Computing e^x

CS 330

1 Computing `exp(x)`

For this project we will compute $f(x) = e^x = \texttt{exp}(\texttt{x})$ using Taylor series. A number of tracks allow us to reduce the range of x, so we can get accurate results even far from the base point of the Taylor series.

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:

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

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

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

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

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

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 Program Template

A zip file containing a skeleton of this project is available in Blackboard, as part of the assignment. It contains a pair of skeleton C files, along a Makefile, and a Perl script that can help test your program. You should probably get a copy of the zip file and unpack it somewhere before continuing.

4 What to do

1. First write a program `rerr.c` that prints n and the corresponding upper bound for the relative error using Equation 12 for $n = 1$ to 15. Use double precision numbers (`doubles`) for these calculations[1]. 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 `floats`). Document your result in a comment at the beginning of `myexp.c`.

2. Write a program `myexp.c` 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) 2^m, for integer value m, can be efficiently computed via `ldexpf(1,m)`.

   (b) Compute e^u using the series in Equation 9 (use Horner's Rule for polynomial evaluation). Use the minimal n found earlier. You may also use the *fused multiply and add* operation `fmaf()`

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

   The program should read exactly one number from the command line and output your approximation to e^x to `stdout` in scientific notation with 9 digits of precision. This is already handled in the provided skeleton.

3. The provided Perl script `test.pl` will exercise you program with a battery of values:

   ```
   $ ./test.pl | more
   exp(-1.0000000e+01)=4.53999298e-05 : 4.53998728e-05 (rerror=1.25556328e-06)  : FAIL!
   exp(-9.7979798e+00)=5.55637361e-05 : 5.55637635e-05 (rerror=4.92288828e-07)
   exp(-9.5959596e+00)=6.80029415e-05 : 6.80029188e-05 (rerror=3.33485031e-07)
   exp(-9.3939394e+00)=8.32269459e-05 : 8.32268852e-05 (rerror=7.29212981e-07)
   exp(-9.1919192e+00)=1.01859190e-04 : 1.01859194e-04 (rerror=4.05950545e-08)
   ...
   ```

   This script computes the relative error using Perl's `exp` as ground truth; In each case, if the *relative error* exceeds ϵ then `FAIL!` is printed. You program should successfully pass all of the tests.

[1] It is acceptable to use the `pow` function for `rerr.c` since this does not have to be efficient. Do <u>not</u> use `pow` for `myexp` function.

5 What to submit

You will submit your project via Blackboard. Turn in a single archive file (zip or compressed tarball) containing all of your files (`myexp.c`, `rerr.c`, `Makefile`) in the same format as the skeleton zip file. Your `myexp()` function must be contained in `myexp.c`. You may, but are not required to, add additional files. If you do so, adjust the Makefile so it will successful generate binaries named `rerr` and `myexp`.

Include a comment at the top of `myexp.c` which:

1. Gives your name
2. Identifies the project
3. Describes the results of your experiment from 4.1, above.