The Newton Method as a Short WEB Example

August 31, 1996
15:22

Abstract: This is written in John Krommes' FWEB but with the output higher level language being **C**. FWEB is the most supported of the dialects of literate programming.

This is an expository note about the WEB style of Literate Programming. We do not purport this to be the best way of coding a solution of this problem. Some constructs are used to illustrate concepts of literate programming.

Table of Contents

Contents

1. The Newton Method

The Newton method is based on the Taylor's Series:

$$f(x) = \sum_{n=0}^{+\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots$$

In order to use the Newton method a number of conditions must be met: $f(x)$ must be analytic and the second derivative of $f(x)$ must be monotone in the neighborhood of x. If these conditions are met, then the Newton method will converge quadratically and monotonically (*i.e.* with each iteration the number of significant digits will double).

2. This program is written to run on the CRAY, standard *unix* systems, and standard *DOS* systems producing similar results. We will normally use the `gcc` compiler on SUN computers and `xlc` on RS-6000's.

3. Precision dependence is controlled by the environment variable *CRAY*. It should be straightforward to accomodate other environments, such as VMS.

The `tangle` command line to produce *CRAY* code is:

```
tangle Newton -m"CRAY"
```

The most significant difference we are concerned with is the use of 64-bit arithmetic for the floating point numbers. We wish to use the same precision on both systems. We realize that there is a slight difference in the two but it is not as significant as those between IEEE floating point and IBM 360's.

4. Charles Karney of Princeton University furnished macros for machine dependencies as part of a large body of code released with John Krommes' `FWEB` system. The default is the IEEE floating point system that is typical of *unix* and *DOS* systems.

```
"NewtonC.c" 4 ≡
@#if defined (CRAY)
  @m CRAY 1
@#else
  @m CRAY 0
@#endif
```

5. We introduce another macro to make it easy to run the tests with 64-bit precision in all the environments. The Cray uses 64-bit precision for *real* variables by default. The other machines will use this when *real*∗8 is specified. However, the actual storage will may vary among machines.

Another macro is used to convert floating point constants by appending the "L" exponent on 32-bit machines.

"NewtonC.c" 5 ≡
  @f *floating float*
@#if *CRAY*
  @m *floating* **float**
  @m *const*(*x*) *x*
@#else
  @m *floating* **double**
  @m *const*(*x*) *x*##*L*
@#endif

6. A Newton method program

This program will use the Newton method to solve the equation $\cos(x) = 0$. In **C** we have to **#include** some files to get out standard input-output and also the mathematics functions. Later we must remember to include the `-lm` switch for loading the library.

"NewtonC.c" 6 ≡

```
#include <stdio.h>
#include <math.h>
  main( ) {
  ⟨ Declarations 7 ⟩
  ⟨ Initialize Variables 9 ⟩
  ⟨ Iterate on the answer 10 ⟩
  ⟨ Print Results 17 ⟩
  }
```

7. We declare the following variables: *x_0* will represent x_k from the Taylor Series, x will correspond to x_{k+1}, and *delta_x* will be calculated as the difference between x_{k+1} and x_k. These will be defined as **floating** which will be translated to the *real* type in the **C** source code for the machine specified in the `FTANGLE` command line.

⟨ Declarations 7 ⟩ ≡
  floating *x_0*, *x*, *delta_x*;

See also sections 8 and 13.

This code is used in section 6.

8. We will also declare two **int** variables. These are an iteration index, k, and a *limit* for the number of iterations.

⟨ Declarations 7 ⟩ +≡
    int k, *limit*;

9. The initial value of x is not quite arbitrary. This problem has an ∞ of solutions and the initial value can cause convergence to roots that are not desired. In the case of $x = 1.2$, x will converge to $\frac{\pi}{2}$. We will also need to set *delta_x* to any nonzero value. This is needed to allow the iteration loop to execute at least once.

Of course, in most reasonable programs, this would not be hardcoded, there would be a dialog to establish these values. The *limit* of 10 iterations and *delta_x_max*= 0.5 are based upon knowing the problem and algorithm's performance.

⟨ Initialize Variables 9 ⟩ ≡
    x_0 = **const** (1.2);
    delta_x = **const** (0.000001);
    limit = 10; /∗ more than safe ∗/

See also section 14.

This code is used in section 6.

10. The following is the heart of the Newton method. We will continue calculating x_{k+1} until $|\Delta x|$ becomes sufficiently small. With each iteration of the loop, the number of significant digits in x will double, approximately.

⟨ Iterate on the answer 10 ⟩ ≡
    for ($k = 1$; *delta_x* > **const** (0.0) ∧ $k \leq$ *limit*; k++) {
    ⟨ Calculate the Newton change 11 ⟩
    ⟨ Apply convergence strategies? 15 ⟩
    ⟨ Make the step 12 ⟩
    }

This code is used in section 6.

11. In the vanilla Newton method, *x_0* will receive the value of x from the previous iteration. To calculate x, we use the Newton formula described in Module 1 by substituting *x_0* for x_k. Now, we will calculate *delta_x* to determine if the iteration loop should be terminated. The Newton method is an iterative method using the first two terms of the aforementioned Taylor Series:

$$f(x_{k+1}) = f(x_k) + f'(x_k)(x_k - x_{k+1})$$

Since $f(x_{k+1}) = 0$, the formula is then transformed into the Newton Method:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

⟨ Calculate the Newton change 11 ⟩ ≡
```
delta_x = -(cos(x_0))/(-sin(x_0));
printf("The␣Newton␣step␣is␣%g\n", delta_x);
```

This code is used in section 10.

12. The step may have been adjusted. Regardless, we add the step and update the variables.

⟨ Make the step 12 ⟩ ≡
```
printf("The␣step␣is␣%g\n", delta_x);
x = delta_x + x_0;
x_0 = x;
delta_x_previous = delta_x;
```

This code is used in section 10.

13. Convergence Strategies

It is well known that if certain conditions are met, then the convergence of the Newton method is quadratic and monotone. This applies within the *radius of convergence*. In some cases, we can apply strategies that will expand the radius of convergence and still allow rapid convergence in the immediate proximity of the solution. The strategies will include constraining the step or change and noticing when convergence is not monotone.

⟨ Declarations 7 ⟩ +≡
```
floating delta_x_max, delta_x_previous;
```

14. These variables need initial values.

⟨ Initialize Variables 9 ⟩ +≡
```
delta_x_previous = const (0.0);
delta_x_max = const (0.5);
```

15. When we solve practical problems, we will have some estimate of the answer, or at least its order of magnitude. After calculating *delta_x*, we do not let its magnitude exceed *delta_x_max*. We don't check for quadratic convergence.

```
⟨ Apply convergence strategies? 15 ⟩ ≡
    if (delta_x > delta_x_max)
      delta_x = delta_x_max;
    else if (delta_x < (−delta_x_max))
      delta_x = −delta_x_max;
```

See also section 16.

This code is used in section 10.

16. This problem is an example of those problems that do not exhibit monotone convergence. In this case, the assumptions are not met because there is an inflection point at the solution. When we do not have monotone convergence, we should note it. We find this by comparing the signs of consecutive changes.

We could have included some quite complicated strategies, but we are just illustrating WEB.

```
⟨ Apply convergence strategies? 15 ⟩ +≡
    if ((delta_x∗delta_x_previous) < const (0.0))
      printf ("Oscillating␣%d\n", k);
```

17. Finally, it's time to let the world know the results we have calculated.

```
⟨ Print Results 17 ⟩ ≡
    printf ("The␣solution␣to␣cos(x)=0␣is␣%g\n", x);
```

This code is used in section 6.

18. The next page is intentionally blank.

19. Index

⟨ Apply convergence strategies? 15, 16 ⟩ Used in section 10.
⟨ Calculate the Newton change 11 ⟩ Used in section 10.
⟨ Declarations 7, 8, 13 ⟩ Used in section 6.
⟨ Initialize Variables 9, 14 ⟩ Used in section 6.
⟨ Iterate on the answer 10 ⟩ Used in section 6.
⟨ Make the step 12 ⟩ Used in section 10.
⟨ Print Results 17 ⟩ Used in section 6.

COMMAND LINE: `"fweave NewtonC"`.
WEB FILE: `"NewtonC.web"`.
CHANGE FILE: `(none)`.
GLOBAL LANGUAGE: C.