Factoring a Quadratic from a Degree *N* Polynomial Using Newton's Method

Pablo R. Manalastas
Ateneo de Manila University
Quezon City, Philippines
pmanalastas@gmail.com

ABSTRACT

Given a polynomial $P(x)$ of degree n, namely the polynomial $P(x) = c_0x^n + c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n$, where the coefficients c_j, $j = 0, 1, \cdots n$ are real, we want to numerically find a quadratic factor $K(x) = x^2 + ax + b$ of this polynomial. In order to find zeroes of $P(x)$, we just find a quadratic factor $K(x)$, and find the zeroes of $K(x)$ using the quadratic formula. Repeatedly factoring out a quadratic gives all the zeroes of the polynomial $P(x)$. Starting with initial guesses for the coefficients a and b, say the initial guesses a_0 and b_0, we use long division of $P(x)$ by $K_0(x) = x^2 + a_0x + b_0$, to obtain a quotient $Q(x) = q_0x^{n-2} + q_1x^{n-3} + \cdots + q_{n-2}$ and remainder $R(x) = r_0x + r_1$. If both $r_0 = 0$ and $r_1 = 0$, then $K_0(x)$ is a quadratic factor of $P(x)$, and we are done. Otherwise we look for better choices for a and b, using Newton's method. The new ideas presented in this paper are (1) the two-loop algorithm in Section 5 for long division of a degree n polynomial $P(x)$ by a monic quadratic $K(x)$, and (2) the recursive algorithm in Section 6 for computing the partial derivatives $\partial r_0/\partial a$, $\partial r_0/\partial b$, $\partial r_1/\partial a$ and $\partial r_1/\partial b$.

KEYWORDS

Polynomial, quadratic, Newton's method

1 LONG DIVISION OF $P(x)$ BY $K(x)$

Long division of the polynomial $P(x) = c_0x^n + c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n$ by the quadratic $K(x) = x^2 + ax + b$ can be carried out using the following long division scheme, shown in Section 2. In this scheme, the quotient obtained is $Q(x) = q_0x^{n-2} + q_1x^{n-3} + \cdots + q_{n-2}$ and the remainder is $R(x) = r_0x + r_1$, where $r_0 = q_{n-1} = c_{n-1} - bq_{n-3} - aq_{n-2}$ and $r_1 = q_n = c_n - bq_{n-2}$. The first row in this scheme is the divisor $K(x) = x^2 + ax + b$. The second row in this scheme are the column headers, which are the powers of x in decreasing order from x^n to 1. The third row are the coefficients q_0, q_1, q_2, $\cdots$, q_{n-2} of the quotient $Q(x)$, the coefficients only without the powers of x, which are in the column headers. The fourth row are the coefficients of the dividend polynomial $P(x)$, namely c_0, c_1, c_2, $\cdots$, c_n.

We obtain the fifth row by multiplying each of the terms of $K(x) = x^2 + ax + b$ by $q_0 = c_0$, to get the results q_0, aq_0, and bq_0, which we put on the fifth row under the columns for x^n, x^{n-1}, and x^{n-2}. Fourth row minus fifth row gives the results 0, $q_1 = c_1 - aq_0$, and $c_2 - bq_0$, which we put in the sixth row. Then we multiply $K(x) = x^2 + ax + b$ by q_1 to get the results q_1, aq_1, and bq_1 which we put in the seventh row, under the columns for for x^{n-1}, x^{n-2}, and x^{n-3}. Then we do sixth row minus seventh row and put the results in the eight row, and so on. This is just the traditional long division of $P(x)$ by $K(x)$.

The coefficients of the quotient polynomial obtained by this long division are, in sequence,

$$\begin{aligned} q_0 &= c_0, \\ q_1 &= c_1 - aq_0, \\ q_2 &= c_2 - bq_0 - aq_1, \\ &\cdots, \\ q_{n-2} &= c_{n-2} - bq_{n-4} - aq_{n-3} \end{aligned} \tag{1}$$

and the remainder is $R(x) = r_0x + r_1$, where

$$\begin{aligned} r_0 &= q_{n-1} = c_{n-1} - bq_{n-3} - aq_{n-2}, \\ r_1 &= q_n = c_n - bq_{n-2}. \end{aligned} \tag{2}$$

2 LONG DIVISION SCHEMA

Long division of the degree n polynomial $P(x) = c_0x^n + c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n$ by the quadratic factor $K(x) = x^2 + ax + b$ is shown in Figure 1.

3 NEWTON'S METHOD OF COMPUTING $K(x)$

Starting with initial guesses for the coefficients a and b of $K(x)$, say the initial guesses a_0 and b_0, we use long division of $P(x)$ by $K_0(x) = x^2 + a_0x + b_0$, to obtain a quotient $Q(x) = q_0x^{n-2} + q_1x^{n-3} + \cdots + q_{n-2}$ and remainder $R(x) = r_0x + r_1$. If both $r_0 = 0$ and $r_1 = 0$, then $K_0(x)$ is a quadratic factor of $P(x)$, and we are done. Otherwise we look for better choices for a and b, using the quadratically converging Newton's method. Note that both r_0 and r_1 are functions of a_0 and b_0, namely that $r_0 = f(a_0, b_0)$ and $r_1 = g(a_0, b_0)$, where $f()$ and $g()$ are well-defined by the long division of $P(x)$ by $K_0(x)$ in Section 1 and Section 2. Thus better guesses a_1 and b_1 can be obtained by solving the system of two nonlinear equations $f(a_0, b_0) = 0$ and $g(a_0, b_0) = 0$, by using the Newton's method.

For notational convenience, when treating as variables we use a and b, but when talking about specific values of a and b, we use the subscripted versions a_0, b_0, a_1, and b_1. We want the remainder of long division to be zero. That is, we want to solve the system of two nonlinear equations, namely

$$\begin{aligned} r_0 &= f(a, b) = 0, \\ r_1 &= g(a, b) = 0. \end{aligned}$$

Expand both functions $f()$ and $g()$ by Taylor series around the initial guess (a_0, b_0) to get the infinite series expansions, namely

$$\begin{aligned} f(a, b) &= f(a_0, b_0) + f_a(a_0, b_0)(a - a_0) + f_b(a_0, b_0)(b - b_0) + \cdots \\ g(a, b) &= g(a_0, b_0) + g_a(a_0, b_0)(a - a_0) + g_b(a_0, b_0)(b - b_0) + \cdots \end{aligned}$$

Here $f_a() = \partial f()/\partial a$, $f_b = \partial f()/\partial b$, $g_a() = \partial g()/\partial a$, and $g_b() = \partial g()/\partial b$. Using only the first three terms of each series as shown,

$x^2 + ax + b$

$\mathbf{x^n}$	$\mathbf{x^{n-1}}$	$\mathbf{x^{n-2}}$	$\mathbf{x^{n-3}}$	$\mathbf{x^{n-4}}$	$\cdots$	$\mathbf{x}$	$\mathbf{1}$
		q_0	q_1	q_2	$\cdots$	q_{n-3}	q_{n-2}
c_0	c_1	c_2	c_3	c_4	$\cdots$	c_{n-1}	c_n
$q_0 = c_0$	aq_0	bq_0					
0	$q_1 = c_1 - aq_0$	$c_2 - bq_0$					
	q_1	aq_1	bq_1				
	0	$q_2 = c_2 - bq_0 - aq_1$	$c_3 - bq_1$				
		q_2	aq_2	bq_2			
		0	$q_3 = c_3 - bq_1 - aq_2$	$c_4 - bq_2$			
			q_3	aq_3	$\cdots$		
			0	$q_4 = c_4 - bq_2 - aq_3$	$\cdots$		
					$\cdots$		
					$\cdots$	$q_{n-1} = c_{n-1} - bq_{n-3} - aq_{n-2}$	$q_n = c_n - bq_{n-2}$

Figure 1: Long division of the degree n polynomial $P(x) = c_0x^n + c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n$ by the quadratic factor $K(x) = x^2 + ax + b$

by dropping the second and higher degree terms in $(a - a_0)$ and $(b - b_0)$, and equating them to zero, we get

$$\begin{bmatrix} f(a_0, b_0) \\ g(a_0, b_0) \end{bmatrix} + \begin{bmatrix} f_a(a_0, b_0) & f_b(a_0, b_0) \\ g_a(a_0, b_0) & g_b(a_0, b_0) \end{bmatrix} \cdot \begin{bmatrix} a - a_0 \\ b - b_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

By transposing the first matrix to the right hand side of the equation, then multiplying both sides of the equation by the inverse of the square matrix, we get the resulting equation

$$\begin{bmatrix} a - a_0 \\ b - b_0 \end{bmatrix} = -\begin{bmatrix} f_a(a_0, b_0) & f_b(a_0, b_0) \\ g_a(a_0, b_0) & g_b(a_0, b_0) \end{bmatrix}^{-1} \cdot \begin{bmatrix} f(a_0, b_0) \\ g(a_0, b_0) \end{bmatrix}$$

Carrying out the indicated matrix operations, and simplifying, we get our desired solution for (a, b), namely

$$\begin{aligned} a &= a_0 - \frac{f(a_0, b_0)g_b(a_0, b_0) - g(a_0, b_0)f_b(a_0, b_0)}{D} \\ b &= b_0 - \frac{g(a_0, b_0)f_a(a_0, b_0) - f(a_0, b_0)g_a(a_0, b_0)}{D} \\ D &= f_a(a_0, b_0)g_b(a_0, b_0) - g_a(a_0, b_0)f_b(a_0, b_0) \end{aligned} \tag{3}$$

If the determinant D is nonzero, we are able to compute solutions for new values of a and b, using the above equations.

4 NEWTON'S METHOD

We summarize the Newton's method for computing a monic quadratic factor $K(x)$ of polynomial $P(x)$ as follows.

(1) We start with some initial guesses for the values of a and b, say our initial guesses are $a_0 = 1$ and $b_0 = 2$.
(2) Then we do long division of polynomial $P(x)$ by the quadratic $K_0(x) = x^2 + a_0x + b_0$, to get the quotient coefficients $q_0, q_1, q_2, \cdots, q_{n-2}$, and the remainder coefficients r_0 and r_1.
(3) If $r_0 = 0$ and $r_1 = 0$, then $K_0(x)$ is a quadratic factor of $P(x)$ and we stop computation.
(4) Otherwise, if either r_0 or r_1 is nonzero, we compute "better" values of a and b, using the Newton's formulas in Equation 3 in Section 3 above to obtain new values a_1 and b_1. Then we replace a_0 and b_0 by a_1 and b_1, and repeat the computation from Step 2.

The Newton's method is known to converge to the actual a and b values quadratically. That is, if the error of the current guess is $\epsilon = \max\{|a - a_0|, |b - b_0|\}$, then the error of the next guess is approximately ϵ^2. We shall observe this quadratic convergence in the examples that will be given in Section 9.

5 COMPUTER CODE FOR DIVISION OF $P(x)$ BY $K_0(x)$

The following is a C program fragment for doing long division of the degree n polynomial $P(x)$ by the monic quadratic $K_0(x)$, that is explained in Section 1.

The inputs to the program are: (1) the coefficients $c_0, c_1, c_2, \cdots, c_n$ of the dividend polynomial $P(x)$, and (2) the coefficients a_0 and b_0 of the divisor polynomial $K_0(x)$. (3) `DEG` is the degree n of the polynomial $P(x)$.

The outputs of the program are (1) the coefficients $q_0, q_1, q_2, \cdots, q_{n-2}$ of the quotient $Q(x)$, and (2) the coefficients $r_0 = q_{n-1}$ and $r_1 = q_n$ of the remainder $R(x)$.

```
for(j = 0; j <= DEG; j++) q[j] = c[j];
for(j = 1; j < DEG; j++) {
    q[j] = q[j] - (q[j-1] * a0);
    q[j+1] = q[j+1] - (q[j-1] * b0);
}
```

In the first loop, we copy the coefficients `c[j]` to `q[j]` for j from 0 to n. Here we copy `c[0]` to `q[0]`.

In the second loop, we compute the quotient coefficients `q[j]` for j from 1 to $n - 2$ and remainder coefficients `q[j]` for j from $n - 1$ to n. In this second loop, we actually compute `q[1]=c[1]-a0*q[0]`. We then compute `q[j]=c[j] -b0*q[j-2] -a0*q[j-1]` for j from 2 to $n - 1$. Finally we compute `q[n]=c[n] -b0*q[n-2]`.

6 COMPUTING $f()$ AND $g()$ AND DERIVATIVES

Consider the remainder $R(x) = r_0x + r_1$ obtained when we do long division of $P(x)$ by the quadratic divisor $K(x)$. The coefficient r_0 is a function of a and b, since $r_0 = q_k = c_k - bq_{k-2} - aq_{k-1}$, where $k = n - 1$. Thus we can rewrite r_0 as $q_k(a, b) = f(k, a, b)$, where $k = n - 1$. Similarly the coefficient r_1 is a function of a and b, since $r_1 = q_k = c_k - bq_{k-2}$ where $k = n$. Thus we can rewrite r_1 as $q_k(a, b) = g(k, a, b)$, where $k = n$.

Let us define a new function $F(k, a, b) = q_k, k = 0, 1, 2, \cdots, n$. Thus our functions $f()$ and $g()$ and be defined as follows $f(a, b) =$

$F(n-1,a,b)$ and $g(a,b) = F(n,a,b)$. From the computation in Equation 1 and Equation 2 (in Section 1) of the coefficients q_k, with $k = 0, 1, 2, \cdots, n$, we can define $F(k,a,b)$ casewise as follows.

$$F(k,a,b) = \begin{cases} c_0 & k = 0 \\ c_1 - c_0 a & k = 1 \\ c_k - aF(k-1,a,b) - bF(k-2,a,b) & k = 2,3,\cdots,n-1 \\ c_k - bF(k-2,a,b) & k = n \end{cases}$$

We can compute the partial derivatives $F_a()$ and $F_b()$ by differentiating the above definitions. The partial derivative $F_a()$ and its C program implementation `long double dFa()` are

$$F_a(k,a,b) = \begin{cases} 0 & k = 0 \\ -c_0 & k = 1 \\ -aF_a(k-1,a,b) - F(k-1,a,b) & \\ \quad -bF_a(k-2,a,b) & k = 2,3,\cdots,n-1 \\ -bF_a(k-2,a,b) & k = n \end{cases}$$

```
long double dFa(int k)
{
  if(k == 0) return 0.0;
  if(k == 1) return -q[0];
 if(k>=2 && k<DEG) return -a0*dFa(k-1)-q[k-1]-b0*dFa(k-2);
  if(k == DEG) return -b0*dFa(k-2);
}
```

The partial derivative $F_b()$ and its C program implementation `long double dFb()` are

$$F_b(k,a,b) = \begin{cases} 0 & k = 0,1 \\ -aF_b(k-1,a,b) - bF_b(k-2,a,b) & \\ \quad -F(k-2,a,b) & k = 2,3,\cdots,n-1 \\ -bF_b(k-2,a,b) - F(k-2,a,b) & k = n \end{cases}$$

```
long double dFb(int k)
{
  if(k == 0 || k == 1) return 0.0;
 if(k>=2 && k<DEG) return -a0*dFb(k-1)-b0*dFb(k-2)-q[k-2];
  if(k == DEG) return -b0*dFb(k-2)-q[k-2];
}
```

The four partial derivatives $f_a()$, $f_b()$, $g_a()$, $g_b()$ that we need to solve Equation 3 are therefor:

$$\begin{aligned} f_a(a,b) &= F_a(n-1,a,b) \\ f_b(a,b) &= F_b(n-1,a,b) \\ g_a(a,b) &= F_a(n,a,b) \\ g_b(a,b) &= F_b(n,a,b) \end{aligned}$$

7 IMPLEMENTATION OF NEWTON'S METHOD

Given the values of the coefficients $c_0, c_1, c_2, \cdots, c_n$ of the dividend polynomial $P(x)$, and the initial guesses a_0 and b_0 of the coefficients of the divisor polynomial $K_0(x)$, the following C code fragment carries out Newton's method iteratively, at most `NTRIES` number of times, or until $\{a_0 \neq 0$ and $|(a_1 - a_0)/a_0| < \epsilon$ and $b_0 \neq 0$ and $|(b_1 - b_0)/b_0| < \epsilon\}$, whichever comes first. The long division algorithm is the one discussed in Section 5, and the partial derivatives are recursively computed using the code in Section 6.

```
for(i = 1; i <= NTRIES; i++) {
   // Long division algorithm
   for(j = 0; j <= DEG; j++) q[j] = c[j];
   for(j = 1; j < DEG; j++) {
      q[j] = q[j] - (q[j-1] * a0);
      q[j+1] = q[j+1] - (q[j-1] * b0);
   }
   // Remainder coefficients r[0] and r[1]
   ff = q[DEG-1];
   gg = q[DEG];
   if(ff == 0.0 && gg == 0.0) break;
   // The four partial derivatives
   fa = dFa(DEG-1);
   fb = dFb(DEG-1);
   ga = dFa(DEG);
   gb = dFb(DEG);
   // Computing the determinant
   dt = fa * gb - ga * fb;
   if(ABS(dt) < ZERO) exit(1);
   da = (ff * gb - gg * fb) / dt;
   db = (gg * fa - ff * ga) / dt;
   // Computing new values a1 and b1
   a1 = a0 - da;
   b1 = b0 - db;
   if(a0 != 0.0 && ABS(da/a0) < EPS &&
       b0 != 0.0 && ABS(db/b0) < EPS) break;
   a0 = a1;
   b0 = b1;
}
```

The source code of the complete C program `qfactor.c` is available by writing email to the author. The program contains input and output so that it is a stand-alone program. It looks for one quadratic factor, given initial values of a and b, and after the quadratic factor is found, computes the two roots of the quadratic.

8 USING `QFACTOR.C` PROGRAM

The program requires that the user enter the $(n+1)$ real coefficients of the degree n polynomial $P(x) = c_0x^n + c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n$, and then enter the initial guess for the two coefficients a_0 and b_0 of the quadratic factor $K(x) = x^2 + a_0x + b_0$. These values can be entered using any of these formats: (1) integer like 23, -57, or 0, (2) decimal number like -10.75, 0.0, or 52.5, (3) scientific notation like -2.234e14 meaning $-2.234 \cdot 10^{14}$, or 76.34e-12, meaning $76.34 \cdot 10^{-12}$, or (4) any mix of these formats.

Upon completing the entry of the coefficients of the polynomial and the initial guess of the coefficients of the quadratic factor, the program computes better and better guesses for the values of a and b, until the values are correct to a predetermined accuracy (`EPS`,ϵ), or until the maximum number of iterations (`NTRIES`) of Newton's method is reached, whichever comes first. Then the program stops and prints (1) the degree $(n-2)$ quotient polynomial $Q(x)$, (2) the quadratic factor $K(x)$ that the Newton's method found, and (3) the two roots of this quadratic factor. At this point, the user can do one of three procedures.

Procedure 1. The user can stop.
Procedure 2. The user can enter a new set of starting values a_0 and b_0 in order to possibly find a new quadratic factor of the original degree n polynomial
Procedure 3. The user can stop the current program run, then run the program again, but using as input the degree $n-2$ polynomial printed out by the previous run, using copy-paste from the previous run to the new program run, so that there is no retyping of coefficient values. Then we can continue this "collapsing" of degree of the polynomial until we get all the roots.

9 SOLVED EXAMPLES

For the examples given here, we sometimes have to break a single long line of computer output (printout) into two or more lines, so that the entire line can be seen on the page. Other than that, we made no modifications in the computer output.

9.1 Example from Rafiq [8]

Find a quadratic factor of the following degree four polynomial, which is Example 3 in Rafiq [8]: $P(x) = 1.9520 \cdot 10^{-14}x^4 - 9.5838 \cdot 10^{-11}x^3 + 9.7215 \cdot 10^{-8}x^2 + 1.671 \cdot 10^{-4}x - 0.20597$. When we start with $a0 = 100$ and $b0 = -100$, we get the quadratic factor $x^2 + 163.940632x - 1452496.708774$, with roots 1126.009751 and -1289.950382, which are the same real roots found by Rafiq [8].

```
Enter degree of polynomial [>2]: 4
Enter 5 coeffs decr order: 1.952e-14 -9.5838e-11
9.7215e-08 0.0001671 -0.20597
Deg 4 poly entered: 1.952e-14 -9.5838e-11 9.7215e-08
0.0001671 -0.20597
Enter a b [0 0 to end]: 100 -100
Quad coeffs:
Iter 0: 100.0000000000000000 -100.0000000000000000
Iter 1: -204.3860830346749444 -1925019.3555461554128669
Iter 2: 411.0866081826201767 -1134217.9762273866740543
Iter 3: 210.0010925603964294 -1393117.1294690310264741
Iter 4: 165.9440620727080588 -1449912.9227901836666206
Iter 5: 163.9446382221588250 -1452491.5408436994840713
Iter 6: 163.9406316534402896 -1452496.7087531190807113
Iter 7: 163.9406316373749578 -1452496.7087738420658525
Iter 8: 163.9406316373749578 -1452496.7087738420659662
Deg 2 Quotient: 0.000000 -0.000000 0.000000
Quadratic factor: x^2 +163.940632x -1452496.708774
Roots are:1126.009751 and -1289.950382
```

When we start with $a0 = -5000$ and $b0 = 6000000$, we get the quadratic factor $x^2 - 5073.674238x + 7264554.707450$, with complex roots $2536.837119 + 910.501037\mathbf{i}$ and $2536.837119 - 910.501037\mathbf{i}$, which are the same complex roots found by Rafiq [8].

```
Enter a b [0 0 to end]: -5000 6000000
Quad coeffs:
Iter 0: -5000.0000000000000000 6000000.0000000000000000
Iter 1: -4925.3135204350711991 6739960.3174731292501747
Iter 2: -5099.3963153108729891 7334285.4845929701623390
Iter 3: -5074.1386846732093225 7265539.2090234159468309
Iter 4: -5073.6743357811892570 7264554.7901784460523231
Iter 5: -5073.6742381947481726 7264554.7074500004719084
Iter 6: -5073.6742381947520069 7264554.7074500292092125
Deg 2 Quotient: 0.000000 0.000000 -0.000000
Quadratic factor: x^2 -5073.674238x +7264554.707450
Roots are:2536.837119+910.501037i and
2536.837119-910.501037i
```

9.2 Laguerre polynomials [12]

Find a quadratic factor of the fifth degree Laguerre polynomial $L_5(x) = (-x^5 + 25x^4 - 200x^3 + 600x^2 - 600x + 120)/120$. For purposes of finding quadratic factors, we can drop the factor $1/120$ and multiply by -1 to get the simpler polynomial $-120 \cdot L_5(x) = x^5 - 25x^4 + 200x^3 - 600x^2 + 600x - 120$. If we start with $a0 = -1$ and $b0 = 1$, the roots found are 1.413403 and 0.263560, which are the same roots given by Salzer [11].

```
Enter degree of polynomial [>2]: 5
Enter 6 coeffs decr order: 1 -25 200 -600 600 -120
Deg 5 poly entered: 1 -25 200 -600 600 -120
Enter a b [0 0 to end]: -1 1
Quad coeffs:
Iter 0: -1.0000000000000000 1.0000000000000000
Iter 1: -1.4742143692395518 0.4460485972359056
Iter 2: -1.6512809504470201 0.3744097884444698
Iter 3: -1.6764817698291903 0.3724802596240043
Iter 4: -1.6769632072206185 0.3725169354173948
Iter 5: -1.6769633788246361 0.3725169621487077
Iter 6: -1.6769633788246577 0.3725169621487120
Deg 3 Quotient: 1.000000 -23.323037 160.515605
-322.132982
Quadratic factor: x^2 -1.676963x +0.372517
Roots are: 1.413403 and 0.263560
```

If we run the qfactor program again, giving the degree three quotient above as input, namely the polynomial $1.000000x^3 - 23.323037x^2 + 160.515605x - 322.132982$, and if we start with $a0 = -1$ and $b0 = 1$, the roots found are 7.085809 and 3.596426, which are the same roots given by Salzer [11]. The remaining degree one quotient, namely $1.000000x - 12.640802$, has root 12.640802.

```
Enter degree of polynomial [>2]: 3
Enter 4 coeffs decr order: 1.000000 -23.323037
160.515605 -322.132982
Deg 3 poly entered: 1 -23.323 160.516 -322.133
Enter a b [0 0 to end]: -1 1
Quad coeffs:
Iter 0: -1.0000000000000000 1.0000000000000000
Iter 1: -6.7919792691737646 14.6899797401748590
Iter 2: -9.4972166412802723 21.8904845703736939
Iter 3: -10.4967954382757727 24.8820133132670366
Iter 4: -10.6762176607512623 25.4632166840639728
Iter 5: -10.6822283676961523 25.4835643087874602
Iter 6: -10.6822350233573271 25.4835874017388127
Iter 7: -10.6822350233654133 25.4835874017672732
Iter 8: -10.6822350233654132 25.4835874017672732
Deg 1 Quotient: 1.000000 -12.640802
Quadratic factor: x^2 -10.682235x +25.483587
Roots are: 7.085809 and 3.596426
```

To summarize, the five roots of the Laguerre polynomial $L_5(x)$ are 0.263560, 1.413403, 3.596426, 7.085809, and 12.640802, which are the five roots given by Salzer [11].

9.3 Legendre polynomials [13]

Find a quadratic factor of the eight degree Legendre polynomial $P_8(x) = (6435x^8 - 12012x^6 + 6930x^4 - 1260x^2 + 35)/128$. This polynomial is symmetric since $P_8(-x) = P_8(x)$, and so we only need to find the positive roots. For purposes of finding quadratic factors, we can drop the factor 1/128. If we start with $a0 = -1$ and $b0 = 1$, the roots found are 0.525532 and 0.183435, which are the same roots given by Lowan [5].

```
Enter degree of polynomial [>2]: 8
Enter 9 coeffs decr order: 6435 0 -12012 0 6930 0
-1260 0 35
Deg 8 poly entered: 6435 0 -12012 0 6930 0 -1260 0 35
Enter a b [0 0 to end]: -1 1
Quad coeffs:
Iter 0: -1.0000000000000000 1.0000000000000000
Iter 1: -0.9165466406754666 0.6862151332071878
Iter 2: -0.8481340483475369 0.4592487346715779
Iter 3: -0.7940460635972765 0.2991915280826729
Iter 4: -0.7537103360782697 0.1922807479153829
Iter 5: -0.7267567446934254 0.1292608837302444
Iter 6: -0.7128130963480143 0.1020724784389720
Iter 7: -0.7091617860047420 0.0966138318667571
Iter 8: -0.7089674431118241 0.0964011661958203
Iter 9: -0.7089670524130436 0.0964008497335634
Iter 10: -0.7089670524119788 0.0964008497328791
Iter 11: -0.7089670524119788 0.0964008497328791
Deg 6 Quotient: 6435.000000 4562.202982 -9397.887867
-7102.593104 2800.459878 2670.129796 363.067339
Quadratic factor: x^2 -0.708967x +0.096401
Roots are:0.525532 and 0.183435
```

If we start with $a0 = -1$ and $b0 = 0$, the roots found are 0.960290 and 0.796666, which are the same roots given by Lowan [5].

```
Enter a b [0 0 to end]: -1 0
Quad coeffs:
Iter 0: -1.0000000000000000 0.0000000000000000
Iter 1: -1.3479941413924910 0.3763440860215054
Iter 2: -1.9508272202635273 0.9464537929883982
Iter 3: -1.8603954658225640 0.8624518647859902
Iter 4: -1.7996517864098767 0.8055078706089895
Iter 5: -1.7674303038954489 0.7750142253360990
Iter 6: -1.7577791385829587 0.7658177616274100
Iter 7: -1.7569619157828724 0.7650360866802702
Iter 8: -1.7569563341699185 0.7650307375201544
Iter 9: -1.7569563339111630 0.7650307372719293
Iter 10: -1.7569563339111630 0.7650307372719293
Deg 6 Quotient: 6435.000000 11306.014009 2929.200130
-3502.971512 -1465.496120 105.068188 45.749796
Quadratic factor: x^2 -1.756956x +0.765031
Roots are:0.960290 and 0.796666
```

By symmetry of the Legendre polynomial, the eight roots are therefore ±0.183435, ±0.525532, ±0.796666, and ±0.960290.

9.4 Constant B3 of Bifurcation Theory [6]

"B3 is the third bifurcation point of the logistic map $x_{k+1} = rx_k(1 - x_k)$, which exhibits period doubling shortly before the onset of chaos. Computations using a predecessor algorithm to PSLQ found that B3 is a root of the polynomial, $0 = 4913 + 2108t^2 - 604t^3 - 977t^4 + 8t^5 + 44t^6 + 392t^7 - 193t^8 - 40t^9 + 48t^{10} - 12t^{11} + t^{12}$". if we start with $a0 = -1$ and $b0 = 1$, the roots found are 3.960769 and 3.544090, and 3.544090 is the same root given in MathOverflow [6].

```
Enter degree of polynomial [>2]: 12
Enter 13 coeffs decr order: 1 -12 48 -40 -193 392 44
8 -977 -604 2108 0 4913
Deg 12 poly entered: 1 -12 48 -40 -193 392 44 8 -977
-604 2108 0 4913
Enter a b [0 0 to end]: -1 1
Quad coeffs:
Iter 0: -1.0000000000000000 1.0000000000000000
Iter 1: 0.9604192460035566 -0.7738861558109435
Iter 2: -6.0041965546515304 -11.2421615740261158
Iter 3: -5.4731756863249682 -10.4464404566472117
Iter 4: -4.9946061671995209 -9.7292755261615639
Iter 5: -4.5643126702724995 -9.0843440289446050
Iter 6: -4.1786778608638634 -8.5060436004679316
Iter 7: -3.8347000133191092 -7.9893710690607148
Iter 8: -3.5301517678797782 -7.5295554740640297
Iter 9: -3.2640087644562776 -7.1207867148359447
...
Iter 41: -7.5048590119585681 14.0373219974098523
Deg 10 Quotient: 1.000000 -4.495141 0.227279 24.805436
-10.029086 -31.468765 -51.387132 64.084002 225.279118
187.120245 349.995533
Quadratic Factor: x^2 -7.504859x +14.037322
Roots are: 3.960769 and 3.544090
```

10 PROGRAM LIMITATIONS

The C program presented here uses the data type `long double` for all floating point computations. The number of decimal digits of precision of `long double` depends on the hardware and the C compiler used. In this paper, we used an Intel x86_64 and gcc-11.4.0, which provide 17-18 digits of precision. For problems where coefficients of the polynomial exceed 18 digits, the C program given here is not the right tool. Instead, the algorithms given here should be implemented using the GNU MultiPrecision (GMP) library.

11 CONCLUSION

This paper derives the mathematical formulas for numerically computing a quadratic factor $K(x) = x^2+ax+b$ of a degree n polynomial $P(x) = c_0x^n + c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n$ using Newton"s Method. It also provides a C program implementation of this method. The author believes that the C program implementation of the computation of the quotient $Q(x) = q_0x^{n-2}+q_1x^{n-3}+\cdots+q_{n-2}$ and the remainder is $R(x) = r_0x + r_1$ is new, although the Division Algorithm is quite old and well known, the particular C program implementation in this paper is new. In particular, the C program implementation of the recursive evaluation of the partial derivatives $\partial r_0/\partial a$, $\partial r_0/\partial b$, $\partial r_1/\partial a$ and $\partial r_1/\partial b$ is new.

REFERENCES

[1] Burachik, R.S., Caldwell, B.I., Kaya, C.Y., "A generalized multivariable Newton method", as cited in *https://fixedpoint theoryandalgorithms.springeropen.com/articles/10.1186/s13663-021-00700-9*

[2] Eagan, N. and Hauser, G., "Newton's Method on a System of Nonlinear Equations" as cited in *https://www.cmu.edu/math/ undergrad/suami/pdfs/2014_newton_method.pdf*
[3] Gaik, T.K., Kahar, R.A., Long, K.S., "Solving non-linear systems by Newton's method using spreadsheet Excel" as cited in *https://core.ac.uk/ download/pdf/12006987.pdf*
[4] Licht, M., "Newton's method", UC San Diego, Winter Quarter 2021, as cited in *https://mathweb.ucsd.edu/ mlicht/wina2021/pdf/lecture11.pdf*
[5] Lowan, A.N., Davids, N., Levenson, A., "Table of zeroes of the Legendre Polynomials of order 1-16 and the weight coefficients for Gauss' mechanical quadrature formula", as cited in *https://www.ams.org/ journals/bull/1942-48-10/S0002-9904-1942-07771-8 /S0002-9904-1942-07771-8.pdf*
[6] Mathoverflow, "What Are Some Naturally-Occurring High-Degree Polynomials?",as cited in *https://mathoverflow.net/ questions/27324/what-are-some-naturally- occurring-high-degree-polynomials*
[7] Overton,M., "Quadratic Convergence of Newton's Method" in Numerical Computing, Spring 2017, as cited in *https://cs.nyu.edu/ overton/NumericalComputing/newton.pdf*
[8] Rafiq, N., Shams, M., Mir, N.A., Gaba, Y.U., "A Highly Efficient Computer Method for Solving Polynomial Equations Appearing in Engineering Problems", as cited in *https://www.hindawi.com/ journals/mpe/2021/9826693/*
[9] Remani, C., "Numerical Methods for Solving Systems of Nonlinear Equations" as cited in *https://www.lakeheadu.ca/sites/default/ files/uploads/77/docs/RemaniFinal.pdf*
[10] Saheya, B., Chen, G.Q., Sui, Y.K., Wu, C.Y., "A new Newton-like method for solving nonlinear equations" as cited in *https:// springerplus.springeropen.com/articles/10.1186/s40064-016-2909-7*
[11] Salzer, H.E., Zucker, R., "Table of the zeros and weight factors of the first fifteen Laguerre polynomials", as downloaded from *https://projecteuclid.org/journals/bulletin-of-the-american- mathematical-society/volume-55/issue-10/Table-of-the-zeros-and- weight-factors-of-the-first/bams/1183514167.full*
[12] Wikipedia, "Laguerre polynomials", as cited in *https://en.wikipedia.org/ wiki/Laguerre_polynomials*
[13] Wikipedia, "Legendre polynomials", as cited in *https://en.wikipedia.org/ wiki/Legendre_polynomials*
[14] Wikipedia, "Proof of quadratic convergence for Newton's iterative method" in the article 'Newton's method' as cited in *https://en.wikipedia.org/wiki/Newtonś_method*