Activity 2. Comparing the speed of two different implementations of an Algorithm. Central Limit Theorem

You can compare the speed of two different implementations of an algorithm with the Benchmark module bundled with the Perl distribution. You could just use a stopwatch instead, but that only tells you how long the program took to execute –on a multitasking operating system, a heavily loaded machine will take longer to finish all of its tasks, so your results might vary from one run to the next. Your program shouldn't be punished if something else computationally intensive is running.

What you really want is the amount of CPU time used by your program, and then you want to average that ovr a large number of runs. That's what the Benchmark module does for you.

Let's say you want to compute this infinite fraction:

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+.....}}}$$

At first, this might seem hard to compute because the denominator never ends, just like the fraction itself. But that's the trick. The denominator is equivalent to the fraction. Let's call the answer x.

$$x = \frac{1}{1+\frac{1}{1+\frac{1}{1+.....}}}$$

Since the denominator is also x, we can represent this fraction as

$$x = \frac{1}{1+x}$$

That is equivalent to the familiar quadratic form:

$$x^2 + x - 1 = 0$$

The solution to this equation is approximately 0.6180334. It's the Golden ratio –the ratio of successive Fibonacci numbers, believed by the Greeks to be the most pleasing ratio of height to width for architecture. The exact value of x is the square root of five, minus one, divided by two.

We can solve our equation using the familiar quadratic formula to find the largest root. However, suppose we only need the first three digits. From eyeballing the fraction we know that x must be between 0 and 1; perhaps a *for* loop that begins at 0 and increases by 0.001 will find x faster. This is what we will find out in this activity.

1 Activity

We are going to use the Benchmark module in Perl to see which of the implementations mentioned above is faster to solve the quadratic equation.

The Perls program is

```
#!/usr/bin/perl

use Benchmark;

sub quadratic { # Compute the largest root of a quadratic polynomial
     my ($a, $b, $c) = @_;
      return (-$b + sqrt($b*$b - 4*$a * $c)) /2*$a;
   }

sub bruteforce {  # Search linearly until we find a good-enough choice
        my ($low, $high) = @_;
        my $x;
     for ($x =$low; $x <= $high; $x += 0.001) {
         return $x if abs($x* ($x+1) - 0.999) < 0.001;
        }
}
```

```
timethese(10000, {quadratic => 'quadratic(1,1,-1)',
                  bruteforce => 'forloop(0,1)'    } );
```

After inluding the Benchmark module with *use Benchmark*, this program defines two subroutines. The first computes the larger root of any quadratic equation given its coefficients; the second iterates through a range of numbers looking for one that's close enough. The Benchmark function *timethese()* is then invoked. The first argument, 10000, is the number of times to run each code snippet. The second argument is an anonymous hash with two key-value pairs. Each key-value pair maps your name for each code snippet. After this line is reached, some statistics will be printed on your computer. According to these statistics, which implementation is faster according to the average CPU time?

We are more interested in the CPU time (cpu=user + system, abbreviated *usr* and *sys* in the Benchmark module results) than in the first number, the real (wall clock) time spent. MEasuring CPU time is more meaningful. In a multitasking operating system where multiple processes compte for the same CPU cycles, the time allocated to your process (the CPU time) will be less than the "wall clock" time (the 53 and 5 seconds in this example).

1.0.1 Predicting the probability that average CPU time will be a given amount

What you did above is only one run of what you need to do. We would like to collect about 1000 values of the average CPU time and see how these averages are distributed.

So manage to run the above program 1000 times, and store the result of each implementation in separate variables, called quadraticxbar and bruteforcexbar.

When done collecting the averages, analyze separately the two variables, doing a frequency distribution and graph of each, finding the summary statistics for each. What is the probability that the average for each implementatio is larger than 5 seconds?

After that, create a new variable called meandiff=bruteforcexbar-quadraticxbar. Find the frequency distribution and graph. According to your results, what is the probability that the difference between the two implementations' average CPU time is larger than 20 secs?