One Last Project

This project is substantially easier than the last one, and should be doable in a week. If you have not yet finished the vibrating string project but are close to finishing, you're probably better off finishing it rather than starting on this one; this project is mostly intended for people who have already finished the previous one.

1 Overview

In this project you will calculate π using as simple a method as you could imagine, taking advantage of random numbers.

If you inscribe a circle of radius 1 inside a square of side 2, the area of the circle is π and the area of the square is 4; thus, the ratio of their areas is $\pi/4$.

This means that if you choose a point randomly inside the square, there is a $\pi/4$ chance that it will lie within the circle – and, if you choose many such points, the fraction that will appear inside the circle is also $\pi/4$. Thus, given a lot of random numbers, you can compute π!

2 Coding: random numbers

C has a built-in random number generator called `drand48()`; it returns pseudorandom numbers between 0 and 1. (To produce numbers in another range, you can do things like `x=drand48()*2-1;`, which will produce a random number between -1 and 1.)

This is a *pseudorandom* number generator. Since computers are deterministic, the PRNG will produce the same sequence of random numbers each time unless "seeded" with the function `srand48()`. The same seed produces the same sequence; this is useful if you want a predictable way to test your program. If you want a "truly" random sequence, you should do something like this:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
  srand48(clock()); // use current clock time (in microseconds) to seed drand48
```

```
  int i;
  double x;
  for (i=0; i<100; i++)
  {
    x=drand48();
    printf("Random number: %f\n",x);
  }
}
```

Note that I had to include two new headers: `stdlib.h`, which contains the random number functions, and `time.h`, to use `clock()`.

3 The assignment

1. Write a function, `estimatepi(int N)`, that generates N random (x, y) pairs within the square (-1, -1) - (1, 1), computes the fraction of those that lie within the unit circle, and then estimates π as four times that fraction. (You should not need arrays or anything fancy to do this.)

2. Run this function for $N = 10$ to 10^7 and make a table of your estimates of π. Does the trend match what you expect?

3. Now, evaluate the accuracy of your predictions in a rigorous way. Taking $N = 10^5$, call your function 100 times to produce 100 estimates of π. Compute the mean and standard error of the mean of those estimates (I'll discuss this in class; notes on how to do this are forthcoming Thursday). Quote a central value and error estimate for π. Is your result consistent with the analytical value of π?

4. Ambitious students and grad students: Starting with the definition of the variance $\sigma^2 = \langle x^2 \rangle - \langle x \rangle^2$ in a single random number (which you can compute analytically), calculate the expected standard deviation of your $N = 10^5$ estimates of π. Do they match the observed standard deviation when you actually did the 100 estimates in the previous part? Recall the difference between standard deviation and standard error.