Optimizing random walks

Victor Eijkhout, Susan Lindsey

Fall 2021
last formatted: March 1, 2022
1. Malaria

- A mosquito flies in a straight line for some unit time,
- then it turns in a random direction.
- How far does it get in \( N \) time intervals?
- Answer: about \( \sqrt{N} \).
2.

Code:

```c
float avg_dist{0.f};
for ( int x=0; x<experiments; x++ ) {
    Mosquito m(dim);
    for (int step=0; step<steps; step++)
        m.step();
    avg_dist += m.distance();
}
avg_dist /= experiments;
```

Output

[rand] vec:

\[
\begin{align*}
D=3 & \text{ after } 10000 \\
& \text{ steps, distance=} 83.7997 \\
D=3 & \text{ after } 100000 \\
& \text{ steps, distance=} 224.372 \\
D=3 & \text{ after } 1000000 \\
& \text{ steps, distance=} 922.599 \\
\end{align*}
\]

product took: 2776 milliseconds
class Mosquito {
private:
    vector<float> pos;
public:
    Mosquito( int d )
        : pos( vector<float>(d,0.f) ) { };
}
void step() {
    int d = pos.size();
    auto incr = random_step(d);
    for (int id=0; id<d; id++)
        pos.at(id) += incr.at(id);
};
5. missing snippet

walkrandcoord
vector<float> random_step(int d) {
    for (;;) {
        auto step = random_coordinate(d);
        if ( auto l=length(step); l<=1.f ) {
            if ( l==0.f ) {
                /*
                * Zero lengths can conceivably happen for d==1
                * but should not for higher d.
                */
                assert(d==1);
            } else {
                normalize(step,l);
                return step;
            }
        }
    }
}
7. exercise

Take the basic code, and make a version based on

```cpp
template<int d>
class Mosquito {
    /* ... */
```

How much does this simplify your code? Do you get any performance improvement?

You can base this off the file `walk_vec.cxx` in the repository.
8.

So we move the creation of the vectors outside of the computational routines. The random coordinates are now written into an array passed as parameter:

```cpp
void random_coordinate( vector<float>& v ) {
    for ( auto& e : v )
        e = random_float();
};
```
Likewise the random step:

```c
void random_step( vector<float>& step ) {
    for (;;) {
        random_coordinate(step);
    }
}
```
This process of passing the arrays in stops at the `step` method, which we want to keep parameterless. So we add an option `cache` to the constructor to store the step vector as well as the position:

```cpp
class Mosquito {
private:
    vector<float> pos;
    vector<float> inc;
    bool cache;
public:
    Mosquito( int d, bool cache=false ) :
        pos( vector<float>(d,0.f) ), cache(cache) {
            if (cache) inc =
                vector<float>(d,0.f);
    };
```

Output

[rand] pass:

\[ D=3 \text{ after } 10000 \step, \text{ distance=} 76.7711 \]
\[ D=3 \text{ after } 100000 \step, \text{ distance=} 257.19 \]
\[ D=3 \text{ after } 1000000 \step, \text{ distance=} 956.122 \]

run took: 2852 milliseconds

\[ D=3 \text{ after } 10000 \step, \text{ distance=} 87.034 \]
\[ D=3 \text{ after } 100000 \step, \text{ distance=} 256.655 \]
\[ D=3 \text{ after } 1000000 \step, \text{ distance=} 912.033 \]
11.

```c
void step() {
    int d = pos.size();
    if (cache) {
        random_step(inc);
        step( inc );
    } else {
        vector<float> incr(d);
        random_step(incr);
        step( incr );
    }
}
```
12. Sum of squares

There is still a problem with the length calculation. Since there is no reduction operator for ‘sum of squares’, we need to create a temporary vector for the squares, (or do we?) so that we can do a plus-reduction on it.
13. Exercise

Explore options for this temporary. Discuss what’s most elegant, and measure performance improvement.

• This temporary can be passed in as a parameter;
• it can be stored in a global variable;
• or we can declare it static.
• With the C++20 standard, you could also use the ranges header.
float length( const vector<float>& step ) {
    vector<float> square;
    int s = step.size();
    if (square.size() != s) square.resize(s);
    for ( int i=0; i<s; i++) square[i] = step[i];
    for_each( square.begin(),square.end(),
              [] (float& x) { x *= x; } );
    auto l = sqrt
             ( accumulate( square.begin(),square.end(),0.f ) );
    return l;
};
template<int d>
float length( const array<float,d>& step ) {
    array<float,d> square = step;
    for_each( square.begin(),square.end(),
               [] (float& x) { x *= x; } );
    auto l = sqrt
        ( std::accumulate( square.begin(),square.end(),0.f ) );
    return l;
};
16. Optimization

While above we have removed all unnecessary allocation, we get an extra performance boost from optimizations from the compiler knowing the length of the array. Thus, instead of a loop of length two, the compiler will probably replace this by two explicit instructions.
Code:

```c
float avg_dist{0.f};
for ( int x=0; x<experiments; x++ ) {
    Mosquito<dim> m;
    for (int step=0; step<steps; step++)
        m.step();
    avg_dist += m.distance();
}
avg_dist /= experiments;
```

Output

[rand] arr:

D=3 after 10000 steps, distance= 76.3221
D=3 after 100000 steps, distance= 247.5
D=3 after 1000000 steps, distance= 959.735
product took: 358 milliseconds