Homework 10 or something like that, Monte Carlo

I do have done part A, B and C

NOTE the monte carlo method is random, and in part A and I use the built in RNG, which is different each run, during my tests I did get all the tests to corrcetly work, but some times some will randomly fail. More often than not it works, I have left files from a run where all tests worked, more often than not each individual test works, though getting all at the same time is unlikely

ALSO NOTE , I use fairly high number of points, this means my results are accuracte, but it might take half a minute to run

Part A

Output: OutA.txt I print the result for each integral, and if it was inside target.

In part A I have implemented a plain, box shaped monte-carlo integration, which I test on the integral of xy from 0,0 to 1,1, the function f(x,y)=1 for ($x^{2+y}2<0.8\hat{}2$) 0 otherwise; and the given target integral. In both cases I occasionally come within error of the true result. I do think these two tests are enough, to see that the integration is working.

Part B

Output: OutB.txt I print the result for each integral, and if it was inside target.

In part B I implement the Halton sequence and try it with different prime basis, as suggested I use two different methods to estimate the error. In principle, the results are better, I get closer to the true value, and the error estimate is much much smaller (estimating the error as the difference between two different results foudn with different patterns (that is how I interpret the exercise at least).

Part C

Output: OutC.txt I print the result for each integral, and if it was inside target. points.png shows how the points fall for this integral

In part C I implement a strafified recursive method. I needed to use a set number of point, and calculate how to divide them, the way I do that is that I "sacrifice" 1/16 of the points (or the minimum number of points, whichever is larger) to make a plain monte carlo integral over the entire box, including calculing the sub-variance on every single dimension, then I pick whichever dimension had the largest variance, and divide the remaining 15/16 point between the two halves of this dimension proportional to the variance in each of the two halves. That way most of the points get spend where they are needed the most, and it does seem to work, at least the points (in the figure points.png) do cluster around wherever something interesting happens.

I know it is BAD that I sacrificed the first 1/16 points each recursion to see how we should divide the remaining points, but this is the only way I know as to how to get the sub-variances. These points sacrificed are what are visible on the graph points.png, I know I, in truth, only should have points around where the line is, but I could not find an algorithm for getting the sub-variance without evaluating any points.