The Durand–Kerner Method

CS 330

Due 11:59:59 pm, Friday, September 27, 2013

1 Introduction

For this project you will implement the *Durand-Kerner* method for finding all n (complex) roots of a nth degree polynomial:

$$f(z) = z^n + c_{n-1}z^{n-1} + \cdots + c_2 z^2 + c_1 z + c_0 \tag{1}$$

All coefficients c_j are complex and $f(z)$ is normalized so that $c_n = 1$. This is an iterative method that finds all the roots simultaneously, and is (briefly) explained in Section 2; you can read more about it at

`http://en.wikipedia.org/wiki/DurandKerner_method`

Section 3 describes how to use complex numbers in C99. The details of how your program should work and what you need to submit is given in Section 4.

2 The Method

2.1 Iteration

Starting with some initial (distinct) guesses $\{z_0^{(0)}, z_1^{(0)}, \ldots, z_{n-1}^{(0)}\}$ for each of the n roots, we iteratively refine these guesses using the update formula:

$$z_j^{(k+1)} = z_j^{(k)} - \frac{f(z_j^{(k)})}{Q_j^{(k)}}, \quad j = 0, \ldots, n-1, \tag{2}$$

where

$$Q_j^{(k)} = \prod_{\substack{i=0 \\ i \neq j}}^{n-1} \left(z_j^{(k)} - z_i^{(k)}\right). \tag{3}$$

Note that $Q_j^{(k)} \neq 0$ as long as the $z_j^{(k)}$'s are distinct; this update formula tends to maintain this separation which explains why convergence can be slow for multiple roots. We repeatedly update each z_j until $k \geq$ max iterations or we have convergence:

$$\max_j \left| z_j^{(k+1)} - z_j^{(k)} \right| \leq \epsilon. \tag{4}$$

This method is numerically stable (at least for distinct roots) and converges on the solution rather quickly.

2.2 Initial Values

We first find the radius R of a circle in the complex plane that contains the roots of f :

$$R = 1 + \max_j |c_j|. \tag{5}$$

Then the initial guesses for the n roots are evenly placed around this circle:

$$z_j^{(0)} = (\cos\theta_j + i \sin\theta_j) \cdot R, \quad j = 0, \ldots, n-1, \tag{6}$$

where

$$\theta_j = j\frac{2\pi}{n}. \tag{7}$$

2.3 The Algorithm

The algorithm is outlined in Figure 1. For each iteration we store the change in each z_j in Δz_j (line 6) which is used to update z_j (line 10) only after all the Δz_j's have been computed. We track the maximum $|\Delta z_j|$ (lines 7 and 8) which determines when to quit (line 11). Make sure to use *Horner's Rule* when evaluating $f(z_j)$.

3 Complex Arithmetic in C99

Complex numbers are a primitive data type in C99; include the following header file to access all the goodies:

```
#include <complex.h>
```

and use the `-std=c99` switch for `gcc`. Use the data type `double complex.` C99's arithmetic operations are overloaded for both the `float complex` and `double complex` data types. The preprocessor constant `I` is used for $i = \sqrt{-1}$. For example, the following declares the complex variable `z` and initializes it with the value $3.4 + 10i$:

```
1  Compute initial values $\{z_0, \ldots, z_{n-1}\}$ using Equation 6.
2  **for** $k = 1 \ldots k_{\max}$
3     Let $\Delta z_{\max} = 0$.
4     **for** $j = 0 \ldots n-1$
5        Compute the product $Q_j = \prod_{\substack{i=0 \\ i \neq j}}^{n-1} (z_j - z_i)$ (Equation 3).
6        Set $\Delta z_j = -f(z_j)/Q_j$.
7        **if** $|\Delta z_j| > \Delta z_{\max}$
8           Set $\Delta z_{\max} = |\Delta z_j|$.
9     **for** $j = 0 \ldots n-1$
10       Update $z_j = z_j + \Delta z_j$.
11    **if** $\Delta z_{\max} \leq \epsilon$ quit.
```

Figure 1: The Durand–Kerner iterative algorithm that converges on the n roots of the polynomial $f(z)$. The parameter $k_{\max}$ caps the number of iterations; ϵ determines when we are "close enough" to the solution.

```
double complex z = 3.4 + 10.0*I;
```

There is a large repertoire of functions provided, but the routines needed for this project are the following:

```
double cabs(double complex z);   // absolute value |z|
double creal(double complex z);  // real part
double cimag(double complex z);  // imaginary part
```

4 What to Submit

4.1 Input/Output

To illustrate how your program should work, we follow the wiki example that determines the roots for the cubic function

$$f(z) = z^3 - 3z^2 + 3z - 5. \tag{8}$$

which has three roots 2.5874, $0.2063 \pm 1.3747i$. Your program should read the coefficients c_0, c_1, ..., c_{n-1} from `stdin` as ASCII floating point pairs; in our example, the input would be

```
-5 0
3 0
-3 0
```

Remember that $c_n = 1$ and is not specified. The values of each z_j before each iteration should be output to ten decimal places as follows:

```
$ ./dk < example.in
iter 1
z[0] = 6.0000000000 + 0.0000000000 i
z[1] = -3.0000000000 + 5.1961524227 i
z[2] = -3.0000000000 + -5.1961524227 i
<snip>
iter 7
z[0] = 2.5874135554 + -0.0000000000 i
z[1] = 0.2062932223 + 1.3747410626 i
z[2] = 0.2062932223 + -1.3747410626 i
iter 8
z[0] = 2.5874010521 + -0.0000000000 i
z[1] = 0.2062994740 + 1.3747296371 i
z[2] = 0.2062994740 + -1.3747296371 i
```

4.2 Other Implementation Details

Since we will experiment with roots that are not much larger than one in magnitude we will use absolute error (which should be close to relative error) for a stopping criteria; target 5 or 6 digits of precision by setting $\epsilon = 10^{-6}$. You can assume that all polynomials will have degree $n \leq 20$ (so you can use fixed sized arrays to hold the coefficients $\{c_j\}$, roots $\{z_j\}$, and $\{\Delta z_j\}$). Use $k_{\max} = 50$ to cap the number of iterations.

4.3 Submission Instructions

Name the source code of your project `dk.c` and make sure it compiles and links with the following command:

```
gcc -g -std=c99 -Wall dk.c -o dk -lm
```

Create a text file named `README` that contains the name of the project, your name and email address, a brief description on what your program does and how to build it from source. Archive your source code, `README`, and other supporting files into either a `.zip` or `.tar.gz` and submit electronically by midnight. I will post some test cases for you to try on the course's web site.