Sommersemester 2013, Struktur und Simulation

Übung: Simple Monte Carlo Integration

Assignment due date: 2.5.2013

In this exercise we will implement a simple Monte Carlo algorithm. This exercise has to be handed in two weeks, but there will be a second exercise on Monte Carlo handed out next week. Please send your answers to the questions in PDF or plain text format as well as source code to ast_uebung[at]zbh[dot]uni-hamburg[dot]de (and please include your full name). There should be one homework per person, no group submissions please. Discussing ideas during the exercise is ok, but you should only hand in answers and source code you have written yourself.

Pseudorandom numbers

Monte Carlo algorithms take their name from the random numbers they use. On a computer, we use so-called pseudorandom number generators, which are deterministic algorithms that produce numbers that look sufficiently random. All of these algorithms take some kind of "seed" value, which can be used to recreate the sequence of random numbers exactly. In practise this means that all your programs using random numbers should accept a seed value from the command line so that your calculations become reproducible.

It is important to use a good pseudorandom number generator for Monte Carlo algorithms as one can otherwise get bad results. In the C programming language, you can use the `srand48(seed)` function to set the seed once at the beginning of your program and the `drand48()` function to generate a uniformly distributed random number in the interval $[0.0, 1.0)$. More information can be found on the manpages by typing `man srand48` and `man drand48` at the shell prompt.

Calculating π by Monte Carlo integration

This example should already have been mentioned in the lectures. Imagine a square with side length 1 centered at $(0.5, 0.5)$, and inside it a circle with radius 0.5 also centered $(0.5, 0.5)$. The square has the area $A_s = 1$, and the circle the area $A_c = \frac{\pi}{4}$. If we select n points at random from the square and count the number of points k that lie inside the circle, the ratio $\frac{k}{n}$ will converge to $\frac{A_c}{A_s} = \frac{\pi}{4}$ as n goes to infinity. After n steps, our best estimate for π is $4\frac{k}{n}$. We can estimate the standard error s (of our estimate for π after n steps) as

$$s = 4\sqrt{\frac{pq}{n}}$$

where $p = \frac{k}{n}$ and $q = 1 - p$.

What is being integrated here? If we define a function

$$f(x,y) = \begin{cases} 1, & \text{if } \sqrt{(x-0.5)^2 + (y-0.5)^2} < 0.5 \\ 0, & \text{otherwise} \end{cases}$$

then we are calculating its integral. In this case, we know the correct answer, so it is easy to see how good our Monte Carlo estimate is after a given number of steps.

Assignment

Write a program that calculates an estimate for π as described above as well as an estimate of the standard error s. Make sure you set the seed at the beginning of your program once before generating any random numbers. Print out the current estimates for π and the standard error s to the screen and plot them after the run with gnuplot.

Run your program for up to 100, 1000, 10000 steps (and more if you want to). How fast does the estimate for π converge to the true value? Also consider the change in the size of the error bars with time. Attach relevant plots that support your arguments. What does the formula for the standard error tell you about the speed of convergence, e.g. how does the number of steps n have to change if we want to reduce the standard error by a factor of 10?

Your program should take three command-line arguments

```
./pi seed nsteps nstep_print
```

where `seed` is the seed value for the random number generator, `nsteps` the total number of steps that should be performed, and `nstep_print` is the number of steps between printing out results.

An example run:

```
$ ./pi 23 4 1
         1        0.000000        -3.141593        0.000000
         2        2.000000        -1.141593        1.414214
         3        2.666667        -0.474926        1.088662
         4        3.000000        -0.141593        0.866025
```

The first column is the step number, the second the current estimate for π, the third column is the difference between our estimate and the true value of π, and the fourth column contains the estimate for the standard error s.

C programming notes

Integer division in C is truncating:

```
int a = 1, b = 2;
a / b == 0;
```

If you want to get the fractional part of the answer as well, you have to convert ("cast") one of the variables to a floating-point type (e.g. float or double):

```
int a = 1, b = 2;
(double) a / b == 0.5;
```

Many maths functions such as `sqrt`, `sin`, or `log`, require you to link to the standard math library. You can do this when linking/compiling with `gcc` by appending `-lm` to the command-line:

```
gcc -Wall -Wextra -O3 foo.c -o foo -lm
```

Help on plotting with gnuplot

If you print your results in the order

```
n, pi_estimated, (pi - pi_estimated), s_estimated
```

and store the output in a file

```
./pi seed 100 1 | tee out.dat
```

Then you can start gnuplot and type

```
plot 'out.dat' using 1:2:4 with errorbars, 3.141592653 lw 2
```

If you only want every 10th datapoint (this is useful when you have many rows in the file you want to plot), use

```
plot 'out.dat' every 10 using 1:2:4 with errorbars, 3.141592653 lw 2
```

The numbers 1, 2, and 4 refer to the columns of the file `out.dat`. Column 1 (the step number) gets used for the x-axis, column 2 for the y-axis, and column 4 is used for the error bars. Save your plot to a file by typing

```
set terminal png
set output 'out.png'
replot
```

If you want the old behaviour of showing the plot on the screen back, type

```
set terminal x11
```