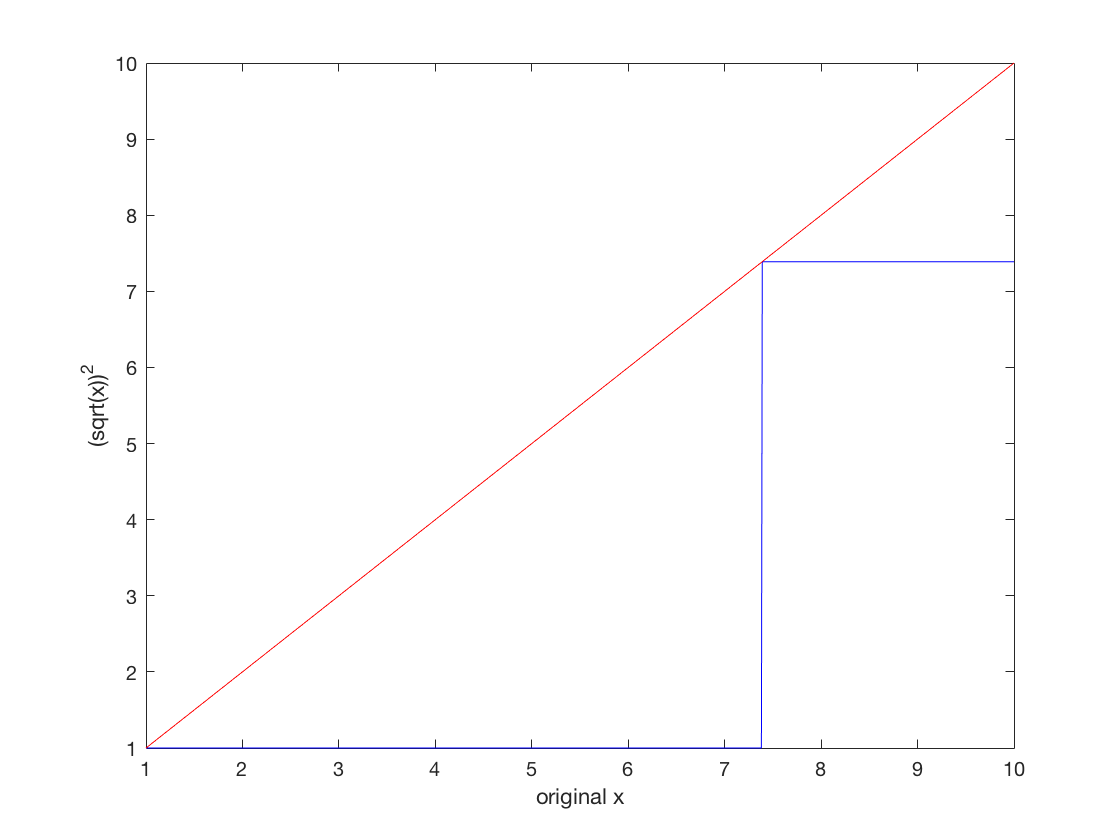
6.

The blue line shows the values of x that are recovered after taking the square root 52 times and then squaring 52 times. It can be seen that only 3 values are recovered, and these are 1, 2.7182…, and 7.38905…

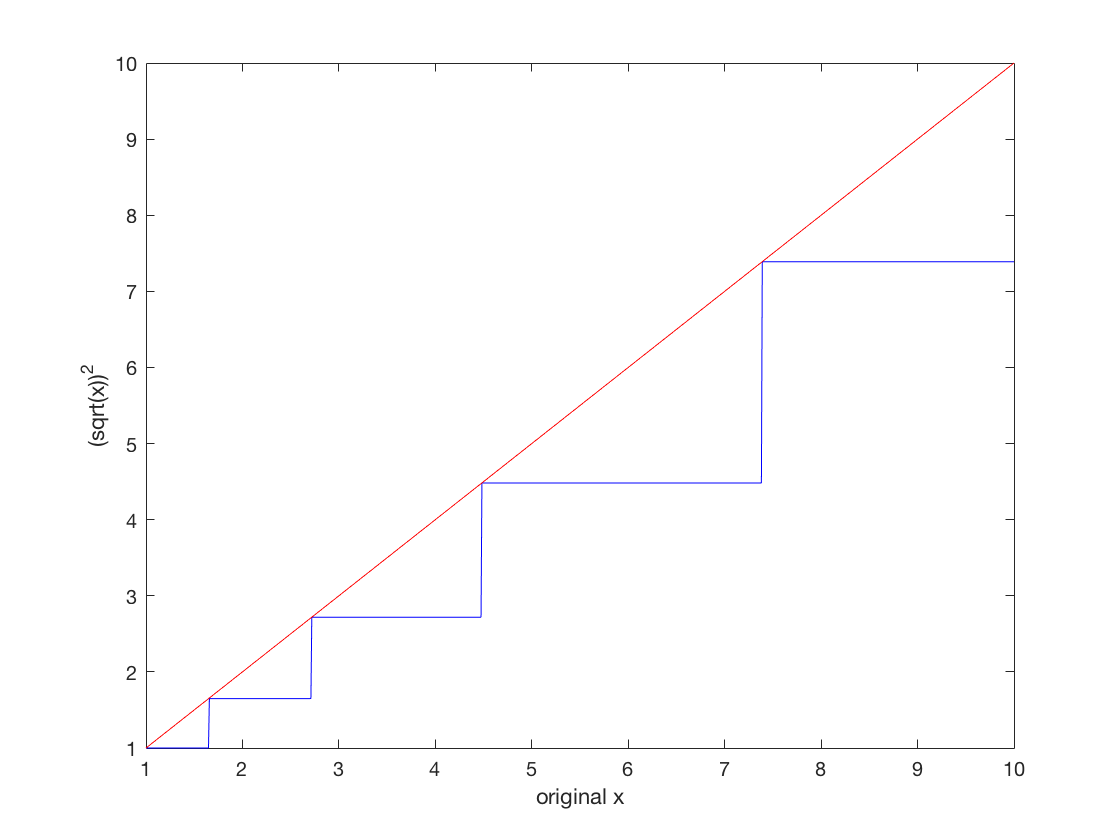
These values are close to exp(0), exp(1), and exp(2).



If we do the same thing but with 53 iterations of square rooting and squaring we get the following plot:



This plot looks similar except we are only able to recover 1 and 7.389…

Alternately, if we do 51 iterations instead of 52 we get the following:

Where we recover the following numbers: 1, 1.6487, 2.7182, 4.481689, 7.389055, which are close to exp(0), exp(1/2), exp(1), exp(3/2), exp(2).

We can understand what is going on by looking at the deviation from 1 after we have taken the square root 52 times. These values should be very small, since after taking the square root so many times we should end up with something that looks like 1 with a bunch of zeros in the decimal places and then a very small number. By looking at the deviation from 1 for the values recovered in the 52 iteration case we see that we get 0, 2\*10^-16, 4\*10^-6. This suggests that by taking the square root so many times we have pushed ourselves to the end of the storage limit of the mantissa. Thus, we only have 2 bits lefts and we can only represent 2 and 4. Then when we square everything again, we can only recover the numbers that would have actually been represented as 1.00000000…0001 with the bits after the truncation point being equal to zero anyway. For 52 iterations we can only recover 3 numbers, and if we take the square root again we are only left with 1 bit of storage so we can only recover 1 (trivial solution) and one other number.

If we check the deviations from 1 using 51 iterations we see the same trend:

0, 2\*10^-16, 4\*10^-16, 6\*10^-16, 8\*10^-16. But if we do one more square root operation we are no longer able to represent the number corresponding to a deviate of 2\*10^-16 (because this bit is no longer stored), or 6\*10^-16 (because this would require an additional bit to be represented so it gets aliased into the 4\*10^-16 number). This corresponds to losing exp(1/2) and exp(3/2) when we move from 51 iterations to 52 iterations.