Report:

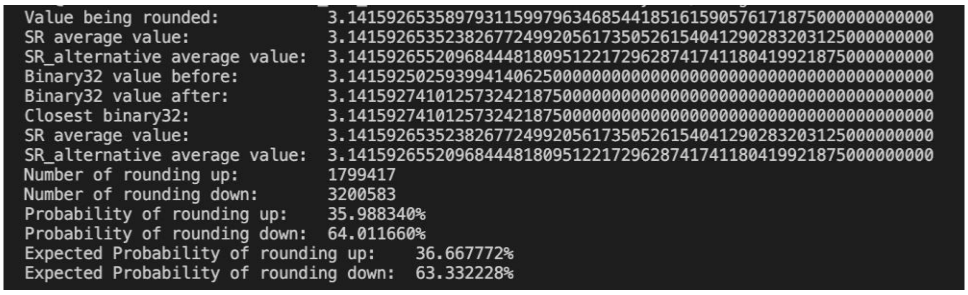
COMP36212 EX1
Testing and Applications of Stochastic Rounding

Ben Welland

Stochastic Rounding:

Stochastic Rounding alternative:

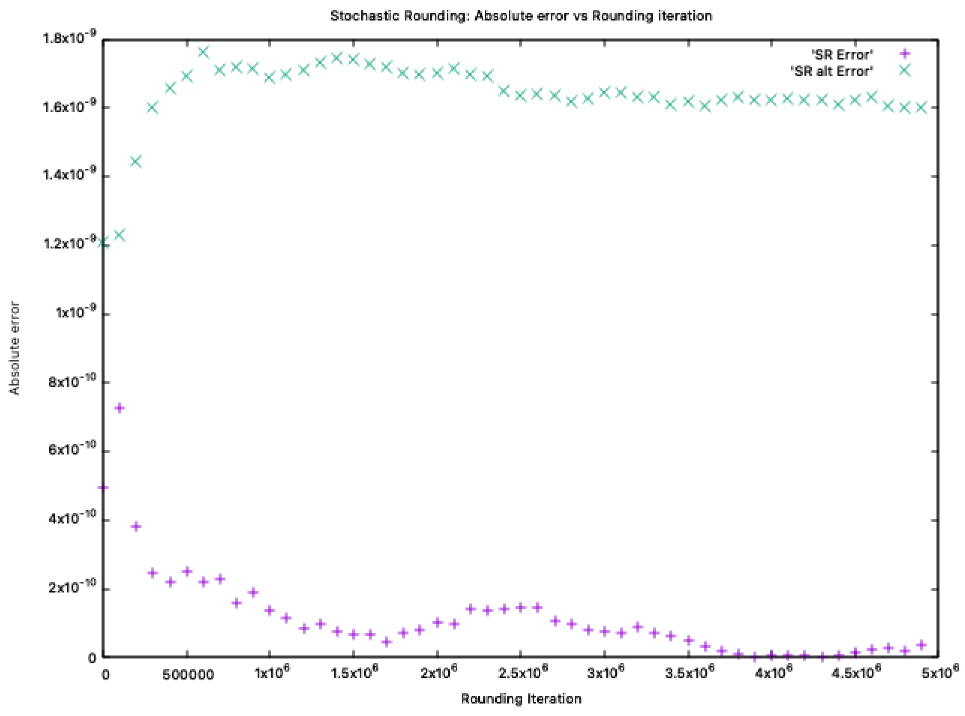
- Takes a parameter of a double, x (binary 64).
- Converts the parameter to a binary 32.
- Works out closest binary 32 numbers to the binary 64 parameter (get closest number both up and down).
- Creates a random number between 0 and 99.
- Work out P = x-RZ(x)/RA(x)-RZ(x), then multiply by 100 to get a percentage out of 100 to compare with random number.
- If random number < P then return the binary 32 number above (RA(x)).
- Else return the binary 32 number below (RZ(x)).
- Apply the SR to the number you are trying to round and then repeat it 5000000 times.
- Work out the average value by dividing the accumulated value by number of repetitions.

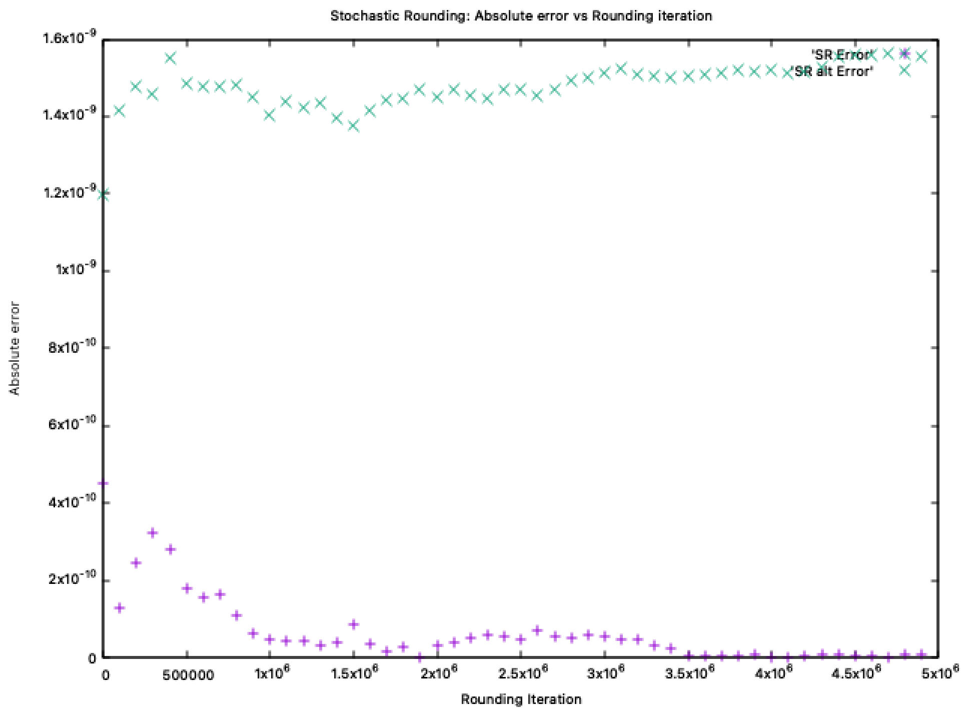


```
 Value being rounded:          3.14159265358979311599796346854418516159057617187500000000000
 SR average value:             3.14159265352382677249920561735052615404129028320312500000000
 SR_alternative average value: 3.14159265520968444818095122172962874174118041992187500000000
 Binary32 value before:        3.14159250259399414062500000000000000000000000000000000000000
 Binary32 value after:         3.14159274101257324218750000000000000000000000000000000000000
 Closest binary32:             3.14159274101257324218750000000000000000000000000000000000000
 SR average value:             3.14159265352382677249920561735052615404129028320312500000000
 SR_alternative average value: 3.14159265520968444818095122172962874174118041992187500000000
 Number of rounding up:        1799417
 Number of rounding down:      3200583
 Probability of rounding up:   35.988340%
 Probability of rounding down: 64.011660%
 Expected Probability of rounding up:    36.667772%
 Expected Probability of rounding down:  63.332228%
```

The average SR value always was more accurate than the SR alternative value this is probably because the SR function had a much greater range of random numbers that it could generate while the SR alternative could only generate integers from 0 to 99, missing the percentiles from 99-100. This gave the SR a greater range of numbers and can more reliably get a better split of probabilities for rounding up and down.

As the results showed, the probability of rounding up and down were fairly accurate, within 1% of the actual value, showing that the equations worked, and we can allow the slight difference as you will not get a 100% accurate answer when working with probability.





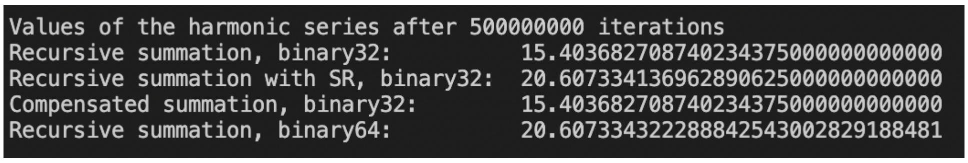
The graph looks accurate for SR. As the rounding iterations continue, it will get more accurate as it has more iterations to get an accurate probability split.

For example, if I threw a coin once I had a 50% chance of getting heads, but I could get a tail giving me a 100/0 split, whereas if I threw it 100 times then if I threw more tail than heads, I would get anywhere from 100/0 to 51/49 which is more accurate to the actual answer. Therefore, as the iterations continue the average of all the rounded numbers so far will be closer to the actual number.

However, for SR alternative, not only is the error much greater than for SR, but it gets more or less accurate at random, due to the probabilities. This is not what I expected as I explained earlier with more repetitions you reach the actual probability split more accurately so having this shows that there is a greater area of chance with SR alternative than with SR.

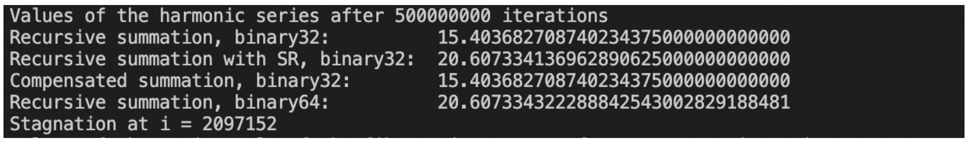
As the rounding iterations increase SR will always get more accurate to the binary 64 value, whereas the SR alternative can either get more or less accurate but will always stay within a margin of error of 1.4x10^-9 to 1.6x10^-9 from the binary 64 value.

Stagnation and the Harmonic series:



```
Values of the harmonic series after 500000000 iterations
Recursive summation, binary32:         15.403682708740234375000000000000
Recursive summation with SR, binary32:  20.607334136962890625000000000000
Compensated summation, binary32:        15.403682708740234375000000000000
Recursive summation, binary64:          20.607334322288842543002829188481
```

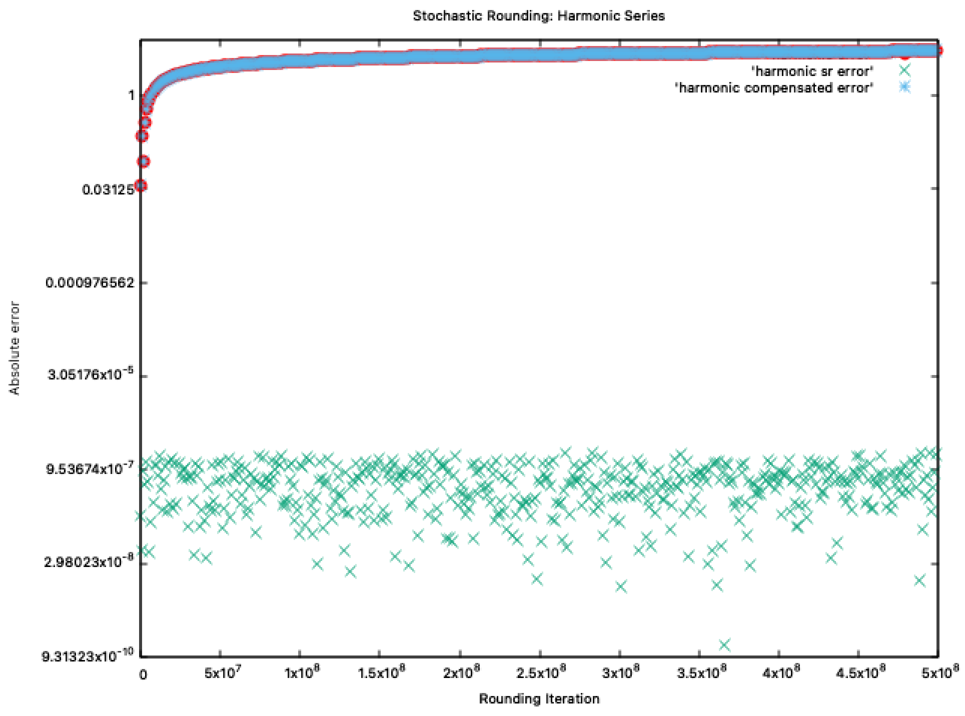
These results give us a fitting example of truncation in calculations. As is evident from the results, when using binary 32 instead of binary 64, there will be numbers added that will not change the value (In recursive summation with SR the calculations are done in binary 64 before being converted to binary 32 by the Stochastic Rounding function).



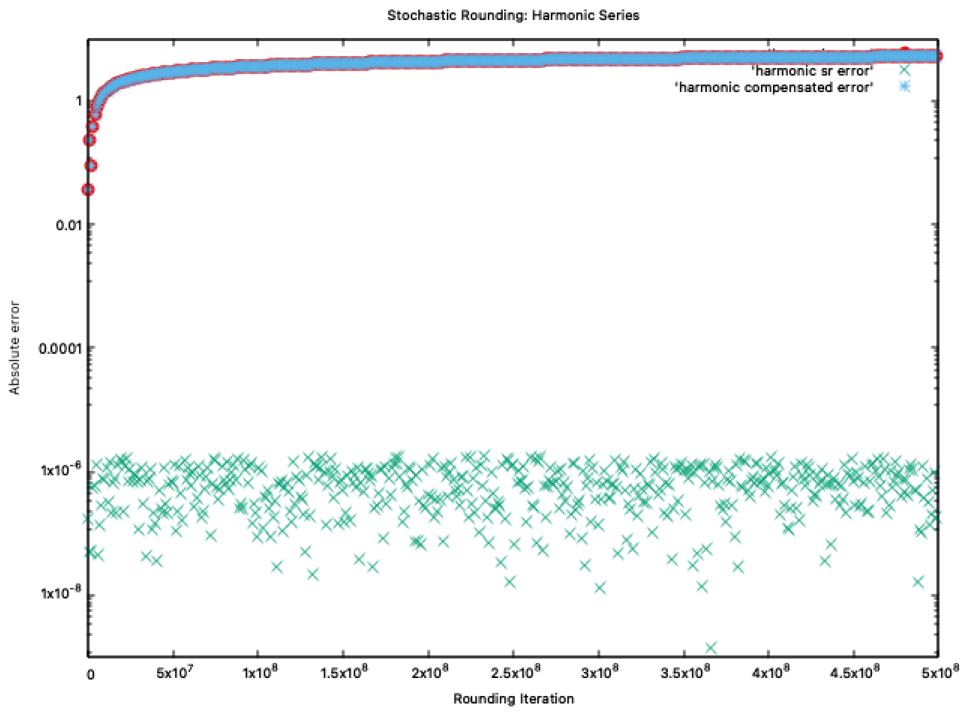
```
Values of the harmonic series after 500000000 iterations
Recursive summation, binary32:         15.403682708740234375000000000000
Recursive summation with SR, binary32:  20.607334136962890625000000000000
Compensated summation, binary32:        15.403682708740234375000000000000
Recursive summation, binary64:          20.607334322288842543002829188481
Stagnation at i = 2097152
```

Here we can see stagnation occurring at I = 2097152 which is about 2/500'ths of the total iterations. This shows not only how early it stagnates but how the first 2 million iterations of the reciprocals add up to 15.4 and only 5.2 is added in the next 498 million iterations, showing just how small the number added must be to stagnate. For example, here it would be around 0.000000000000000001 is the lowest number possible to add to binary 32 without stagnation. So, for 498 million iterations a number less than 1x10^-18 is added to the summation.

Logscale base 2:



Logscale base 10:



There is not a massive difference between the looks of these 2 graphs except that the y axis is much nicer in base 10 than 2. Here we can see the compensated error and the harmonic error increases with increasing number of iterations at a decreasing pace, which makes sense as they both have a small error the more calculations increase until stagnation where the value then does not change and only the error increases. Although, the error does increase it is still slow as you must be adding such small numbers for stagnation/truncation to occur.

We can see that the harmonic stochastic rounding error is small and varies in a zone around 1x10^-6. This is because the calculations are done in binary 64 and then it is converted to binary 32 through SR which creates this random error in a small margin due to the probability of rounding up or down.

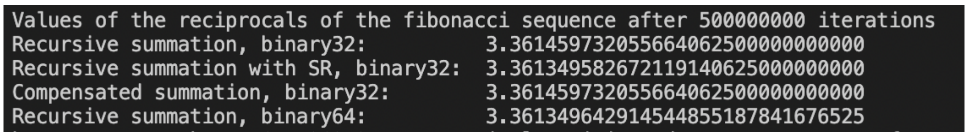
I have concluded from the experiment that the fast2sum can give out a small absolute error that it works out but the error won't factor in stagnation or truncation which can either lead to the error being very small or as large as the nomal binary 32 recursive summation. As the error returned from the fast2sum is around 1x10^-7 however we can see that by the end the absolute error is greater than 5.

Other Applications of SR:

For my application of Stochastic Rounding, I chose to apply it to a convergent series based on a sequence. I chose the famous Fibonacci sequence and decided to make my series the reciprocals of the Fibonacci sequence.

Example:

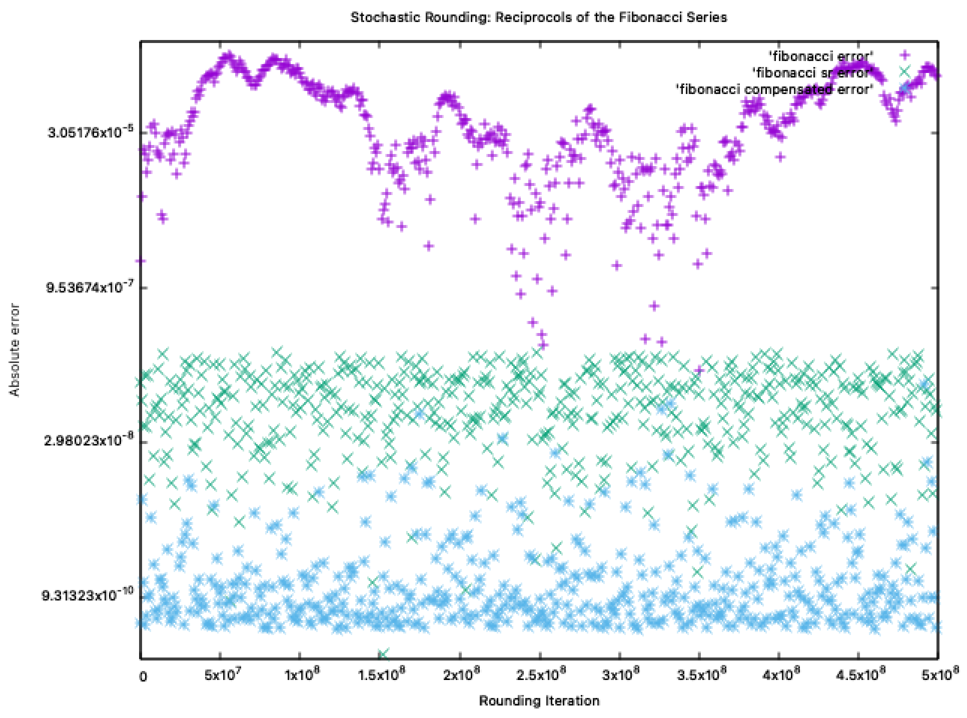
$$\frac{1}{1}+\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\cdots$$



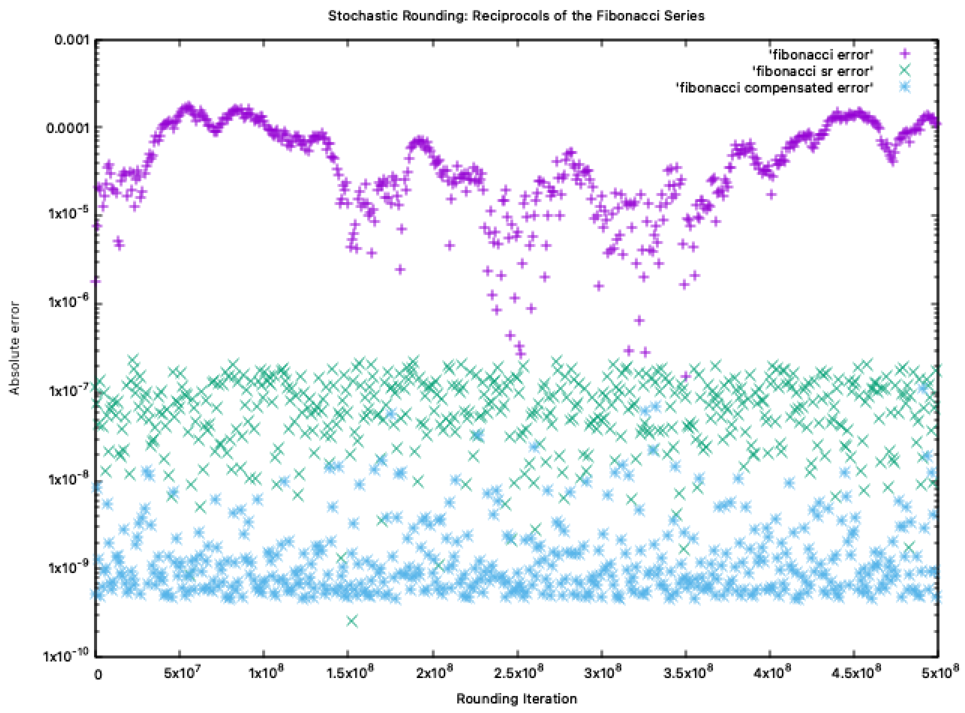
```
Values of the reciprocals of the fibonacci sequence after 500000000 iterations
Recursive summation, binary32:         3.36145973205566406250000000000
Recursive summation with SR, binary32:  3.36134958267211914062500000000
Compensated summation, binary32:       3.36145973205566406250000000000
Recursive summation, binary64:         3.36134964291454485518784167652
```

Here we can see it converging to 3.361

Logscale base 2:



Logscale base 10:



The Fibonacci error is very varied around 1x10^-6 and has these troughs and peaks that seem very different to the other graphs, this is due to errors from rounding either up or down when rounding from binary 64 to binary 32.

The stochastic rounding errors is small and again random within a margin due to the probability of it being rounded up or down.

The compensated method worked well here getting the smallest error margin of any of the tests ran due to the fast2sum having a small absolute error overhead.

CPU: M2 chip

GCC:

Apple clang version 14.0.0 (clang-1400.0.29.202)

Target: arm64-apple-darwin21.6.0

Thread model: posix