

NUMERICAL SOFTWARE — CSC 2307

Assignment #3

Due 25 April 2008.

For this assignment, you are to modify the TEAPACK routine `daquad.f` to use the global error-control strategy discussed in class rather than the local error-control strategy that `daquad.f` incorporates. You can find `daquad.f`, the routines `dquad.f` and `dgleq.f` which it calls, and a simple driver program, `example.f`, in my directory `~krj/quadrature` on the CDF system. You should not have to modify either `dquad.f` or `dgleq.f`, and you should not need to change the calls to these routines in your modified version of `daquad.f`.

One slightly tricky point is to correctly modify the extrapolation process at the end of the `daquad.f` routine. Be careful with this, since, if you modify it incorrectly, your routine will appear to work, but its results will be less accurate than they should be, particularly at more stringent tolerances.

There is also a program `heap.f` in `~krj/quadrature` on the CDF system that you can use to implement your heap. This was supplied by one of the students in the course a few years ago. You are welcome to use it or write your own heap routines.

The main disadvantage of the global error-control strategy is that it uses a lot of storage. See if you can think of modifications of the straightforward approach that maintain the logic of the scheme but reduce the amount of storage required to implement it. Also look for (and comment on in your report) other ways that you can easily improve the implementation of the global error-control strategy without unduly complicating it.

When you examine the three TEAPACK routines, `daquad.f`, `dquad.f` and `dgleq.f`, you will notice that the comments in the double-precision versions are inadequate. It's typically the case that the single-precision TEAPACK routines are moderately well-documented, but the comments have been stripped from the double-precision versions, presumably to save disk space when this was an expensive resource¹. (Oh the joys of *legacy code*!)

You should not need to change the calling sequence of `daquad.f`, and, if you do not, then you need not add the comments from `aquad.f` to `daquad.f` to explain the parameters. However, explain all the changes to `daquad.f` that you do make. In addition, note that all the TEAPACK routines are written in capital letters. Please use **small letters only** in your modifications of these routines so that I can easily see what you have changed. Note that the default for `g77` is to translate all capital letters to small letters (except in Hollerith strings) before it compiles a program. So variables with the same name written in small or capital letters are identical to `g77`. (You may want to check that this is the case for any other compiler that you use for this assignment. If this is not the case, your programs may not run correctly.)

In addition, you are to test and compare the original version of `daquad.f` and your modified versions of it using both Pat Keast's quadrature test program, discussed in his technical report *The Evaluation of One-Dimensional Quadrature Routines*², and the *Performance Profile* approach of Lyness and Kaganove³, discussed in §6.4 of Miller's book. In both cases, use the two-point Gauss-Legendre rule as the basic integration formula. This is the basic formula employed in both `example.f` and `test.f`. (Your modification of `daquad.f` should work for all the other basic quadrature rules available in `daquad.f` as well, but you need report the tests with the two-point Gauss-Legendre rule only.)

As shown in Miller's book, you can present the results for the Performance Profiles quite clearly and concisely as graphs. Try to present the results from Keast's test program in an equally clear and concise way.

You may find that the results from the two approaches to testing lead to opposite conclusions. Therefore, in discussing whether the local or global error-control strategy performs better, you should comment on the

¹To encourage students to buy the *Teapack Manual* before it went out of print, the comments were also stripped from the single-precision teapack routines in `/u/csc/src/na/teapack` on the CDF machines. Therefore, I have included the commented single-precision versions of each of the three TEAPACK routines in `~krj/quadrature`.

²I'll leave a copy of this report for you in the Engineering and Computer Science Library, which is located on the second floor of the Sanford Fleming Building.

³I'll leave a copy of the Lyness and Kaganove paper for you in the Engineering and Computer Science Library too.

appropriateness of the tests and in particular assess the appropriateness of the quality and efficiency ratings in Keast's program.

The program `~krj/quadrature/test.f` contains Keast's quadrature test program set-up to test `daquad.f` with the two-point Gauss-Legendre rule. It should be very simple to change it to test your modified version of `daquad.f`.

You are also to produce two *Performance Profile* graphs, each similar to Figure 6.16 on page 160 of Miller's book. In both cases, the integrals are over $[0, 1]$. For the first graph, use the integrand

$$f_{\alpha}(x) = \frac{.01/ (.0001 + (x - \alpha)^2)}{\arctan(100(1 - \alpha)) - \arctan(-100\alpha)}$$

which is similar to the example discussed in Miller's book except that it is normalized to integrate to 1 for all $\alpha \in [0, 1]$. Take α to be uniformly distributed on $[0, 1]$. An easy way to do this is to set $\alpha = \text{rand}(0.)$ in your program, where `rand` is the random number generator in the file `rand.f` in my directory `~krj/quadrature`. There is also a little sample program there, `example.rand.f`, which illustrates how to use the `rand` function⁴.

For the second *Performance Profile* graph, use the integrand

$$f_{\alpha}(x) = \frac{1/(1 + \alpha - x)}{\log(1 + 1/\alpha)}$$

which is similar to the example discussed in the paper *Local Versus Global Strategies for Adaptive Quadrature* by Malcolm and Simpson⁵, except that this integrand also is normalized to integrate to 1 for all α . In this case, take $\alpha = \exp(-15 \cdot \text{rand}(0.))$, giving exponentially distributed random numbers on $[e^{-15}, 1]$, many of which are "close" to zero.

In addition, plot the true error against the requested tolerance at the 80% and 95% confidence levels for the local and global methods and the two performance-profile problems discussed above. This will give some indication of the reliability of the routines.

Give me a hardcopy of your modified `daquad.f` routine, the test results you obtained from running it on Keast's test problems on the CDF system, your two *Performance Profile* graphs, your two requested error versus tolerance graphs and a short (about 5 pages) evaluation of `daquad.f` and your modified version of this routine. Discuss not only your test results but also any other aspects of the programs relevant to the assessment of mathematical software. Leave a working copy of your programs and data in a subdirectory called quadrature of your main directory on the CDF system. Be sure that these files and the directory are readable.

⁴I would like to use a better random number generator, but I couldn't find one on the CDF system. Previously, we used the Sun routine `drand`, which I think is much better, but that doesn't seem to be on the CDF system. If you know of a better random number generator that we can use on the CDF system, please let me know.

⁵I'll leave a copy of this paper for you in the Engineering and Computer Science Library too.