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

ECE 204 *Numerical methods*

Interpolating polynomials

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

Introduction

- In this topic, we will
  - See how to find interpolating linear and quadratic polynomials
  - Learn about the Vandermonde matrix and see how to interpolate n points with a polynomial of degree $n - 1$
  - Review why it is necessary to use partial pivoting when solving systems of linear equations
  - See how to decrease the error by shifting x-values before finding the interpolating polynomial
  - Look at implementations of functions for finding interpolating quadratic polynomials in both C++ and MATLAB

Linear interpolation

- Given two points (x_1, y_1), (x_2, y_2) with $x_1 \neq x_2$, you can find the straight line that passes through these two points
  - In secondary school, you likely found a line $y = mx + b$ as follows:

$$y_1 = mx_1 + b$$
$$y_2 = mx_2 + b$$

  - Subtract Equation 1 from Equation 2 to get

$$y_2 - y_1 = mx_2 - mx_1$$

  - Isolate m

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

  - And substitute m back in and solve for b:

$$b = y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1$$

Linear interpolation

- What you may have noticed by now is that this is a system of two linear equations in two unknowns:

$$\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

  – Recall that it is $mx_1 + b\cdot 1 = y_1$ and $mx_2 + b\cdot 1 = y_2$
- Interpolate (1.5, 7.2) and (3.3, 10.8)
  – In MATLAB, we would do the following:

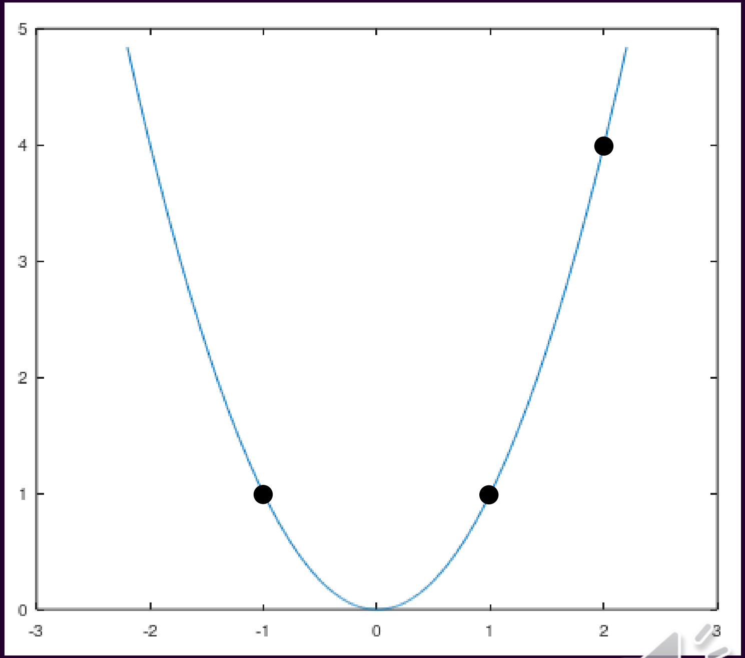
```
>> a = [1.5 1; 3.3 1] \ [7.2 10.8]'
     ans =
           2.0000
           4.2000
```

$$y = 2x + 4.2$$

Quadratic interpolation

- Given three points (x_1, y_1), (x_2, y_2), (x_3, y_3) with all three x values being different, can you find a straight line passing through them?
  - Of course not; for example (–1, 1), (1, 1), (2, 4)
    - The straight line $y = 1$ passes through the first two points, but not through the third
    - There is, however, a quadratic that passes through these three points:

$$y = x^2$$



Quadratic interpolation

- Given three points (x_1, y_1), (x_2, y_2), (x_3, y_3) with all three x values being different, can you find a quadratic passing through them?
  - Suppose the quadratic is of the form $ax^2 + bx + c$

$$y_1 = ax_1^2 + bx_1 + c$$

$$y_2 = ax_2^2 + bx_2 + c$$

$$y_3 = ax_3^2 + bx_3 + c$$

  - This is a system of three linear equations in three unknowns:

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Quadratic interpolation

- From here on in this course, we always write down an unknown polynomial as $a_2 x^2 + a_1 x + a_0$
  - The subscript of the coefficient equals the exponent of the term
  - Thus, our previous problem would be to solve:

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Linear algebra

- Does this system of three equations and three unknowns have a unique solution?

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}\begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

  – We can calculate the determinant:

$$\det\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} = (x_1 - x_2)(x_1 - x_3)(x_2 - x_3)$$

  – As long as all three *x* values are different,
    the determinant is non-zero

  – If any two *x*-values are equal, the determinant is zero

Quadratic interpolation

- Let us do an example:
  - Find the interpolating quadratic polynomial passing through

$$(0.4, 3.12), (1.2, 3.28), (3.5, 18)$$

  - We must solve

$$\begin{pmatrix} 0.4^2 & 0.4 & 1 \\ 1.2^2 & 1.2 & 1 \\ 3.5^2 & 3.5 & 1 \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 3.12 \\ 3.28 \\ 18.0 \end{pmatrix}$$

  - In MATLAB, we would do the following:

```
>> V = [0.4^2 0.4 1
        1.2^2 1.2 1
        3.5^2 3.5 1];
>> a = V \ [3.12 3.28 18.0]'
    a =
        2
       -3
        4
```

$$y = 2x^2 - 3x + 4$$

Quadratic interpolation

- Writing down these matrices in MATLAB can be error prone
  - The `vander` function in MATLAB generates this matrix

```
>> V = [0.4^2 0.4 1
        1.2^2 1.2 1
        3.5^2 3.5 1]
    V =
        0.1600    0.4000    1.0000
        1.4400    1.2000    1.0000
       12.2500    3.5000    1.0000
>> V = vander( [0.4 1.2 3.5] )
    V =
        0.1600    0.4000    1.0000
        1.4400    1.2000    1.0000
       12.2500    3.5000    1.0000
```

  - This matrix is called the *Vandermonde* matrix

Quadratic interpolation

- Remember, however, just because the determinant is non-zero does not mean we can find the solution numerically
  - Suppose the three points were 0, 0.001 and 100:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0.000001 & 0.001 & 1 \\ 10000 & 100 & 1 \end{pmatrix}$$

```
>> V = vander( [0 0.001 100]' )
       0              0              1
       0.000001       0.001          1
   10000            100              1
>> det( V )
    -9.9999
>> cond( V )
      14143693.10488148
```

Quadratic interpolation

- Equally spaced numbers result in a smaller condition number:

```
>> cond( vander( [0 1 2] ) )
     13.9125
>> cond( vander( [0 0.9 2] ) )
     14.0740
>> cond( vander( [0 1.1 2] ) )
     14.0770
>> cond( vander( [-1 0 1] ) )
      3.2255
```

  – Fortunately, most engineering data comes from periodically sampled sensors, so sampled data will, in general, see have *x*-values that are equally spaced
    - Actually, they will usually be *t*-values as they are samples in time

You don't have to know why or prove that equally-spaced points result in smaller condition numbers

General interpolation

Theorem

Given n points $(x_1, y_1), \ldots, (x_n, y_n)$ with all x-values being distinct, there exists a unique polynomial

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2x^2 + a_1x + a_0$$

of degree $n - 1$ that passes through all n points.

Proof:

The proof is constructive.

The system of linear equations is as follows:

$$\begin{pmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \cdots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{pmatrix} \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$

General interpolation

Now, normally, one cannot use the determinant to determine numerically if a matrix is invertible

However, with an algebraic matrix, it is possible to use induction to deduce that the determinant of the matrix is

$$\det\begin{pmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \cdots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{pmatrix} = \prod_{i=1}^{n}\prod_{j=i+1}^{n}\left(x_i - x_j\right)$$

That is, it is the product of all pairwise differences of the *x* values.

If all *x* values are different, all differences are non-zero

Thus the determinant is non-zero, and thus a unique solution exists. ■

🚫 You only need to be aware of the theorem; you don't need to know the proof.

General interpolation

- Once again, having *n* points where the *x*-values are equally spaced tends to have smaller condition numbers
  - Equally spaced *x*-values are not actually ideal, but the ideal points are beyond the scope of this point
  - Also, as mentioned before, these points tend to come from a sensor being periodically sampled

- For example:

```
>> cond( vander( [1 2 3 4] ) )
    1171.0
>> cond( vander( [1 1.1 3.9 4] ) )
    4932.3
>> cond( vander( [-1.5 -0.5 0.5 1.5] ) )
       9.1617
>> cond( vander( [-1.5 -0.80233 0.80233 1.5] ) )
       7.3919
```

Numeric error

- In the previous example, we discussed how failing to account for floating-point arithmetic may lead to errors in finding solutions to systems of linear equations
  - Suppose we want to interpolate the points

$$(0.00001532, 4.545), (2.523, 5.237)$$

  - We must therefore solve

$$\left(\begin{array}{ll:l} 0.00001532 & 1 & 4.545 \\ 2.523 & 1 & 5.237 \end{array}\right)$$

  - Using our six-digit floating-point representation
    - Without partial pivoting, the solution is $y = 4.545$
    - With partial pivoting, the solution is $y = 0.2743x + 4.545$

Numeric error

- You may ask why anyone would interpolate

  (0.00001532, 4.545), (2.523, 5.237)

  instead of

  (0.0, 4.545), (2.523, 5.237)

- Consider this program:

```
#include <iostream>

int main();

int main() {
    std::cout.precision( 16 );

    for ( double x{-1.0}; x < 0.05; x += 0.1 ) {
        std::cout << x << std::endl;
    }

    return 0;
}
```

Output:

```
-1
-0.9
-0.8
-0.7000000000000001
-0.6000000000000001
-0.5000000000000001
-0.4000000000000001
-0.3000000000000002
-0.2000000000000001
-0.1000000000000001
-1.387778780781446e-16
```

This is for information purposes only
- This is not on the examination

Numeric error

- We can even see the issues in MATLAB:

```
>> small = 2^-53 + 2^-55
    small = 1.387778780781446e-16
>> a = [small 1; 2.523 1] \ [4.545 5.237]'
    a =
        0.27428
        4.54500
>> A = [small 1 4.545; 2.523 1 5.237];
    A =
      -1.3878e-16   1     4.5450
       2.5230       1     5.2370
>> A(2,:) = A(2,:) - A(2,1)/A(1,1)*A(1,:)
    A =
      -1.3878e-16   1.0000       4.5450
       0.0000       1.8180e+16   8.2629e+16
>> a0 = A(2,3)/A(2,2)
    a0 =  4.5450
>> a1 = (A(1,3) - a0*A(1,2))/A(1,1)
    a1 = 0
```

$$\left(\begin{array}{cc|c} 1.3878\times10^{-16} & 1 & 4.545 \\ 2.523 & 1 & 5.237 \end{array}\right)$$

This is for information purposes only
- This is not on the examination

Shifting the x-values

- Recall also that points should be closer to zero
  - Assume we are interpolating

$$(x_1, y_1), (x_2, y_2), (x_3, y_3)$$

with $x_1 < x_2 < x_3$

- It makes more sense to interpolate the points

$$(x_1 - x_2, y_1), (0, y_2), (x_3 - x_2, y_3)$$

- If we wanted to evaluate the interpolating polynomial at x, we would evaluate the new polynomial at $x - x_2$
- The Vandermonde matrices are

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} \qquad \begin{pmatrix} (x_1 - x_2)^2 & x_1 - x_2 & 1 \\ 0 & 0 & 1 \\ (x_1 - x_2)^2 & x_3 - x_2 & 1 \end{pmatrix}$$

Shifting the *x*-values

- At this point, we are solving:

$$\begin{pmatrix} (x_1 - x_2)^2 & x_1 - x_2 & 1 \\ 0 & 0 & 1 \\ (x_3 - x_2)^2 & x_3 - x_2 & 1 \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

- We may immediately deduce that $a_0 = y_2$,
  so we are really only solving

$$\begin{pmatrix} (x_1 - x_2)^2 & x_1 - x_2 \\ (x_1 - x_2)^2 & x_3 - x_2 \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 - y_2 \\ y_3 - y_2 \end{pmatrix}$$

Shifting the x-values

- Therefore, suppose we are finding the interpolating quadratic of

$$(x_1, y_1), (x_2, y_2), (x_3, y_3)$$

- Find the interpolating quadratic of

$$(x_1 - x_2, y_1), (0, y_2), (x_3 - x_2, y_3)$$

  - This requires us to find the interpolating polynomial

$$p(x) = a_2 x^2 + a_1 x + y_2$$

  - This requires us to solve:

$$\begin{pmatrix} (x_1 - x_2)^2 & x_1 - x_2 \\ (x_3 - x_2)^2 & x_3 - x_2 \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 - y_2 \\ y_3 - y_2 \end{pmatrix}$$

- To evaluate the interpolating polynomial at $x_1 < x < x_3$, we calculate

$$p(x - x_2)$$

Shifting the x-values

- Now, if the three points are equally spaced by h:

$$(x_2 - h, y_1), (x_2, y_2), (x_2 + h, y_3)$$

- Find the interpolating quadratic of

$$(-h, y_1), (0, y_2), (h, y_3)$$

  - Solving

$$\begin{pmatrix} h^2 & -h \\ h^2 & h \end{pmatrix}\begin{pmatrix} a_2 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 - y_2 \\ y_3 - y_2 \end{pmatrix}$$

- This means that

$$a_2 = \frac{y_3 - 2y_2 + y_1}{2h^2}$$

$$a_1 = \frac{y_3 - y_1}{2h}$$

$$a_0 = y_2$$

Shifting the *x*-values

- We can now implement this in C++:

```
double quad_interp( double x2, double h, double y[3],
                    double x ) {
    double a1{ (y[2] - y[0])/(2.0*h) };
    double a2{ (y[2] - 2.0*y[1] + y[0])/(2.0*h*h) };

    return a2*(x - x2)*(x - x2) + a1*(x - x2) + y[1];
}
```

This is for information purposes only
- This is not on the examination

Shifting the *x*-values

- We can also implement this in MATLAB:

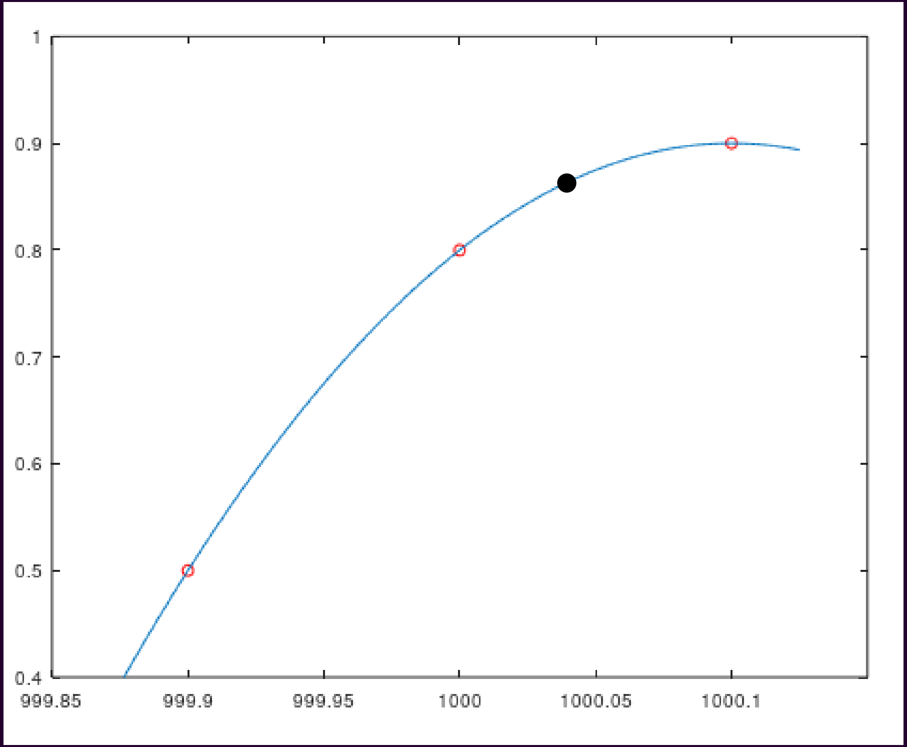
```
function [y] = quad_interp( x2, h, ys, x )
    a1 = (ys(3) - ys(1))/(2.0*h);
    a2 = (ys(3) - 2.0*ys(2) + ys(1))/(2.0*h*h);

    y = a2*(x - x2)*(x - x2) + a1*(x - x2) + ys(2);
end
```

Shifting the *x*-values

- Let us look at an example:

  (999.9, 0.5), (1000.0, 0.8), (1000.1, 0.9)

- To find the interpolating polynomial at $x = 1000.04$
  - The exact answer is $y = 0.864$



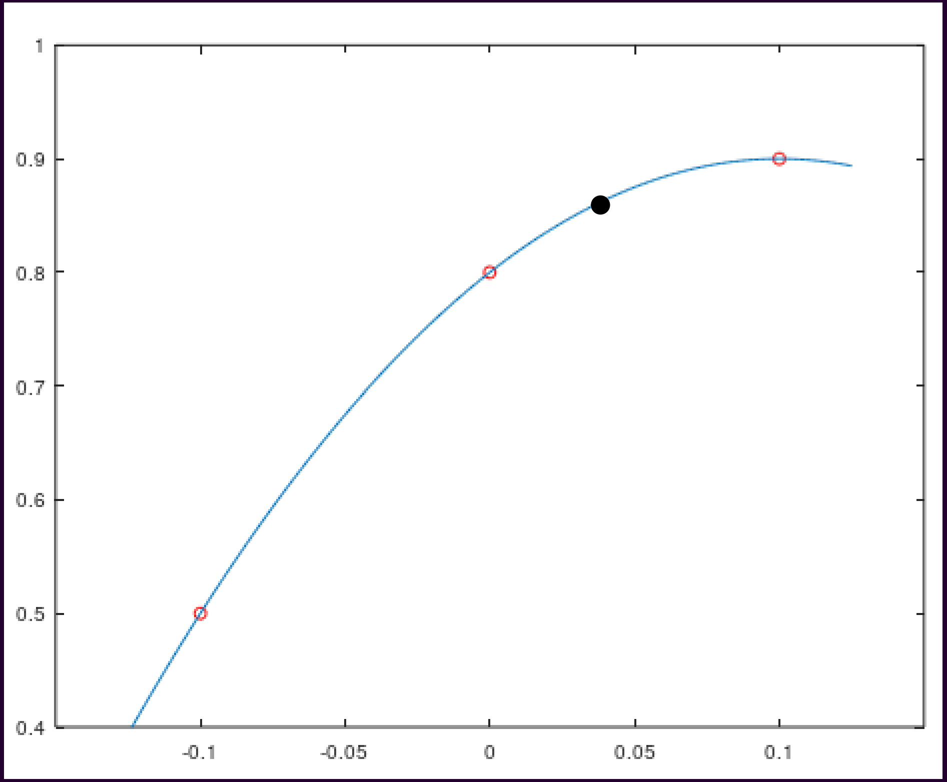
Shifting the *x*-values

- We can solve the system of linear equations

$$\begin{pmatrix} 999.9^2 & 999.9 & 1 \\ 1000.0^2 & 1000.0 & 1 \\ 1000.1^2 & 1000.1 & 1 \end{pmatrix}\begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.8 \\ 0.9 \end{pmatrix}$$

- Alternatively, we can use the technique we just described

  (–0.1, 0.5), (0, 0.8), (0.1, 0.9)



Shifting the *x*-values

- In MATLAB:

```
>> a = vander( [999.9 1000 1000.1] ) \ [0.5 0.8 0.9]'
   a =
          -9.999999962993799e+00
           2.000199992597534e+04
          -1.000199916298154e+07

>> a(1)*1000.04^2 + a(2)*1000.04 + a(3)
    ans = 0.8640000019222498

>> quad_interp( 1000.0, 0.1, [0.5 0.8 0.9], 1000.04 )
    ans = 0.8639999999999564
```

- The relative error 44056 times larger

Numeric error

- Recall that the condition number tells you how much an error may be magnified

```
>> cond( vander( [999.9 1000 1000.1] ) )
    212132567547302.3

>> cond( vander( [-0.1 0.1] ) )
    9.999999999999996
```

  - The first is larger by a factor of 21 trillion

Other issues

- Other issues:
  - Higher-order interpolating polynomials result in higher error
    - Consequently, we will restrict ourselves to lower degree interpolating polynomials
  - If we are interpolating the *x*-values

    $$x_1 < x_2 < \cdots < x_n$$

    it is generally only safe to evaluate the interpolating polynomial for values of $x_1 < x < x_n$
    - If evaluating the interpolating polynomial for values of x either

      $$x < x_1 \text{ or } x > x_n$$

      it is called *extrapolation*, and the error increases rapidly

Further improvements

- Incidentally, here is an improvement to our program:
  - Map the *x*-values so that `x2 - h`, `x2` and `x2 + h` map to `-1`, `0` and `1`, respectively

```
double quad_interp( double x2, double h, double y[3],
                    double x ) {
    double a1{ y[2] - y[0] };
    double a2{ y[2] - 2.0*y[1] + y[0] };

    double dx{ (x - x2)/h };

    return (a2*dx + a1)*dx/2.0 + y[1];
}
```

This is for information purposes only
- This is not on the examination

Further improvements

- Turning it into a class is even more efficient:

```
class Quad_interp {
    public:
        Quad_interp( double x2, double h, double y[3] );
        double eval( double x ) const;
    private:
        double a_[3];
        double x2_;
        double h_;
};

Quad_interp::Quad_interp(double x2, double h, double y[3] ) :
a_{ y[1], (y[2] - y[0])/2.0, (y[2] - 2.0*y[1] + y[0])/2.0 },
x2_{ x2 },
h_{ h } {
    // Empty constructor
}
```

This is for information purposes only
- This is not on the examination

Further improvements

- The `eval` member function only uses the stored coefficients

```
double Quad_interp::eval( double x ) const {
    x = (x - x2_)/h_;
    return (a_[2]*x + a_[1])*x + a_[0];
}
```

This is for information purposes only
- This is not on the examination

Summary

- Following this topic, you now
  - Understand how to find interpolating polynomials using linear algebra
  - Know the Vandermonde matrix
    - You also understand why the last column of the matrix is all ones in the Vandermonde matrix
  - Revisited the justification for using partial pivoting with Gaussian elimination
  - Understand that large *x*-values may result in large increases in the error
  - Are aware of further issues with interpolating polynomials
  - Have seen some further improvements to our functions

References

[1] https://en.wikipedia.org/wiki/Gaussian_elimination
[2] https://en.wikipedia.org/wiki/Pivot_element
[3] https://en.wikipedia.org/wiki/Jacobi_method
[4] https://en.wikipedia.org/wiki/Condition_number

Acknowledgments

Brian Nguyen for spotting an error in the formula for backward substitution on p.18.

Tazik Shahjahan for pointing out typos.

Aaron Hong for noting the indices on one term of a polynomial were wrong.

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