Programming Project #1, Computing e^x

CS 330

Due 11:59 pm Wednesday, September 4, 2013

1 Computing `exp(x)`

For this project we will follow the procedure outlined in Computer Exercise 1.4.11 on page 66 (Exercise 2.2.11 on page 72 in the 6th edition) to compute $f(x) = e^x = \texttt{exp(x)}$.

2 Range reduction

$$f(x) = e^x = 2^{\frac{x}{\ln 2}} \tag{1}$$

Letting $z = \frac{x}{\ln 2}$ we can split z into the sum $z = m + w$ where m is the closet integer to z and w is the left over fraction:

$$\begin{aligned} z &= \frac{x}{\ln 2} && (2)\\ m &= \mathrm{round}(z) && (3)\\ w &= z - m. && (4) \end{aligned}$$

So Equation 1 becomes

$$f(x) = e^x = 2^{m+w} = 2^m 2^w = 2^m e^{w \ln 2} \tag{5}$$

Let

$$u = w \ln 2 \tag{6}$$

and we have reduced the problem of computing e^x to computing

$$f(x) = e^x = 2^m e^u. \tag{7}$$

Since we rounded z to the nearest integer, we know that $|w| \leq \frac{1}{2}$. Therefore, we focus on evaluating e^u for the narrow range where

$$-\frac{\ln 2}{2} \leq u \leq \frac{\ln 2}{2}. \tag{8}$$

The Taylor Series for e^u expanded at $u = 0$ is the classical formula

$$f(u) = 1 + u + \frac{u^2}{2!} + \frac{u^3}{3!} + \cdots + \frac{u^n}{n!} + E_{n+1} \tag{9}$$

where

$$E_{n+1} = \frac{e^{\xi}}{(n+1)!} u^{n+1} \tag{10}$$

for some value ξ between 0 and u. We can get an upper bound on the error term by choosing the largest possible values for ξ and u on the interval (specifically $\xi = u = \frac{\ln 2}{2}$) :

$$|E_{n+1}| \leq \frac{e^{\frac{\ln 2}{2}}}{(n+1)!} \left(\frac{\ln 2}{2}\right)^{n+1} = \frac{\sqrt{2}}{(n+1)!} \left(\frac{\ln 2}{2}\right)^{n+1}. \tag{11}$$

The *relative error* is then

$$\text{rel. error} = \frac{|E_{n+1}|}{|e^u|} \leq \frac{|E_{n+1}|}{|e^{-\frac{\ln 2}{2}}|} = \sqrt{2}\,|E_{n+1}| = \frac{2}{(n+1)!}\left(\frac{\ln 2}{2}\right)^{n+1}. \tag{12}$$

3 What to do

1. First write a program that prints n and the corresponding upper bound for the relative error using Equation 12 for $n = 1$ to 15. Use double precision numbers (`double`'s) for these calculations. From this table determine the smallest n such that the relative error is guaranteed to be below $\epsilon = 1.19209 \times 10^{-7}$ (machine-ϵ for `float`'s).

2. Write a program that contains the function

   ```
   float myexp(float x) { /* your code here */ }
   ```

   which uses Equation 7 to compute e^x. This splits the problem into two pieces:

   (a) Compute 2^m for integer value m which can be efficiently computed via shifting 1 to the left by m : (`1<<m`). For negative m compute 2^{-m} and use the reciprocal.

   (b) Compute e^u using the series in Equation 9 (use Horner's Rule for polynomial evaluation). Use the minimal n found earlier.

   Perform your calculations in single precision. All constants should be determined at compile time (I found the preprocessor constants `M_LOG2E` and `M_LN2` useful which represent $\log_2 e = \frac{1}{\ln 2}$ and $\ln 2$ respectively).

3. Test your `myexp()` function by comparing its results with the math library's `exp()` function. Try 30 values on the interval $-20.0 \leq x \leq 20.0$ and compute the relative error

   $$\text{rel. error} = \left|\frac{\texttt{myexp}(x) - \texttt{exp}(x)}{\texttt{exp}(x)}\right|. \tag{13}$$

   For each `x`, print `x, myexp(x), expl(x),` and the relative error using scientific notation:

   ```
   printf("%+0.9Le %0.9e %0.9Le %0.15Le\n", x, y1, y2, rerr);
   ```

   ```
   -2.000000000e+01 2.061153470e-09 2.061153622e-09 7.389813709130085e-08
   -1.862068966e+01 8.187241107e-09 8.187234572e-09 7.982045327504172e-07
   -1.724137931e+01 3.252102587e-08 3.252101600e-08 3.034678532047436e-07
   -1.586206897e+01 1.291786873e-07 1.291787199e-07 2.526464084845737e-07
   ...
   ```

   Note that I use `long double`'s for "ground truth" values.

4 What to submit

Create an archive (`exp.zip` or `exp.tar.gz`) containing a `README` text file and your C source code. The `README` file should contain the following information:

1. Your name, SID, and email address.

2. List of files in submitted archive; e.g.:

- `README`
- `relerror.c`
- `exp.c`

3. A brief description of your experiment describing the results. For example, describe how you obtained the minimal n for your Taylor series. Some of the relative errors in your last experiment may exceed ϵ – why is this?

Each C source file should compile under `gcc` with very pedantic error checking; For example the following, which assumes your code conforms to the ANSI C99 standard, should yield a clean build with no warnings or errors:

```
gcc -g -Wall -Wextra -Wpedantic -std=c99 exp.c -o exp
```

For speed tests, we could compile with full optimization:

```
gcc -O3 -std=c99 exp.c -o exp
```