Monte Carlo Methods

- We often need to approximate some realistic model
- Getting exact data is too difficult, sometimes essentially impossible
  - Within realistic expectations
- E.g. write a program to compute pi
  - :(

Monte Carlo Methods

- But, we can approximate pi by:
  - Sample uniformly from $[0,1]^2$
  - Count how many of those two-dimensional points are more than length 1 away from the origin
  - The ratio of those points to the total should approach pi/4
    - Because a unit circle has pi/4 of its area in the first quadrant

Other Numerical Concerns

- Many models require integration
  - Remember related rates, physical flow, change in volume, etc. from calculus!
- Many integrals do not have a closed form
- We can approximate integrals similar to how we approximated pi (which was really the area under a curve!)

$$\int_a^b f(x)dx \approx (b-a)\frac{1}{N}\sum_{i=1}^{n} f(x_i)$$

Monte Carlo vs Riemann

- Looking closely at the above, this is similar to classical Riemann sums
- Slight problem: in d dimensions, we would need a huge number of points: N^d
  - :(
- But we are saved by parallelism!
  - Do many different simulations in parallel
  - Take the average of all of them

Monte Carlo Estimation

- If a single, sequential, estimator has a standard deviation of s, then the mean of N independent versions will have standard deviation s/sqrt(N)
- So, more simulations = more accurate
- Requirement: having access to “good” random number generation

Pseudo-Random Numbers

- Recall that standard random numbers are generated simply by “clock” arithmetic:
  - Start at seed x_0
  - Return random number k by computing

    $x_k = (a*x_{k-1} + b) \bmod m$
  - Where *a*, *b*, *m* are hard-coded
  - The period is *m*, usually a large power of 2, e.g. 2^{31}

Lagged Fibonacci Numbers

- More general formulation, for any binary operator:

$$X_i = X_{i-p} \otimes X_{i-q}$$

- Where p and q are hard-coded with initial values
  - The randomness will be sensitive to this
  - More choices = more chance that the numbers don’t actually look that random

Parallel Random Numbers

- How? With the previous recurrence, the parallelism is not directly trivial
  - Shared memory = big bottlenecks
  - Split sequences, if starting points are close together (e.g. adjacent seeds with big step)
    - But will be very strongly correlated (i.e. not random-looking)
  - Can use one process to generate “random” start points for other workers (not too shabby)

Example: Ising Model

- Situation: modeling magnetism, where there are some atoms arranged in a “lattice” and they have spin, which dictates the magnetic field