Homework 2

Mayer Goldberg

April 12, 2018

Contents

1 Overview

In this assignment, you shall write a program in x86 assembly language, to find a single root of a polynomial of *complex* coefficients, i.e., find a point z where:

$$c_0 + c_1 \cdot z + c_2 \cdot z^2 + \cdots + c_n \cdot z^n \quad = \quad 0$$

This assignment shall require you to work with the x87 subsystem, the SSE register set, arrays, and interfacing with the C standard library. The x87 does not support *complex numbers*, so you will need to implement the representation for complex numbers and the functions defined over them using support for floating-point numbers in the x87 subsystem.

2 Computing roots

You shall implement the *Newton-Raphson* approximation algorithm over complex numbers: Starting with an approximation z for the root of the function f, the algorithm constructs the sequence $z_0 = z, z_1, z_2, \ldots$ defined inductively, as follows:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

where f' is the *derivative* of f. The sequence $\{f(z_n)\}_{n=0}^{\infty} \to 0$, so $\{z_n\}_{n=0}^{\infty}$ converges to a root of f.

3 The format of the input

3.1 The input format for complex numbers

Complex numbers can be represented in *rectangular coördinates* using two floating point numbers. The input format shall be two floating-point numbers separated by a single space. For example:

- `3.4 4.5` shall be taken to represent the number $2.3 + 4.5i$
- `0.0 1.0` shall be taken to represent i
- `3.0 0.0` shall be taken to represent 3
- `0.0 0.0` shall be taken to represent 0

3.2 The input to the program

The program shall read from *stdin* the following items:

- A specification of *tolerance* (how close must we get to an actual zero of the function)
- A specification of the *order* (highest power in the polynomial)
- The coeffcients (since they are indexed, these may be given in any order)
- An initial value z with which to start the computation

Below you shall find a sample input:
Sample input:

```
epsilon = 1.0e-8
order = 2
coeff 2 = 2.0 0.0
coeff 1 = 5.0 0.0
coeff 0 = 3.0 0.0
initial = 1.0 -1.0
```

The blanks around the equal (`=`) sign are intentional and mandatory.

3.3 The output we expect

The output will be a complex number, and have the following format:

```
root = -1.0000000000081413 2.2064722707984763e-11
```

Note that this number is an approximation for the real root -1.

Don't worry if your output differs by a few digits; What we check is whether the numbers are within the specified tolerance level (`epsilon`). The test we employ to determine whether your result is acceptable is as follows: If $\Re, \Im : \mathbb{C} \to \mathbb{R}$ return the *real* and *imaginary* components of a complex number, and z_{returned} is the value returned by your program, then your program found a root correctly if

$$\begin{aligned} ||f(z_{\text{returned}})|| &< \epsilon \\ \text{where } ||z|| &= \sqrt{\Re^2(z) + \Im^2(z)} \end{aligned}$$

4 How you can test your own program

Start with a polynomial the root of which you know. This is easy to do: Pick n complex numbers $z_1, \ldots, z_n \in \mathbb{C}$, and expand the polynomial $(z - z_1) \cdots (z - z_n)$ to obtain an n-th degree polynomial with $n + 1$ complex coefficients. Prepare the input to your program as specified above, giving values for the *tolerance*, *order*, the $n + 1$ coefficients, and some initial value. If your roots are complex, you might as well pick a complex initial value, because if you start with a real initial value, you will not find the roots! Then check what your program returned: If it is sufficiently close to one of the roots, in the sense specified above, then your program passed this one test.

Test your program with polynomials that have n complex roots, n real roots, and mixed, and try different initial values.

5 What to study in the x87 subsystem

You want to familiarize yourself with the following list of x87 instructions: `fabs`, `fadd`, `faddp`, `fcom`, `fcomp`, `fcompp`, `fdiv`, `fdivp`, `fdivr`, `fdivrp`, `finit`, `fld`, `fmul`, `fsqrt`, `fst`, `fsub`, `fsubp`, `fsubr`, `fsubrp`, `ftst`, `fxch`. You may wish to use additional x87 instructions, but this list should cover [nearly] anything you really need. Consult the Intel manuals available on the course website, as well as any online tutorials that you may find on the x87. **You may not use code that you find on the internet, and suspected cheaters shall be sent before the disciplinary committee!**

You will also need to use the SSE register sets (registers `xmm0`, `xmm1`, etc) as per the example I posted on the course website (`64bit-fp01.x86-64`), in order to communicate with the `scanf` and `printf` procedures from the C standard library. Remember that the register `rax` must contain the number of arguments passed to such a call. For example, if you call `printf` with 2 `float` or `double` arguments, then the value of `rax` must be 2.