Applied Programming

Finding Roots

of Polynomials

(real and complex)

Roots of Polynomials

- The Bisection, Newton, and Secant algorithms can ***only find real solutions***

- Real polynomials equations may also ***have complex solutions***: *e.g.*, $x^2 - x + 1 = 0$
  - Need a method that can find complex roots
  - Requires complex arithmetic !!

Roots of Polynomials

- To find roots of polynomials specialized algorithms are better.

- The most common is Laguerre's method (that finds both real and complex roots, *i.e.*, may require complex arithmetic)

- We start by looking at the ***numerical evaluation of polynomials***

Roots of Polynomials

1) How many roots ?

A ***real polynomial of degree n*** has ***exactly n roots*** (some may be complex)

Notes:

- A real polynomial is a polynomial with real coefficients
- The **roots** of polynomial are usually called their **zeros**

Example:

- The polynomial $p(x)=x^5+1$ has exactly 5 roots (some are complex) *Can you find all of them ?*

Roots of Polynomials

General approach:

1. ***Find a root*** (usually the smallest)
2. ***Remove it*** from the polynomial
3. ***Repeat*** (e.g., goto 1) until all the roots have been found.

The process of removing a root from a polynomial (step 2) is called "deflation"

Polynomial Deflation

Let x^* be a root of $p(x)$ (e.g., $p(x^*)=0$). Then

$$p(x) = (x-x^*)q(x)$$

One degree less than $p(x)$

where

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
$$q(x) = b_{n-1} x^{n-1} + b_{n-2} x^{n-2} + \cdots + b_1 x + b_0$$

The objective is to obtain the coefficients of $q(x)$ directly from the coefficients of $p(x)$ and the root to be deflated x^*. This can be achieved by synthetic division (you did this in high school)

Recall: Synthetic Division

- Deflate the root $x=1$ from the polynomial $\boldsymbol{p(x)=x^4+2x^3+4x^2-2x-5}$

- Using synthetic division: divide $p(x)$ by $(x-1)$

Root to deflate

$$\begin{array}{c|ccccc}
 & 1 & 2 & 4 & -2 & -5 \\
1 & & 1 & 3 & 7 & 5 \\
\hline
 & 1 & 3 & 7 & 5 & \boxed{0}
\end{array}$$

Remainder

- Deflated Polynomial: $\boldsymbol{q(x)=x^3+3x^2+7x+5}$

Deflation Algorithm

- ***Synthetic division*** can be organized efficiently in a ***"deflation algorithm"*** (here *r* is a root of the polynomial, e.g., $p(r)=0$, where,

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

$$q(x) = b_{n-1} x^{n-1} + b_{n-2} x^{n-2} + \cdots + b_1 x + b_0$$

Inputs: r ; a_n , a_{n-1} ,…a_0

Output: b_{n-1} , b_{n-2} ,…,b_0

```
b[n-1] = a[n];
for k = n-2:-1:0
  b[k] = b[k+1]*r + a[k+1];
end
```

Note: High order first e.g. ax^2+bx+c

Deflation: Complex Roots

- Deflation works both for real and complex roots.
- If a polynomial has ***real coefficients*** then its ***complex roots come in complex conjugate pairs***:
  - If you find that $x=a+ib$ is a root of $p(x)$

    then $x=a-ib$ is also a root of $p(x)$

Consequence:

- When solving for roots of real polynomials ***once a complex root is found deflate both the complex root and its conjugate***
- For example, ***if*** you find that ***$x=1+i\,4$ is a root*** then you already know that ***$x=1-i\,4$ is also a root*** (no need to find it)

Example/Class Exercise

- Find all the roots of the following polynomial by sequential deflation (*i.e.*, using synthetic division) until the polynomial becomes a 2nd order polynomial. Then use the quadratic formula to find the remaining roots

$$p(x) = x^4 - 2x^3 - 9x^2 + 2x + 8$$

- *Hint: the constant term is the product of the roots is 8*

Example

$p(x) = x^4 - 2x^3 - 9x^2 + 2x + 8$ given "1" is a root

	1	-2	-9	2	9	
1		1	-1	-10	-8	
	1	-1	-10	-8	0	must be zero

$p(x) = x^3 - x^2 - 10x - 8$ now given "-1" is a root

	1	-1	-10	-8	
-1		-1	2	8	
	1	-2	-8	0	must be zero

$p(x) = x^2 - 2x - 8$

Example

p(x) = x^2 -2x -8 now given "-2" is a root

```
   | 1 -2   -8
-2 |   -2    8
   ------------
     1 -4    0   must be zero
```

p(x) = x -4

- At the end of deflation we are left with a first degree polynomial
  whose root is found trivially to be $x=4$

Laguerre's Algorithm

- Laguerre's method is an **iterative** approach to find roots of an equation.
- Suppose we want to find all the roots of a polynomial of ***degree n*** *of the form:*

$$p(x) = c(x - x_1)(x - x_2)\cdots(x - x_n)$$

- The method proceeds by ***sequentially computing one root at a time*** using a special correction function followed by a deflation step

Laguerre's Algorithm: The Magic

Laguerre defines two helper functions:

$$G(x) = \frac{d}{dx} \ln |p(x)| = \frac{p'(x)}{p(x)}$$

$$H(x) = -\frac{d^2}{dx^2} \ln |p(x)| = \left(\frac{p'(x)}{p(x)}\right)^2 - \frac{p''(x)}{p(x)}$$

And a "correction factor":

$$\alpha^{(i)} = \frac{n}{G(\tilde{x}^{(i)}) \pm \sqrt{(n-1)(n\,H(\tilde{x}^{(i)}) - G(\tilde{x}^{(i)})^2)}}$$

Which are solved iteratively, trying to make the correction factor as small as possible.

Laguerre's Algorithm

$$G(x) = \frac{d}{dx}\ln|p(x)| = \frac{p'(x)}{p(x)}$$
$$= \frac{1}{x-x_1} + \frac{1}{x-x_2} + \cdots + \frac{1}{x-x_n}$$

$$H(x) = -\frac{d^2}{dx^2}\ln|p(x)| = \left(\frac{p'(x)}{p(x)}\right)^2 - \frac{p''(x)}{p(x)}$$
$$= \frac{1}{(x-x_1)^2} + \frac{1}{(x-x_2)^2} + \cdots + \frac{1}{(x-x_n)^2}$$

Note: *G(x)* and *H(x)* *require* $p(x), p'(x), p''(x)$

Derivatives

- Laguerre's Algorithm requires the first and second derivatives, both evaluated at *x*.
  - Could just reused the Newton's derivative code fragment twice!
  - The use a polynomial evaluator at *x*

Note: Assumes low order first.

```
for (i = 1; i < sizeof(f)/sizeof(double); i++)   {  df[i - 1] = i*f[i];    }
for (i = 1; i < sizeof(df)/sizeof(double); i++)  {  ddf[i - 1] = i*df[i];  }
```

e.g. $a+bx+cx^2$

- There is a faster way

Computing p, p' & p''

- p, p' and p'' can all be evaluated simultaneously at a point x using the following:

```
p = a_n,   q = r = 0;
for k = n-1: -1 : 0
   r = rx  + q;
   q = qx  + p;
   p = px + a_k
p′ = q;
p″ = 2r;
```

Note: Assumes low order first.

n is the degree

e.g. $a+bx+cx^2$

$$G(x) = \frac{p'(x)}{p(x)}$$

$$H(x) = G(x)^2 - \frac{p''(x)}{p(x)}$$

The correction factor α

$$\alpha^{(i)} = \frac{n}{G(\tilde{x}^{(i)}) \pm \sqrt{(n-1)(n\,H(\tilde{x}^{(i)}) - G(\tilde{x}^{(i)})^2)}}$$

- The **sign denominator +/- must be chosen** to make the magnitude of the denominator as large as possible
  - If $\tilde{x}^{(i)}$ is ***real*** choose it equal to $\operatorname{sign} G(\tilde{x}^{(i)})$
  - If $\tilde{x}^{(i)}$ is ***complex*** you must evaluate both expressions (with the + and - sign) and choose the ***denominator with largest absolute value***
  - ***n*** is the degree of the equation

Laguerre's Algorithm

Superscript denotes iteration

1. Choose initial guess $\tilde{x}^{(o)}$ and set *i=0*
2. Compute $p(\tilde{x}^{(i)}), p'(\tilde{x}^{(i)}), p''(\tilde{x}^{(i)})$ and then $G(\tilde{x}^{(i)}), \quad H(\tilde{x}^{(i)}),$
3. Compute the ***correction factor***

$$\alpha^{(i)} = \frac{n}{G(\tilde{x}^{(i)}) \pm \sqrt{(n-1)(n\,H(\tilde{x}^{(i)}) - G(\tilde{x}^{(i)})^2)}}$$

   where the **sign (+ or -) must be chosen** to make the magnitude of the denominator as large as possible

4. If $|\alpha^{(i)}| < \mathrm{tolerance}$

   stop

   else

   Update root estimate

$$\tilde{x}^{(i+1)} = \tilde{x}^{(i)} - \alpha^{(i)}, \quad i = i + 1$$

   goto 2.

Note that the stopping criterion is based on the error between two consecutive updates

Laguerre's Convergence

- Laguerre's algorithm has ***guaranteed convergence*** (one root at a time) provided that we use appropriate initial guesses.
  - Always ***converges*** regardless of our initial guess.

- Its order of convergence is ***cubic to single roots*** and ***linear to roots of higher multiplicity***.

Finding all the roots

1. Find one root using ***Laguerre's Algorithm***
2. ***Deflate*** the root,
   - If the root was complex also deflate its complex conjugate (two consecutive deflation steps)
3. If degree of polynomial > 2 goto 1
4. Find the ***remaining roots using formulas***

Warning:

Deflation may introduce ***large round-off errors***. For high accuracy you may need to "refine" the roots (*e.g.*, using Newton's method)

Reminder: Roots 1 & 2

- If ***Deflation*** results in a 2nd order (quadratic) equation, use the quadratic formula to solve for the roots.

1. Standard form: $ax^2 + bx + c = 0$

2. $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

3. If ***Deflation*** results in a 1st order, then you know the root.

Laguerre's Gotchas

- The imaginary part may be “nearly” zero.
  - Never check for exact zero
- cabs – complex absolute value
  - fabs - real floating point
- csqrt – complex square root
  - Squt – real floating point

Reminder: ANSI C99

- **C99** *supports complex numbers* with **`complex.h`**
  - Compile with: -Wall **-std=c99** -pedantic –g
    - "c99_complex.c" sample code included with homework

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
. . .
  int k;
  double complex* a;
  double complex b,c;         /* double precision, use this */
  complex d;                  /* single precision */

  /* dynamic array of complex numbers, no error checking */
  a = (double complex*)malloc(10*sizeof(double complex))

  /* Initialize array */
  for(k = 0;k < 10; k++){ a[k] = (k+1) + (k+1)*I;  }
  b = a[0]*a[9];
  c = a[1]/a[6]+a[3];
  printf("b = %5.2lf %+5.2lfI\n",creal(b),cimag(b));
. . .
```

Complex Numbers: ANSI C99

- Pointers and arrays of complex numbers are used in the same way as other primitive types

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
. . .
  int k;
  double complex b,c;
  /* dynamic array of complex numbers */
  double complex* a = malloc(10*sizeof(double complex))
  if (a == NULL){
    fprintf(stderr,"Memory allocation error");
    exit(EXIT_FAILURE);
  }
  /* Initialize array */
  for(k=0;k<10;k++){
    a[k] = (k+1) + (k+1)*I;
  }
  b = a[0]*a[9];
  c = a[1]/a[6]+a[3];
  printf("b = %5.2lf %+5.2lfI\n",creal(b),cimag(b));
. . .
```

Example: Finding All Roots

- Apply Laguerre's method to find all the roots of the polynomial

$$p(x) = x^5 - 3.39\,x^4 + 5.4239\,x^3 - 4.1672\,x^2 \ldots \\ + 1.4866\,x - 0.1988$$

use a tolerance of $1.15\text{x}10^{-15}$ and a starting guess $x^{(o)}=0$.

Answer:

$$\begin{aligned} x_1 &= 0.443093525519815 \\ x_2 &= 0.473453237240093 + 0.0131872677955576i \\ x_3 &= 0.473453237240093 - 0.0131872677955576i \\ x_4 &= 1.00000000000000 + 1.00000000000000i \\ x_5 &= 1.00000000000000 - 1.00000000000000i \end{aligned}$$

Python code: **`poly.py`**, Matlab/Octave code: **`ex_laguerre`**

Exercise 1

You are given the following polynomial:

$$p(x) = 16x^4 + 70x^3 - 169x^2 - 580x + 75$$

A Laguerre's solver return a root of x= -5.
Deflate the polynomial

```
   | 16   70   -169 -580   75
-5|      -80     50  595 -75
   ---------------------------
     16  -10   -119   15    0    must be zero
```