UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 204 *Numerical methods*

Evaluating a polynomial at a point and Horner's rule

Douglas Wilhelm Harder, LEL, M.Math.
dwharder@uwaterloo.ca
dwharder@gmail.com

Introduction

- In this topic, we will
  - Describe the representation of polynomials
  - Discuss the idea of evaluating polynomials
  - Look at a sequence of more efficient implementations
    - We will use C++
  - Describe the evaluation of polynomials in MATLAB

Representing a polynomial

- As with our representation, in C++, we will represent a polynomial as an array where `a[k]` is the coefficient of x^k

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

  – For example:

```
double a[3]{ 2.0, 3.0, 1.0 }; // x^2 + 3x + 2
```

Representing a polynomial

- In MATLAB, a polynomial is represented with a vector:
  - An n-dimensional vector is polynomial of degree $n - 1$
  - If `p` is an n-dimensional vector, `p(k)` is the coefficient of x^{n-k}
- Consequently, the following represents $x^2 + 3x + 2$

```
>> p = [1 3 2];
```

- If a vector is passed to a function expecting a polynomial, the vector is interpreted as described above:

```
>> roots( p )
    ans =
         -2
         -1
```

This Matlab code is provided for demonstration purposes and is not required for the examination.

Representing a polynomial

- Some other useful polynomial functions in Matlab:

```
>> polyder( [1 3 2] )    % take the derivative
    ans =
        2   3
>> polyint( [1 3 2] )    % find the antiderivative
    ans =
        0.3333    1.5000    2.0000         0
```

This Matlab code is provided for demonstration purposes and is not required for the examination.

Evaluating a polynomial

- Suppose you have the polynomial

$$1.2\,x^4 - 3.8\,x^3 + 4.9\,x^2 - 0.7\,x + 5.6$$

  - How would you evaluate this at $x = 2.5$?
  - Author a function `pow( double x, unsigned int n )` and call it

Evaluating a polynomial

- The most expensive way of calculating x^n:

```
template <typename T>
T pow_O_n( T x, int n ) {
    if ( n >= 0 ) {
        T result{ 1.0 };

        for ( int k{1}; k <= n; ++k ) {
            result *= x;
        }

        return result;
    } else if ( n == INT_MIN ) {
        return pow_O_n( 1.0/x, -(n + 1) )/x;
    } else {
        return pow_O_n( 1.0/x, -n );
    }
}
```

$$x^n = \underbrace{x \cdot x \cdot x \cdots x}_{n \text{ time}}$$

$$\mathrm{O}(n)$$

$$x^n = \left(\frac{1}{x}\right)^{-n}$$

This C++ code is meant to demonstrate sub-optimal algorithms not required on the examination

Evaluating a polynomial

- The most expensive form:

```
template <typename T>
T polyval_O_n2( T coeffs[], unsigned int degree, T x ) {
    T result{ coeffs[0] };

    for ( unsigned int k{1}; k <= degree; ++k ) {
        result += coeffs[k]*pow_O_n( x, k );
    }

    return result;
}
```

$$O\left(n^2\right)$$

This C++ code is meant to demonstrate sub-optimal algorithms not required on the examination

Evaluating a polynomial

- A recursive means of calculating x^n:

```
template <typename T>
T pow_O_ln_n_rec( T x, int n ) {
    if ( n > 0 ) {
        T result{ pow_O_ln_n_rec( x, n/2 ) };
        result *= result;
        return ( (n&1) == 0 ) ? result : result*x;
    } else if ( n == 0 ) {
        return 1.0;
    } else if ( n == INT_MIN ) {
        return pow_O_ln_n_rec( 1.0/x, -(n + 1) )/x;
    } else {
        return pow_O_ln_n_rec( 1.0/x, -n );
    }
}
```

$$x^n = \begin{cases} 1 & n = 0 \\ \left(x^{\frac{n}{2}}\right)^2 & n \text{ is even} \\ x\left(x^{\frac{n-1}{2}}\right)^2 & n \text{ is odd} \end{cases}$$

$$\mathrm{O}(\ln(n))$$

This C++ code is meant to demonstrate sub-optimal algorithms not required on the examination

Evaluating a polynomial

- A more efficient approach:

```
template <typename T>
T polyval_O_n_ln_n_rec( T coeffs[], unsigned int degree, T x ) {
    T result{ coeffs[0] };

    for ( unsigned int k{1}; k <= degree; ++k ) {
        result += coeffs[k]*pow_O_ln_n_rec( x, k );
    }

    return result;
}
```

$$\mathrm{O}\left(n \ln\left(n\right)\right)$$

This C++ code is meant to demonstrate sub-optimal algorithms not required on the examination

Evaluating a polynomial

- An iterative means of calculating x^n:

```
template <typename T>
T pow_O_ln_n_iter( T x, int n ) {
    if ( n >= 0 ) {
        T result{ 1.0 };

        for ( ; n > 0; n <<= 1, x *= x ) {
            if ( (n&1) == 1 ) {
                result *= x;
            }
        }

        return result;
    } else if ( n == INT_MIN ) {
        return pow_O_ln_n_iter( 1.0/x, -(n + 1) )/x;
    } else {
        return pow_O_ln_n_iter( 1.0/x, -n );
    }
}
```

$$x^{21} = x^{16+4+1} = x^{16}x^{4}x^{1}$$

$$\mathrm{O}\left(\ln\left(n\right)\right)$$

This C++ code is meant to demonstrate sub-optimal algorithms not required on the examination

Evaluating a polynomial

- An even more efficient approach:

```
template <typename T>
T polyval_O_n( T coeffs[], unsigned int degree, T x ) {
    T result{ coeffs[0] };
    T term{ 1.0 };

    for ( unsigned int k{1}; k <= degree; ++k ) {
        term *= x;
        result += coeffs[k]*term;
    }

    return result;
}
```

$\mathrm{O}(n)$

  – A total of approximately $3n$ FLOPs

This C++ code is meant to demonstrate sub-optimal algorithms not required on the examination

Evaluating a polynomial

- All these evaluate the polynomial in the standard form

$$1.2\,x^4 - 3.8\,x^3 + 4.9\,x^2 - 0.7\,x + 5.6$$

  – What about rewriting it?

$$(1.2\,x - 3.8)\,x^3 + 4.9\,x^2 - 0.7\,x + 5.6$$

$$((1.2\,x - 3.8)\,x + 4.9)\,x^2 - 0.7\,x + 5.6$$

$$(((1.2\,x - 3.8)\,x + 4.9)\,x - 0.7)\,x + 5.6$$

    - This has only $2n$ FLOPs
    - This is known as *Horner's rule* for evaluating polynomials

Evaluating a polynomial

- This has approximately half the run time of the previous version:

```
template <typename T>
T polyval_horner( T coeffs[], unsigned int degree, T x ) {
    T result{ coeffs[degree] };

    for ( unsigned int k{degree - 1}; k <= degree; --k ) {
        result = result*x + coeffs[k];
    }

    return result;
}
```

$$O(n)$$

Evaluating a polynomial

- This implementation is approximately 1% faster:

```
template <typename T>
T polyval_horner_ptr( T coeffs[], unsigned int degree, T x ) {
    T *coeff{ coeffs + degree };
    T result{ *coeff };

    while ( coeff > coeffs ) {
        result = x*result + *--coeff;
    }

    return result;
}
```

  – If you require such efficiency, use assembly language…

This demonstrates effective use of pointer arithmetic, but is not required for the examination

Horner's rule

- In MATLAB, evaluating a polynomial is straight-forward:

```
>> p = [1 3 2];
>> polyval( p, 0.3 );   % calculate p(0.3)
    ans =
        2.9900
>> polyval( [1 3 2], [0.3 0.2; 0.5 -0.1] )
    ans =
        2.9900   2.6400
        3.7500   1.7100
```

Summary

- Following this topic, you now
  - Understand the representation of polynomials that:
    - We will use for C++
    - Is used in MATLAB
  - Have seen successively more efficient evaluations of polynomials
  - Are aware that this ends with the very efficient Horner’s rule
  - Know that in MATLAB, calling `polyval` will evaluate the polynomial using Horner’s rule

References

[1] https://en.wikipedia.org/wiki/Horner%27s_method

[2] https://www.mathworks.com/help/matlab/polynomials.html

Acknowledgments

None so far.

Colophon

These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

Mathematical equations are prepared in MathType by Design Science, Inc.

Examples may be formulated and checked using Maple by Maplesoft, Inc.

The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 204 *Numerical methods* course taught at the University of Waterloo. The material in it reflects the author's best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.