Error Analysis, Clenshaw Algorithm

September 24, 2018

1 Reading Portion

Chapter 5 on function evaluation of Numerical Recipes.

2 The Quadratic Equation

We will explore how accurate the solution of the quadratic equation is, as we move through the double root point. Consider the equation

$$y^2 + ay + b = 0$$

We change variables by defining $z = y/\sqrt{b}$ to get

$$z^2 b + a\sqrt{b}z + b = 0$$

i.e.,

$$z^2 + \frac{a}{\sqrt{b}}z + 1 = 0$$

Let $2\alpha = a/\sqrt{b}$. So this equation is a single parameter equation in z. Its solution is obtained as

$$z = -\alpha \pm \sqrt{\alpha^2 - 1} \tag{1}$$

One of these solutions involves near cancellation when $\alpha \gg 1$. For that case we normalize the numerator and move the term to the denominator:

$$z = \left(-\alpha \pm \sqrt{\alpha^2 - 1}\right) \frac{-\alpha \mp \sqrt{a^2 - 1}}{-\alpha \mp \sqrt{\alpha^2 - 1}} = \frac{1}{-\alpha \mp \sqrt{\alpha^2 - 1}}$$

Cancellation occurs when the terms are of opposite sign. So we rewrite Eq. 1 as

$$\begin{aligned} p &= -\left(\alpha + (\operatorname{sgn}\alpha)\sqrt{\alpha^2 - 1}\right) \\ z_1 &= p \\ z_2 &= 1/p \end{aligned} \tag{2}$$

Note that the standard formula is still to be used when $|\alpha| < 1$.

The codes here are to be done in C so that you can see the effect of using lower precision. Python does things to high precision so we are not often aware of the effect of a poor algorithm on accuracy. Even in C, note that internal calculations are done in 80 bit accuracy. So when compiling, use

```
gcc -ffloat-store -fexcess-precision=standard -g -O0 qroot.c -lm
```

What this does is to ensure that the optimizer does not do the calculations in double precision and only storing the results back in a float array.

-ffloat-store says write back results to variables instead of keeping them in the register

-fexcess-precision=standard says use the corresponding precision intermediate values (float for float variables and double for double variables)

-g asks for debugging symbols to be included

-O0 turns off debugging, so that if you assign a value to a variable it actually gets assigned and does not get "optimized away"

-lm invokes the math library. This is not linked by default.

If you used C++, use instead

```
g++ -ffloat-store -g -O0 qroot.cpp
```

Similarly, if using gfortran, use

```
gfortran -ffloat-store -g -O0 qroot.f90
```

1. Write a C program to compute the root using this formula and its complex form for a range of α values (given α_{start}, α_{end} and N). Note that the range of values will be over many orders of magnitude. So the samples should be spaced geometrically.

2. Also calculate the roots using "float" precision. For this, you have to have variables declared as float.

3. Also calculate the roots using "float" precision with the more accurate formula.

4. Determine the error between the expressions in parts 2 and 3 (taking part 1 as the exact answer) and plot the magnitude of error vs α in a log log plot. Discuss.

3 Stable and unstable series

Chebyshev satisfies the following recursion relation:

$$T_{n+1} = 2xT_n - T_{n-1},\ T_0 = 1, T_1 = x$$

while the cosine function satisfies, with $p_n(x) = \cos nx$

$$p_{n+1} = (2\cos x)\, p_n - p_{n-1},\ p_0 = 1, p_1 = \cos x$$

Additional recursions of interest include

$$\text{sine series: } p_{n+1} = (2\cos x)\, p_n - p_{n-2}$$

$$\text{Bessel Series: } Z_{n+1}(x) = \frac{2n}{x} Z_n(x) - Z_{n-1}(x)$$

where $Z(x)$ could be $J_n(x)$ or $Y_n(x)$ or any linear combination of the two.

We want to compute the series

$$S(x) = \sum_{n=0}^{40} \frac{1}{n+1} J_n(x)$$

where J_n corresponds to the decaying root for $n \gg x$.

- Write a Python function that uses the built in routines to compute the sum. This is our exact solution.
- Code the series with a forward series computing the function using the recursion as you update the sum. Obtain the error and plot the error vs n for $x = 1.5$ and for $x = 15$. Explain the difference.
- Code the series with a backward series assuming that $J_{60}(x) = 1$ and $J_{61}(x) = 0$. Normalize the value of $J_0(x)$ to 1. Determine the error in this code for $x = 1.5$ and for $x = 15$.

Clenshaw Algorithm

- Implement Clenshaw algorithm in Python for arbitrary coefficients in the recursion.
- Implement the Chebyshev sum for e^x between -1 and 1 using both Clenshaw and using the direct method. Compare the errors.
- Repeat for Fourier.

The claim is that Clenshaw is about 30% better in most cases. How will you check this claim? Obtain the mean and standard deviation of the errors and use it to validate.