CSC 180 lab 12: Something numerical

Thurs Nov 29 or Mon Dec 3, 2001

We’ve discussed a handful of numerical computing topics. Here is an exercise which touches upon some of them. It is rather long for a lab, but if you get only part of it done it will still be a useful exercise. The items below are roughly in order of value; I suggest starting and seeing how far you get.

The assignment is to write an adaptive quadrature program to calculate the integral from 0 to 1 of the square root function.

To make this program simpler and hopefully lab-sized, you can hard-code in the values 0 and 1, and the function sqrt (use the math library sqrt function).

1. Start by dividing the interval into 20 subintervals. (This will involve 41 function evaluations, in step 2; then in a later step we will perform 40 more function evaluations which constitute 20 splittings of the intervals with the largest error estimate.)

You can represent each interval with a struct something like this:

```
struct interval {
    double left, mid, right;  /* x values */
    double fleft, fmid, fright;  /* y values ("f(left)", etc) */
    double area;  /* area of the whole trapezoid from ’left’ to ’right’ */
    double errest;  /* error estimate */
};
```

You can represent all 40 intervals with an array of these structs, like this:

```
struct interval inter[40];
```

Divide the interval [0,1] into twenty intervals for which you fill up the left, mid, and right values of inter[0] through inter[19] inclusive.

2. Add to your loop to fill in values for fleft, fmid, and fright as well. As mentioned above, for example be sure not to call sqrt(0.1) for each of inter[1].fright and inter[2].fleft—you should do something like inter[1].fright = sqrt(0.1) but then inter[2].fleft = inter[1].fright. We assume that function evaluations are “expensive”. There should be a total of 41 function evaluations performed at this stage, for 0, 0.025, 0.05, ..., 1 (the 0.025 value, for example, is the “mid” of interval 0).

3. The area of the trapezoid is calculated like this:

```
inter[i].area = (inter[i].right - inter[i].left)
                * (inter[i].fleft + inter[i].fright) / 2;
```

Fill in the error estimates for all intervals according to the following formula, using temporary variables for the area of the left subtrapezoid and the area of the right subtrapezoid:

```
leftarea = (inter[i].mid - inter[i].left)
            * (inter[i].fleft + inter[i].fmid) / 2;
rightarea = (inter[i].right - inter[i].mid)
            * (inter[i].fmid + inter[i].fright) / 2;
inter[i].errest =
            ((inter[i].area - (leftarea + rightarea)) / 3;
```

(the “/3” is a part of one of the standard error formulas, which I propose we not worry about justifying right now)

(continued)

You’ll want to write a function to perform the above. Its header will look like this:

```
        void calc(struct interval *p)
```

You can call it by saying something like calc(&inter[i]). Thus inside the function, you will use something like ‘‘p->mid’’ instead of ‘‘inter[i].mid’’.

4. For the next stage, we begin to realize that an array of intervals, while being the easiest data structure to use, makes for an extremely inefficient locating of the interval with the worst error estimate and an even more awful insertion of a new interval when you do a subdivision. But it is indeed much easier and to do better is not the point of this lab right now.

Somewhat like the ‘‘mostcommon’’ calculation in the agehist program (http://www.dgp.toronto.edu/~ajr/180/example/4/agehist1.c write a loop to go through the array and find the interval with the largest absolute error estimate. (You need to take the fabs() of everything; an error of −1 is worse than an error of 0.5, but an error of 1 is worse than an error of −0.5, etc.)

5. Split this interval by moving all later intervals up by one array index, then moving around all the struct fields so as to split this interval. We have already calculated function values for the new left and right of each interval; calculate the two new fmid values by calling sqrt() twice (only). Then call calc on both of the new subintervals. (We won’t worry about saving the old ‘‘area’’ values to form the new leftarea and rightarea values. This is a simple formula, and not involving any further function evaluations.)

(Obviously, this moving up of everything shows that we are using the wrong data structure; it’s very slow. But it’s also easiest, and data structures aren’t the point right now. We haven’t done the more complex data structures we’d ideally use at this point to make the maintenance of the list straightforward.)

6. After 20 intervals have been split, for a total of 40 intervals, calculate the integral of sqrt from 0 to 1 by adding together all the ‘‘area’’ values.

As mentioned in the introduction, it is okay if you don’t finish this lab. It’s rather long. I suggest just starting and seeing how far you get, and then perhaps pursuing your interests in this regard further. I will publish some solutions some time before the exam, which in this case might be of interest even if you only did half the lab. In the case of most of the labs, you should do the whole lab before looking at my solutions. In the case of this lab, you should do a lot of the lab before looking at the solution.