# CSC349A Numerical Analysis

## Lecture 11

Rich Little

University of Victoria

2023

# Table of Contents I

# Introduction

A polynomial of order (degree) $n$ can be written as

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

as well as

$$f(x) = a_n (x - r_1)^{m_1} (x - r_2)^{m_2} \ldots (x - r_k)^{m_k} \text{ with } \sum_{j=1}^{k} m_j = n$$

if $f(x)$ has $k$ distinct roots (real or complex) and $r_j$ is a zero of multiplicity $m_j \geq 1$. If the coefficients $a_i$ are real, then any complex roots occur in conjugate pairs, $\lambda \pm \mu i$ where $i = \sqrt{-1}$.

# Polynomial roots using Newton/Raphson

University of Victoria
Computer Science

One approach to computing the roots of a polynomial $f(x)$ is to use the Newton/Raphson method.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Main issues:

- Efficient evaluation of $f(x_i)$ and $f'(x_i)$.
- How to implement Newton to compute all $n$ roots of $f(x)$
- How to compute complex roots

# Table of Contents I

# Horner's Algorithm

Given a polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ and a value $x_0$, this algorithm is used to efficiently evaluate $f(x_0)$ and $f'(x_0)$. To illustrate the basic idea, consider the case $n = 4$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \tag{1}$$

can be rewritten in the form:

# Horner's Algorithm

This form of the polynomial is more efficient to compute.

$$f(x) = a_0 + x * (a_1 + x * (a_2 + x * (a_3 + x * a_4))) \qquad (2)$$

Evaluation of (1) at $x_0$ requires 10 multiplications and 4 additions, whereas (2) requires only 4 multiplications and 4 additions. The general case (for a polynomial of order $n$): form (1) requires $n(n+1)/2$ multiplications and $n$ additions, from (2) requires $n$ multiplications and $n$ additions

# Deriving the Algorithm Example

Consider the nested form again:

$$f(x) = a_0 + x * (a_1 + x * (a_2 + x * (a_3 + x * a_4)))$$

# The algorithm

Evaluate $f(x_0)$, assuming that $f(x) = \sum_{i=0}^{n} a_i x^i$ is written in the **"nested" form**, as in (2):

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + b_n x_0 \\ b_{n-2} &= a_{n-2} + b_{n-1} x_0 \\ \dots & \\ b_0 &= a_0 + b_1 x_0 \\ b_0 &= f(x_0) \end{aligned}$$

# Number of arithmetic operations

More compact form:

$$b_k = a_k + b_{k+1}x_0 \quad \text{for } k = n-1, n-2, \ldots, 1, 0$$

**NOTE** that execution of this algorithm requires **exactly** $n$ multiplications and $n$ additions.
Furthermore, if we rearrange and solve for each $a_k$ we get

$$a_k = b_k - b_{k+1}x_0$$

# Algorithm for evaluating $f'(x_0)$

Let $a_n, a_{n-1}, \ldots, a_0$ be defined as above, and consider:

$$\begin{aligned} f(x) &= a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n \\ &= (b_0 - b_1 x_0) + (b_1 - b_2 x_0)x + \cdots + (b_{n-1} - b_n x_0)x^{n-1} + b_n x^n \\ &= (x - x_0)(b_1 + b_2 x + b_3 x^2 + \cdots + b_n x^{n-1}) + b_0 \end{aligned}$$

Let:

$$Q(x) = b_1 + b_2 x + b_3 x^2 + \cdots + b_n x^{n-1}$$

then

$$f(x) = (x - x_0)Q(x) + b_0$$

# Algorithm for evaluating $f'(x_0)$

Differentiating with respect to $x$ gives

$$f'(x) = Q(x) + (x - x_0)Q'(x)$$

which implies that

$$f'(x_0) = Q(x_0)$$

Thus, to evaluate $f'(x_0)$, one first needs to evalute $f(x_0)$ as above, which gives the coefficients $b_n, b_{n-1}, \ldots, b_0$, and then evaluate $Q(x_0)$. The most efficient way to evalute $Q(x_0)$, is to use the nested form for the polynomial $Q(x)$.

# Horner's algorithm

The following algorithm evaluates both $f(x)$ and $f'(x_0) = Q(x_0)$ using *nest multiplication* to evaluate both of the polynomials.

**HORNER'S ALGORITHM**

Given values $a_0, a_1, \ldots, a_n$ and $x_0$, compute:

$$\begin{aligned} b_n &= a_n & c_n &= b_n \\ b_{n-1} &= a_{n-1} + b_n x_0 & c_{n-1} &= b_{n-1} + c_n x_0 \\ b_{n-2} &= a_{n-2} + b_{n-1} x_0 & c_{n-2} &= b_{n-2} + c_{n-1} x_0 \\ &\ldots \\ b_0 &= a_0 + b_1 x_0 & c_1 &= b_1 + c_2 x_0 \end{aligned}$$

$$Then$$

$$b_0 = f(x_0) \qquad c_1 = f'(x_0)$$

# EXAMPLE

Let $f(x) = x^4 - 2x^3 + 2x^2 - 3x + 4$ and use Horner's algorithm to evaluate $f(1)$ and $f'(1)$:

# Note

The explicit form of $f'(x)$, namely

$$f'(x) = 4x^3 - 6x^2 + 4x - 3$$

is not obtained; only the *value* of $f'(1)$ is computed. Since $Q(x)$ depends on the value of $x_0$, which is equal to 1 above, all computations must be re-done in order to evaluate $f'(x)$ at a different value of $x$.

# Table of Contents I

# Polynomial Deflation

Having computed one zero, say $r_1$ of a polynomial $f(x)$ having $n$ zeros $r_1, r_2, \ldots, r_n$ the deflated polynomial is

$$\hat{f}(x) = \frac{f(x)}{x - r_1}$$

Note that $\hat{f}(x)$ is a polynomial of order $n-1$ having roots

$$r_2, \ldots, r_n$$

$\hat{f}(x)$ can be easily determined from Horner's algorithm.

# Table of Contents I

# Newton's algorithm with Horner

Outline of a procedure to compute a zero of a polynomial $f(x)$ using Newton's method and Horner's algorith:

- Let $x_0$ be an initial approximation to a zero of $f(x)$
- for i = 1 to imax
  use Horner's algorithm to evaluate $f(x_{i-1})$ and $f'(x_{i-1})$
  set $x_i \leftarrow x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$
  if $\left|1 - \frac{x_{i-1}}{x_i}\right| < \varepsilon$ exit
  end
  output failed to converge in imax iterations

# Polynomial Deflation

Suppose that the values $x_0, x_1, x_2, \ldots$ computed above converge in $N$ iterations. Then $x_N$ is the final computed approximation to some zero, say $r_1$ of $f(x)$. Now the final computation in the above procedure with Newton's method (after $N$ iterations) is:

$$x_N \leftarrow x_{N-1} - \frac{f(x_{N-1})}{f'(x_{N-1})}$$

If $b_n, b_{n-1}, \ldots, b_0$ are the values computed by Horner's algorithm to evalute $f(x_{N-1})$ that is, in the last step of the above procedure (when $i = N$), then from page 2 of Handout number 13 it follows that:

$$f(x) = (x - x_{N-1})Q(x) + b_0 \tag{3}$$

# Polynomial Deflation II

$$Q(x) = b_1 + b_2 x + b_3 x^2 + \cdots + b_n x^{n-1} \tag{4}$$

On letting $x = x_{N-1}$ in (3), we obtain:

$$b_0 = f(x_{N-1}) \approx 0 \text{ since } x_{N-1} \approx x_N \approx \text{ the zero } r_1 \text{ of } f(x)$$

Therefore from (3),

$$f(x) \approx (x - x_{N-1})Q(x)$$

and consequently

$$Q(x) \approx \frac{f(x)}{x - x_{N-1}}$$

# Polynomial Deflation III

That is, the polynomial $Q(x)$ defined in (4) above, is the **deflated polynomial**, it is a polynomial of degree $n-1$, whose zeroes are equal to those of $f(x)$, except for the zero at $x_{N-1} \approx r_1$. Note that the coefficients $b_1, b_2, \ldots, b_n$ of $Q(x)$ are determined from the last application (when $i = N$) of Horner's algorithm in the procedure at the beginning of these notes.

**Note:** If several zeros of $f(x)$ are approximated as above, and several deflations are carried out giving a sequence of deflated polynomials of degrees $n-1, n-2, n-3, \ldots,$ then the successive computed zeros tend to become less and less accurate.

# Root Polishing

Aply Newton's method to approximate deflated polynomial $Q(x)$, giving a value $\hat{r}$. The value $\hat{r}$ approximates some root $r_2$ of $f(x)$, but will not be fully accurate. Use $\hat{r}$ as the initial approximation for Newton's method applied to $f(x)$. This will converge very quickly (1 or 2 iterations) to the fully accurate root $r_2$ (as $\hat{r}$ is very clsoe to $r_2$).

# Complex roots using Newton

One approach is to use Newton's method with complex arithmetic. This requires a complex-valued initial value $x_0$. Usually needs a very good initial approximation to a complex root for convergence.