Programming techniques for physical simulations
Exercise 13

December 9, 2009

Monte Carlo Integration

Calculate the value of π using Monte Carlo integration using `boost::lagged_fibonacci607` and `drand48` as your random number generator. In order to use a boost random number generator you need to include the `boost/random.hpp` header.
See the documentation on `http://www.boost.org`.

- Draw random numbers and check whether they are within the unit circle. The number of hits divided by the total number of trials gives you an estimate for $\frac{\pi}{4}$.
- Calculate the standard error of the mean. You can consult `http://en.wikipedia.org/wiki/Standard_error_(statistics)` for the formula.
- Calculate the difference of the Monte Carlo estimate for π with the actual value of π. Hint: one easy way to retrieve π in order to check your results is $\pi = 4\arctan(1)$. Compare this difference with the standard error of the mean. What do you observe concerning the two random number generators?

Expression templates and algorithmic derivation

The purpose of this exercise is to understand and complete the attached code for expression templates and algorithmic derivation.

Expression templates were explained in week 11 of the lecture. The key insight is that the tree associated with any expression can be represented with types at compile-time; the compiler can then perform powerful optimizations on the expression or, under specific circumstances, evaluate the whole expression or parts of it at compile-time.

Here's an example of such a tree:

```
3*(x+1)+4


     +
    / \
   *   4
  /  \
 3    +
     / \
    x   1
```

Note that it consists of vertices representing some operation and leafs representing either a constant or the independent variable.

Algorithmic derivation relies on the observation that for a given tree representing an expression, one can easily calculate the derivative of the expression using the chain rule and the product rule.

We want to combine this into a very simple code that is able to calculate expressions and derivatives thereof, where the only allowed operations are addition and multiplication and we allow only one independent variable, which may however occur several times. The basic ingredient is the class `Expression` that represents vertices of the tree, which has three template parameters:

- The type of the left-hand vertex,
- the type of the right-hand vertex,
- the operation (from an `enum`).

Additionally, we have two classes for the leafs, which can be either a constant or a placeholder for the independent variable.

Each class must implement two functions:

- `operator()`, which calculates the value of the part of the expression that follows below that vertex,
- `derivative`, which calculates the corresponding derivative, obeying chain and product rule.

All this is being represented by types, i.e. the type of the above expression would be:

```
Expression<
    Expression<
        Constant<int>,
        Expression<
            Placeholder,
            Constant<int>,
            Plus
        >,
        Mul
    >,
    Constant<int>,
    Add
>
```

Try to understand the code on the website and fill in the two missing lines in the `derivative` function of the `Expression` such that it obeys chain and product rule. Two `.cpp` files are given on the website: one for compiling into a full program, the other just for assembly. For illustration purposes, the assembler output for the example is also in the archive. You can see that with a smart compiler and the right options, everything is evaluated at compile-time; the assembly therefore contains the final numbers (974 and 157)