Module 1 – Shyam Natarajan

1. Integers are also real numbers. Rational numbers are those that can be expressed in the form p/q, where p and q are coprime integers (i.e. they are in reduced form) and q =/= 0.

2. Yes there is a rational number between every 2 rational numbers which is the average of the two numbers.

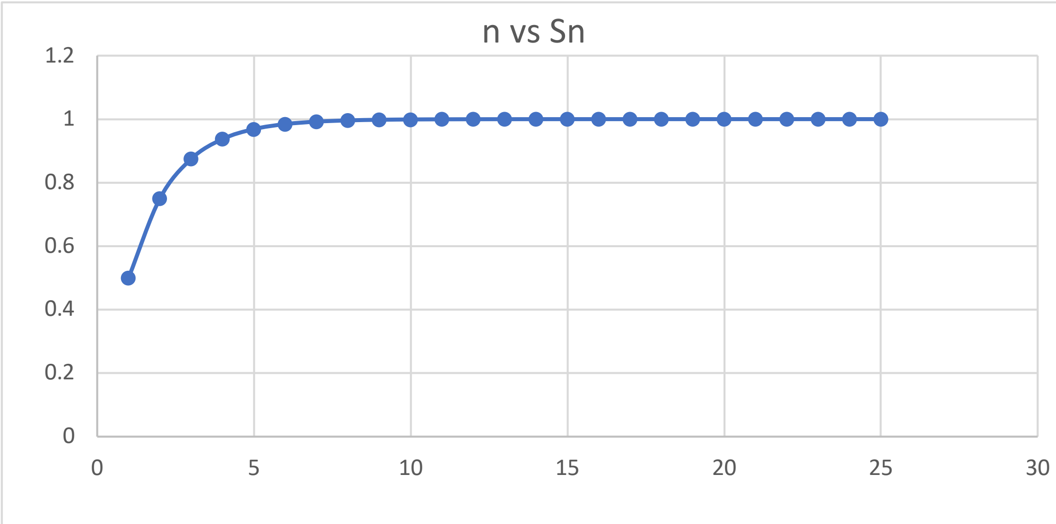
3. Integers, natural numbers and rational numbers all represent countable infinities and hence have the same cardinality. A basic non-rigorous explanation of rational numbers being countable is by representing all rational numbers in an infinite matrix where one axis is the numerator and the other in the denominator. Naturally any rational number would be present here and we can create a bijection onto the set of integers by counting along a zig zag pattern (i.e. 1/1, ½, 2/1, 1/3, 3/1 … mapping to 1, 2, 3 … and similarly for negative numbers. We would not need to consider both negative numerator and denominators so for simplicity the sign can depend on just the numerator with the denominator always being positive.)

4. A modified check would f(a) * f(m) > 0. This condition would only be met if both f(a) and f(m) are the same sign.

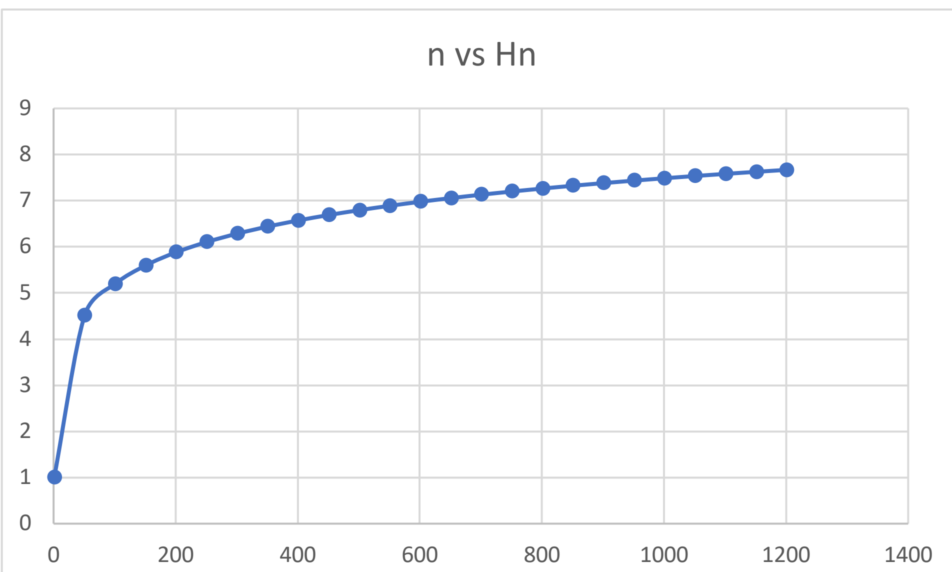
5. A calculator gives the output 1.73205080757 while the program gives the output 1.734375. The accuracy can be improved by putting a smaller value on the right hand side of the conditional "Math.abs (f(m)) > 0.01". This method can be written recursively by removing the while loop and instead of updating m, the new values for a and b can be passed to the same function with the value being returned once it is below the accuracy threshold. The efficiency of the program in this case would not improve since the number of computations and comparisons remains unchanged whether done iteratively or recursively.

6. As n increases, the value approaches 1.

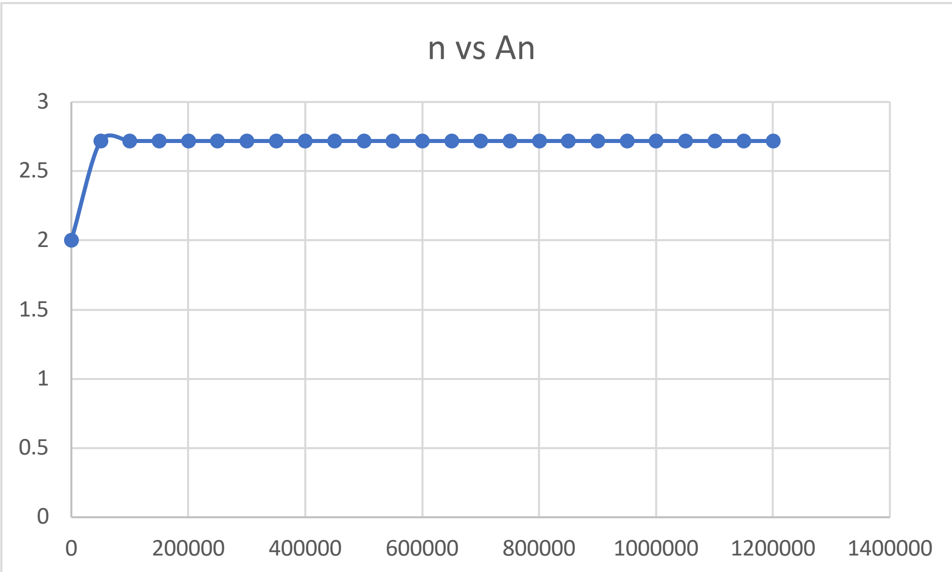
7. The sum approaches (n+1)/2. The double method is closer to the approached value than the float method.



8. The series does not converge to any value within the domain bounded by Integer.MAX_VALUE



9.



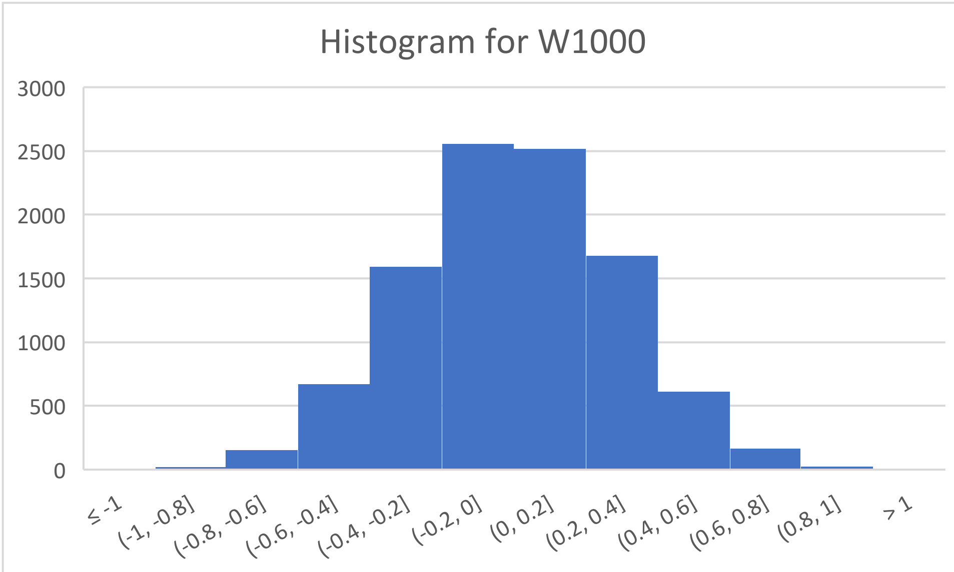
10. This sequence's sum oscillates between positive and negative but it approaches 0 in an irregular fashion, i.e it gets closer and farther as n increases.

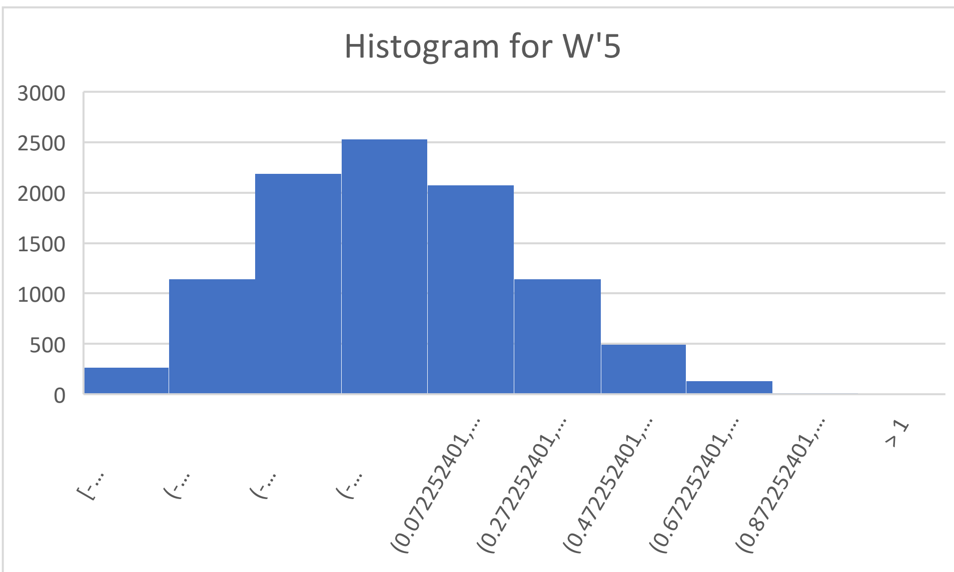
11. Un does not have a limit, but Vn approaches 0.5. However since there is no guarantee that it will ever get close to 0.5 this cannot be considered a limit. The difference between the approaches is than RandomSequence3 divides by n in each iteration so effectively each successive term is divided by a larger number so it is more like a sum where the nth term is (1/n)*(a random number between 0 and 1).

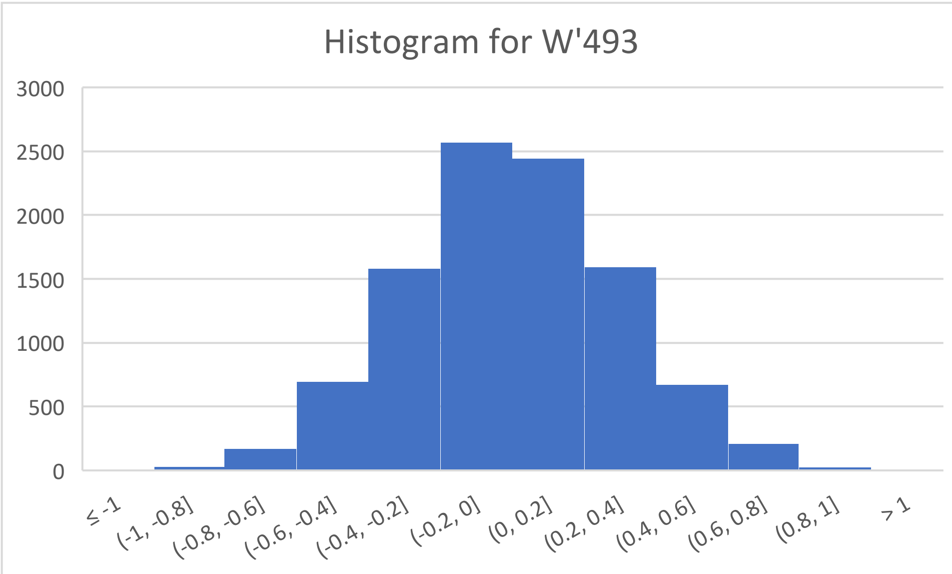
12. The histograms do not differ. This is intuitive as any Un term is just a random number between 0 and 1 and whether it is U1 or U99999 the underlying value is generated by the same random generation method, and over many samples it converges to being distributed uniformly across the intervals.

13. The higher the value of n, the tighter the distribution is around 0.5. This makes sense as the limit of this sequence is 0.5. For n = 10000, the minimum value is 0.490 and maximum is 0.512 which makes sense since with more samples we would expect the randomness to converge to the expected value.

14. The sequence Wn does not converge for large values of n. There are slightly more samples in the buckets closer to 0 for n = 493 but not to the same degree as the previous sequences.


Histogram for W1000
3000
2500
2000
1500
1000
500
0
≤ -1
(-1, -0.8]
(-0.8, -0.6]
(-0.6, -0.4]
(-0.4, -0.2]
(-0.2, 0]
(0, 0.2]
(0.2, 0.4]
(0.4, 0.6]
(0.6, 0.8]
(0.8, 1]
> 1

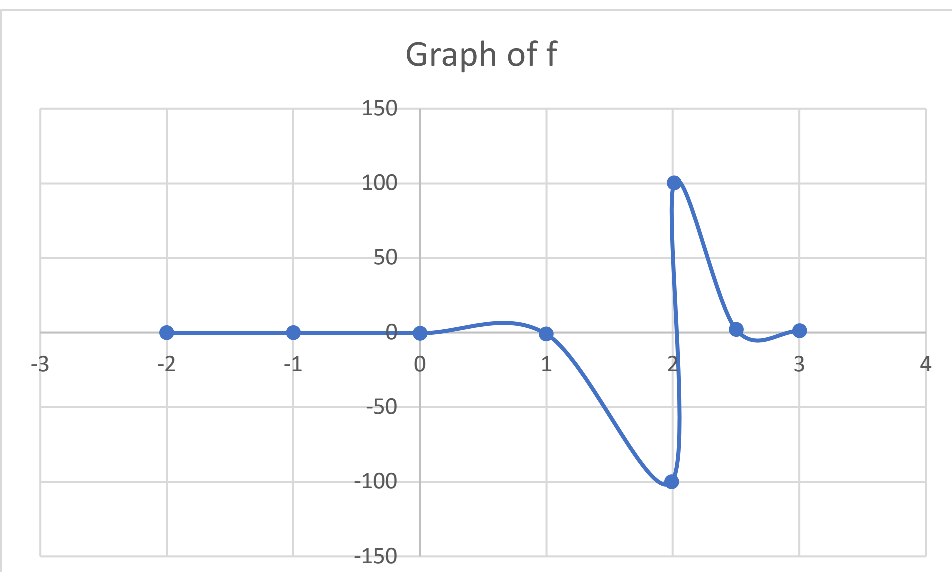

Histogram for W'5
3000
2500
2000
1500
1000
500
0
[-...
(-...
(-...
(-...
(0.072252401,...
(0.272252401,...
(0.472252401,...
(0.672252401,...
(0.872252401,...
> 1



15. An example is the function 1/(x−2), where the output would be infinite.

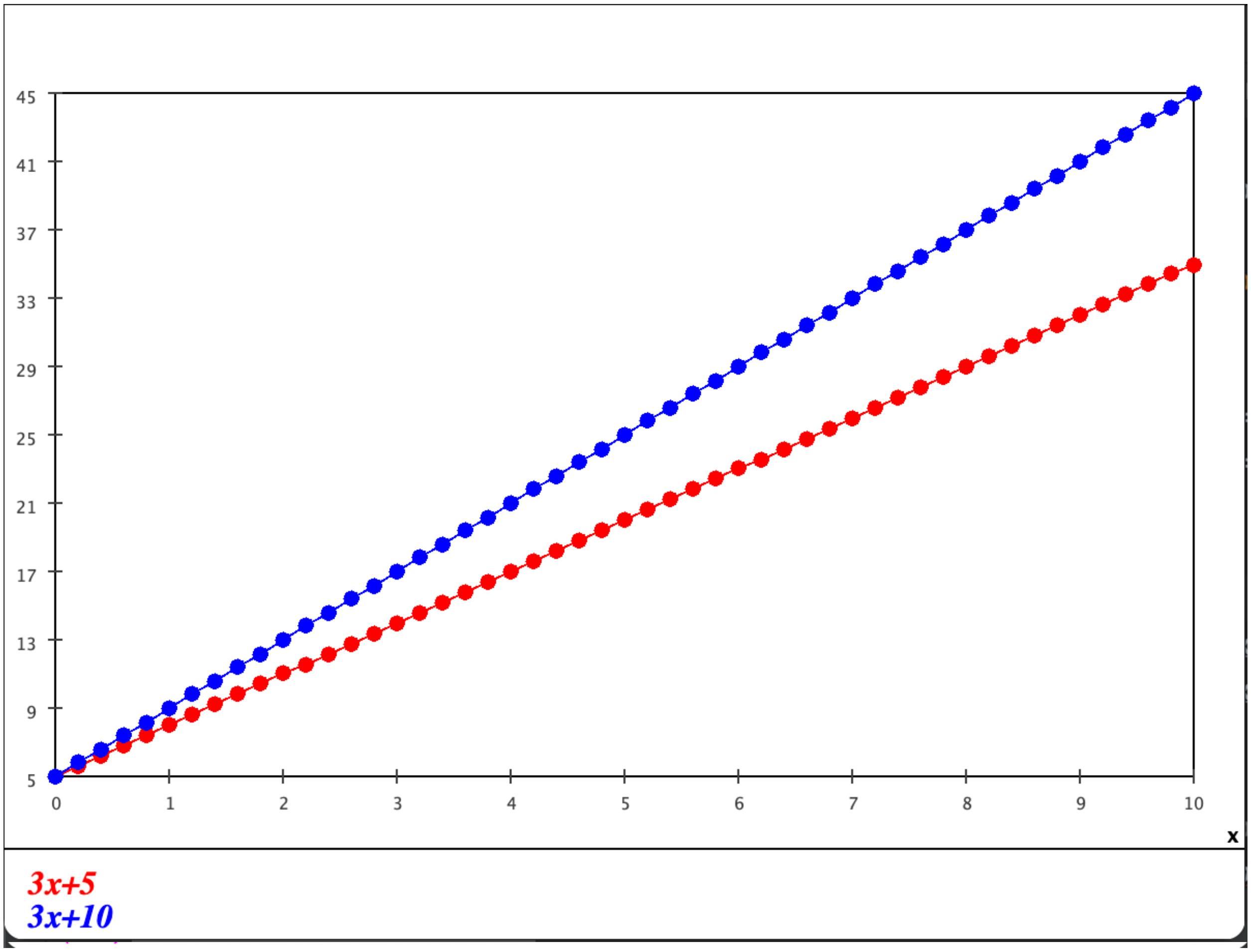
16. The domain is all real numbers and the range is all positive real numbers > 5. I drew it from −2 to 2 and it is a parabola with its vertex at (0, 5)

18. When x = 2 the output is infinity.



21. This is the graph that results. Note that I have not included the specified domain and instead offset so x=2 is not included since that causes the y axis to get compressed by the infinite value.

22.



23. The sign of the distance value depends on whether we use g-f or f-g. The same is true for using h = 20.

25. The distance roughly doubles. This shows that as the number of points we take increases, the total distance also increases.

26. The distance is inversely proportional to the number of points within a fixed interval, which is shown by the distance doubling when we increase from 50 to 100, and then growing 10 times from 100 to 1000. It is directly proportional to the size of the interval since it halves when we make the interval 5 from 10. The same happens for h showing that our distance metric is unreliable.

27. Yes, our metric is directly proportional to the number of points.

28. Rather than taking 500 points, we are taking 100 points but multiplying each addition by 1/5, which gives us the same effective computation.

29. This is due to integer division in java which has a range over the set of integers, and so 1/5 is represented as 0. This was not the case with the previous 10.0 etc. since the .0 signifies it is a floating point number.

30. It is the same as the area of the rectangle.

31. The calculated area is lower than the actual area but get closer with more intervals.

32. The distance is close to the difference in area.

33. The projection changes proportionally to the length of r.

34. It is negative as we are rotating the value and putting it in the 3^{rd} quadrant. Using 2pi brings the rotation back to 0 so it is the same as 4pi/3

35. They are periodic since it represents the ratio of a vector to its projection, or the sides of a right triangle. The period is 2pi.

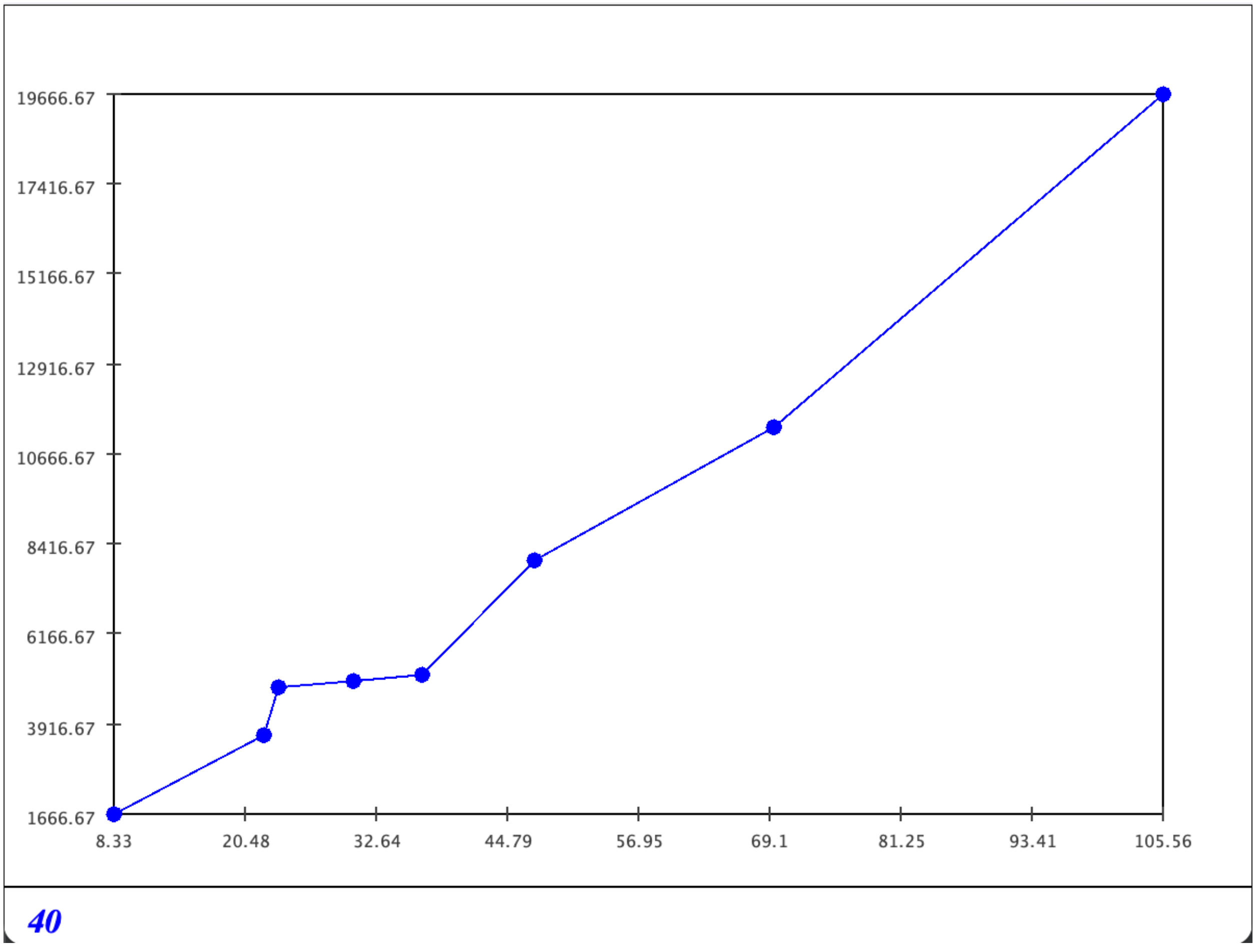
36. We can use smaller increments of x, which yields the desired minimum value -1.

37. The rule allows us to generalize for data we have not seen but generated by the same underlying process.

38. e+(pi*x)

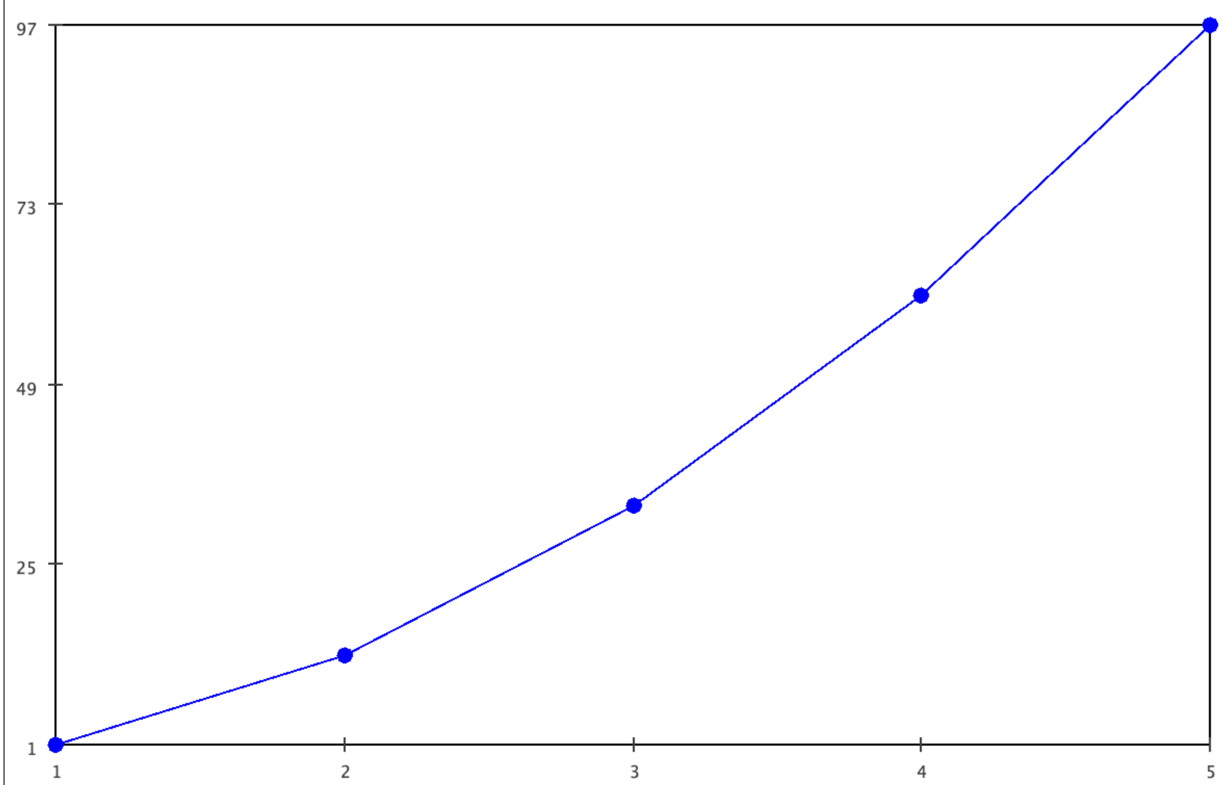
39. f(x) = 3x + 5

40.



19666.67
17416.67
15166.67
12916.67
10666.67
8416.67
6166.67
3916.67
1666.67
8.33
20.48
32.64
44.79
56.95
69.1
81.25
93.41
105.56
40

41.

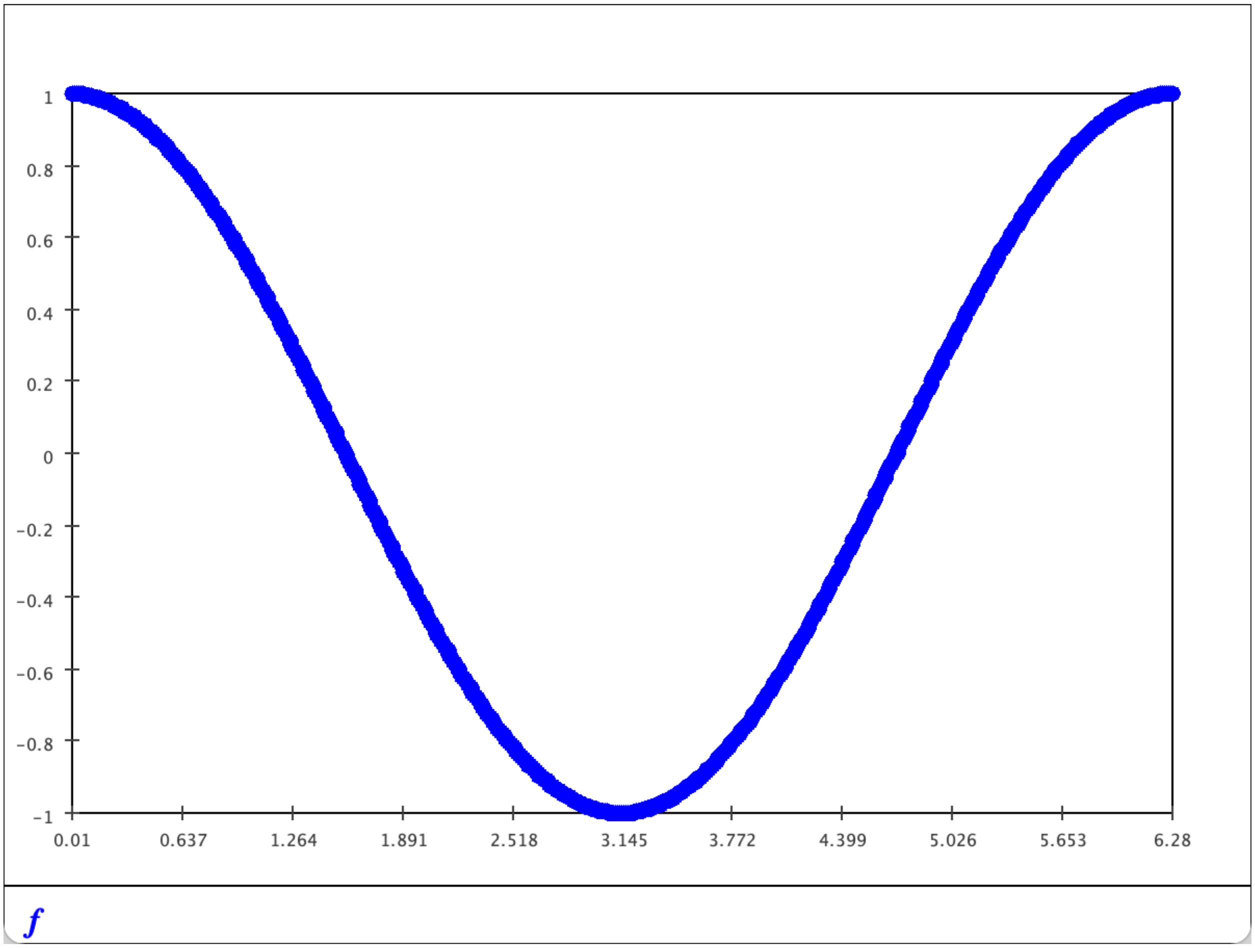


41

43. g(x) is close to 6x.

44. 3x^2 and 3x^2 + 5 have the same derivative. With a smaller value of d, the derivate value is closer to the actual derivative.

45.



46. f(x) is x^3 /3 and g(x) is x^2.

47. We can recursively compute the ith term for I from 1 to n, where we can determine n based on the precision we need. The default java representation is more precise. The java sin function is implemented with a 13 term polynomial over [0,pi/4]

48. The best acceleration function can be found by exploring the state space as a graph where nodes represent states and edges represent actions.

49. x/2 + x/3 + x/6