Derivatives of Unequally Spaced Points

By
Namir Clement Shammas

Contents

Introduction

Numerical analysis offers methods for interpolation of unequally spaced data. The Lagrange and Newton divided-difference methods are good examples. When it comes to calculating derivatives, numerical analysis offers plenty of algorithms (of varying precisions) to estimate the derivative of functions. Of course, it makes sense to estimate the derivatives of given functions. In these cases, the equations use small increments, usually called h, to estimate the derivatives.

When the underlying function is not known and instead, we have unequally spaced x values and their corresponding y values, we can still estimate the derivatives. Such methods are good, for example, in estimating the chemical reaction rates which are the rate of the change of the concentration of chemicals with time. Often the concentrations are measured at unequally spaced time values.

I found on the Internet many publications that approximate the first and second derivatives based on quadratic Lagrange interpolation polynomials. These methods use three sets of (x, y) data and calculate the slope at a value xint. This study reviews these algorithms and presents additional ones that use derivative approximations based on cubic and quartic Lagrange interpolation polynomials.

Existing Published Derivatives

Given three (x, y) points equation (1) shows the estimate of the first derivative assuming the points are not equally spaced:

$$f'(x) = f(x_1)\frac{2x-x_2-x_3}{(x_1-x_2)(x_1-x_3)} + f(x_2)\frac{2x-x_1-x_3}{(x_2-x_1)(x_2-x_3)} + f(x_3)\frac{2x-x_1-x_2}{(x_3-x_1)(x_3-x_2)} \quad (1)$$

Equation (2) shows the approximation of the second derivative:

$$f''(x) = 2 * [\frac{f(x_1)}{(x_1-x_2)(x_1-x_3)} + \frac{f(x_2)}{(x_2-x_1)(x_2-x_3)} + \frac{f(x_3)}{(x_3-x_1)(x_3-x_2)}] \quad (2)$$

The above equations are based on Lagrangian polynomials. In the case of Newton Divided-Difference polynomials we have equation (3):

$$f'(x) = f[x_1, x_2] + f[x_1, x_2, x_3](2x - x_1 - x_2) \quad (3)$$

Equation (4) shows the approximation of the second derivative:

$$f''(x) = 2 * f[x_1, x_2, x_3] \quad (4)$$

Additional Approximations

This section presents additional approximations that use more points and also offers approximations for higher derivatives. For the sake of conveniently dealing with

more elaborate equations, I will present the derivatives as algorithms instead of equations.

Approximation Using Cubic Lagrange Polynomials

The equation for the first derivative using cubic Lagrange polynomials involves more calculations. Equation (5) shows the first term in calculating the first derivative.

$$f'(x) = f(x_1)\frac{3x^2-2x(x_2+x_3+x_4)+x_2x_3+x_2x_4+x_3x_4}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} + \cdots \quad (5)$$

Given four (x, y) data points that are not equally spaced, the next algorithm presents the first derivative approximation:

```
Given points (x1,y1) thought (x4,y4) we calculate the first derivative at x as:

T(1) = 3*x^2 - 2*x*(x4 + x2 + x3) + x4*x2 + x4*x3 + x2*x3
T(2) = 3*x^2 - 2*x*(x1 + x3 + x4) + x1*x3 + x1*x4 + x3*x4
T(3) = 3*x^2 - 2*x*(x1 + x2 + x4) + x1*x2 + x1*x4 + x2*x4
T(4) = 3*x^2 - 2*x*(x1 + x2 + x3) + x1*x2 + x1*x3 + x2*x3
D = 0
For i=1 to 4
  U(i) = y(i)
  For j=1 to 4
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + T(i) * U(i)
Next i
Return D
```

Algorithm 1. The first Derivative Approximation based on a cubic Lagrange polynomial.

☛ In Algorithm 1 I use x(i) and x(j) to mean any one of x1 through x4 accessed using variables i and j, respectively. The same holds for y(i) and the y1 through y4. I will be using this notation for the other algorithms too.

The equation for the second derivative using cubic Lagrange polynomials involves a bit less calculations. Equation (6) shows the first term in calculating the second derivative.

$$f''(x) = f(x_1)\frac{6x-2(x_2+x_3+x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} + \cdots \quad (6)$$

Given four (x, y) data points that are not equally spaced, the next algorithm presents the second derivative approximation:


```
Given points (x1,y1) though (x4,y4) we calculate the second derivative at x as:

T(1) = 6*x - 2*(x4 + x2 + x3)
T(2) = 6*x - 2*(x1 + x3 + x4)
T(3) = 6*x - 2*(x1 + x2 + x4)
T(4) = 6*x - 2*(x1 + x2 + x3)
D = 0
For i=1 to 4
  U(i) = y(i)
  For j=1 to 4
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + T(i) * U(i)
Next i
Return D
```

Algorithm 2. The second Derivative Approximation based on a cubic Lagrange polynomial.

The equation for the third derivative using cubic Lagrange polynomials requires less calculations. Equation (5) shows the first term in calculating the third derivative.

$$f^3(x) = f(x_1)\frac{6}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} + \cdots \quad (7)$$

Given four (x, y) data points that are not equally spaced, the next algorithm presents the third derivative approximation:

```
Given points (x1,y1) though (x4,y4) we calculate the third derivative at x as:

D = 0
For i=1 to 4
  U(i) = y(i)
  For j=1 to 4
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + 6 * U(i)
Next i
Return D
```

Algorithm 3. The third Derivative Approximation based on a cubic Lagrange polynomial.

Approximation Using Cubic Newton Polynomials

The equation for the first derivative using cubic Lagrange polynomials involves more calculations. Equation (8) shows the first term in calculating the first derivative.

$$f'(x) = f[x_1, x_2] + f[x_1, x_2, x_3](2x - x_1 - x_2) + f[x_1, x_2, x_3, x_4][3x^2 - 2x(x_1 + x_2 + x_3) +$$

$$(x_1x_2 + x_1x_3 + x_2x_3)] \quad (8)$$

Given four (x, y) data points that are not equally spaced, the next algorithm presents the first derivative approximation:

```
Given points (x1,y1) thought (x4,y4) we calculate the first derivative at x as:

T1 = 2*x – x1 – x2
T2 = 3*x^2 - 2*x*(x1 + x2 + x3) + x1*x2 + x1*x3 + x2*x3
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d31 = (d22 - d21)/(x4-x1)
D = d11 + d21*T1 + d31*T2
Return D
```

Algorithm 3. The first Derivative Approximation based on a cubic Newton polynomial.

☛ In Algorithm 1 I use x(i) and x(j) to mean any one of x1 through x4 accessed using variables i and j, respectively. The same holds for y(i) and the y1 through y4. I will be using this notation for the other algorithms too.

The equation for the second derivative using cubic Lagrange polynomials involves a bit less calculations. Equation (9) shows the first term in calculating the second derivative.

$$f''(x) = 2f[x_1, x_2, x_3] + f[x_1, x_2, x_3, x_4][6x - 2(x_1 + x_2 + x_3)] \quad (9)$$

Given four (x, y) data points that are not equally spaced, the next algorithm presents the second derivative approximation:

```
Given points (x1,y1) though (x4,y4) we calculate the second derivative at x as:

T1 = 2
T2 = 6*x - 2*(x1 + x2 + x3)
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d31 = (d22 - d21)/(x4-x1)
D = d21*T1 + d31*T2
Return D
```


Algorithm 4. The second Derivative Approximation based on a cubic Newton polynomial.

The equation for the third derivative using cubic Lagrange polynomials requires less calculations. Equation (10) shows the first term in calculating the third derivative.

$$f^3(x) = 6f[x_1, x_2, x_3, x_4] \quad (10)$$

Given four (x, y) data points that are not equally spaced, the next algorithm presents the third derivative approximation:

```
Given points (x1,y1) though (x4,y4) we calculate the third derivative at x as:

T2 = 6
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d31 = (d22 - d21)/(x4-x1)
Return d31*T2
```

Algorithm 5. The third Derivative Approximation based on a cubic Newton polynomial.

Approximation Using Quartic Lagrange Polynomials

The equation for the first derivative using quartic Lagrange polynomials involves more calculations. Equation (11) shows the first term in calculating the first derivative.

$$f'(x) = f(x_1)\frac{4x^3 - 3Ax^2 + 2Bx - C}{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)(x_1 - x_5)} + \cdots \quad (11)$$

Where A = $x_2 + x_3 + x_4 + x_5$, B = $x_2*x_3 + x_2*x_4 + x_2*x_5 + x_3*x_4 + x_3*x_5 + x_4*x_5$, and C = $x_2*x_3*x_4 + x_2*x_3*x_5 + x_2*x_4*x_5 + x_3*x_4*x_5$.

Given five (x, y) data points that are not equally spaced, the next algorithm presents the first derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the first derivative at x as:

T(1) = 4*x^3 - 3*x^2*(x2 + x3 + x4 + x5) + 2*x*(x2*x3 + x2*x4 + x2*x5 + x3*x4 + x3*x5
+ x4*x5) - (x2*x3*x4 + x2*x3*x5 + x2*x4*x5 + x3*x4*x5)
T(2) = 4*x^3 - 3*x^2*(x1 + x3 + x4 + x5) + 2*x*(x1*x3 + x1*x4 + x1*x5 + x3*x4 + x3*x5
+ x4*x5)-(x1*x3*x4 + x1*x3*x5 + x1*x4*x5 + x3*x4*x5)
```



```
T(3) = 4*x^3 - 3*x^2*(x1 + x2 + x4 + x5) +  2*x*(x1*x4+ x1*x5 + x4*x5 + x1*x2 + x2*x5
+ x2*x4)- (x1*x2*x4  + x1*x2*x5 + x1*x4*x5 + x2*x4*x5)
T(4) = 4*x^3 - 3*x^2*(x1 + x2 + x3 + x5) + 2*x*(x1*x2 + x1*x3 + x1*x5 + x2*x3 + x2*x5
+ x3*x5)- (x1*x2*x3 + x1*x2*x5 + x1*x3*x5 + x2*x3*x5)
T(5) = 4*x^3 - 3*x^2*(x1 + x2 + x3 + x4) + 2*x*(x1*x2 + x1*x3 + x1*x4 + x2*x3 + x2*x4
+ x3*x4) - (x1*x2*x3 + x1*x2*x4 + x1*x3*x4 + x2*x3*x4)
D = 0
For i=1 to 5
  U(i) = y(i)
  For j=1 to 5
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + T(i) * U(i)
Next i
Return D
```

Algorithm 6. The first Derivative Approximation based on a quartic Lagrange polynomial.

The equation for the second derivative using quartic Lagrange polynomials involves more calculations. Equation (12) shows the first term in calculating the second derivative.

$$f''(x) = f(x_1)\frac{12x^2-6Ax+2\mathrm{B}}{(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)} + \cdots \qquad (12)$$

Given five (x, y) data points that are not equally spaced, the next algorithm presents the second derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the second derivative at x as:

T(1) = 12*x^2 - 6*x*(x2 + x3 + x4 + x5) + 2*(x2*x3 + x2*x4 + x2*x5 + x3*x4 + x3*x5 +
x4*x5)
T(2) = 12*x^2 - 6*x*(x1 + x3 + x4 + x5) + 2*(x1*x3 + x1*x4 + x1*x5 + x3*x4 + x3*x5 +
x4*x5)
T(3) = 12*x^2 - 6*x*(x1 + x2 + x4 + x5) +  2*(x1*x4+ x1*x5 + x4*x5 + x1*x2 + x2*x5 +
x2*x4)
T(4) = 12*x^2 - 6*x*(x1 + x2 + x3 + x5) + 2*(x1*x2 + x1*x3 + x1*x5 + x2*x3 + x2*x5 +
x3*x5)
T(5) = 12*x^2 - 6*x*(x1 + x2 + x3 + x4) + 2*(x1*x2 + x1*x3 + x1*x4 + x2*x3 + x2*x4 +
x3*x4)
D = 0
For i=1 to 5
  U(i) = y(i)
  For j=1 to 5
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + T(i) * U(i)
Next i
Return D
```

Algorithm 7. The second Derivative Approximation based on a quartic Lagrange polynomial.

The equation for the third derivative using quartic Lagrange polynomials involves more calculations. Equation (13) shows the first term in calculating the third derivative.

$$f^3(x) = f(x_1)\frac{24x-6A}{(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)} + \cdots \quad (13)$$

Given five (x, y) data points that are not equally spaced, the next algorithm presents the third derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the third derivative at x as:

T(1) = 24*x - 6*(x2 + x3 + x4 + x5)
T(2) = 24*x - 6*(x1 + x3 + x4 + x5)
T(3) = 24*x - 6*(x1 + x2 + x4 + x5)
T(4) = 24*x - 6*(x1 + x2 + x3 + x5)
T(5) = 24*x - 6*(x1 + x2 + x3 + x4)
D = 0
For i=1 to 5
  U(i) = y(i)
  For j=1 to 5
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + T(i) * U(i)
Next i
Return D
```

Algorithm 8. The third Derivative Approximation based on a quartic Lagrange polynomial.

The equation for the fourth derivative using quartic Lagrange polynomials involves more calculations. Equation (14) shows the first term in calculating the fourth derivative.

$$f^4(x) = f(x_1)\frac{24}{(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)} + \cdots \quad (14)$$

Given five (x, y) data points that are not equally spaced, the next algorithm presents the fourth derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the fourth derivative at x as:

D = 0
For i=1 to 5
  U(i) = y(i)
  For j=1 to 5
    U(i) = U(i) / (x(i) - x(j))
  Next j
  D = D + 24 * U(i)
```



```
Next i
return D
```

Algorithm 9. The fourth Derivative Approximation based on a quartic Lagrange polynomial.

Approximation Using Quartic Newton Polynomials

The equation for the first derivative using quartic Newton polynomials involves more calculations.

Given five (x, y) data points that are not equally spaced, the next algorithm presents the first derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the first derivative at x as:

T1 = 2*x – x1 – x2
T2 = 3*x^2 - 2*x*(x1 + x2 + x3) + x1*x2 + x1*x3 + x2*x3
T3 = 4*x^3-3*x^2*(x1+x2+x3+x4) + 2*x*(x1*x2+x1*x3+x1*x4+x2*x3+x2*x4+x3*x4) -
x*(x1*x2*x3+x1*x2*x4+x1*x3*x4+x2*x3*x4)+x1*x2*x3*x4
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d14 = (y5-y4)/(x5-x4)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d23 = (d14-d13)/(x5-x3)
d31 = (d22 - d21)/(x4-x1)
d32 = (d23 - d22)/(x5-x2)
d41 = (d32 - d31)/(x5-x1)
D = d11 + d21*T1 + d31*T2 + d41*T3
Return D
```

Algorithm 9. The first Derivative Approximation based on a quartic Lagrange polynomial.

The equation for the second derivative using quartic Lagrange polynomials involves more calculations.

Given five (x, y) data points that are not equally spaced, the next algorithm presents the second derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the second derivative at x as:

T1 = 2
T2 = 6*x - 2*(x1 + x2 + x3)
T3 = 12*x^2-6*x*(x1+x2+x3+x4) + 2*(x1*x2+x1*x3+x1*x4+x2*x3+x2*x4+x3*x4)
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d14 = (y5-y4)/(x5-x4)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d23 = (d14-d13)/(x5-x3)
```



```
d31 = (d22 - d21)/(x4-x1)
d32 = (d23 - d22)/(x5-x2)
d41 = (d32 - d31_/(x5-x1)
D = d21*T1 + d31*T2 + d41*T3
Return D
```

Algorithm 10. The second Derivative Approximation based on a quartic Newton polynomial.

The equation for the third derivative using quartic Lagrange polynomials involves more calculations.

Given five (x, y) data points that are not equally spaced, the next algorithm presents the third derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the third derivative at x as:

T1 = 6
T2 = 24*x-6* (x1+x2+x3+x4)
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d14 = (y5-y4)/(x5-x4)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d23 = (d14-d13)/(x5-x3)
d31 = (d22 - d21)/(x4-x1)
d32 = (d23 - d22)/(x5-x2)
d41 = (d32 - d31_/(x5-x1)
D = d31*T1 + d41*T2
Return D
```

Algorithm 11. The third Derivative Approximation based on a quartic Newton polynomial.

The equation for the fourth derivative using quartic Lagrange polynomials involves more calculations.

Given five (x, y) data points that are not equally spaced, the next algorithm presents the fourth derivative approximation:

```
Given points (x1,y1) though (x5,y5) we calculate the fourth derivative at x as:

T1 = 24
d11 = (y2-y1)/(x2-x1)
d12 = (y3-y2)/(x3-x2)
d13 = (y4-y3)/(x4-x3)
d14 = (y5-y4)/(x5-x4)
d21 = (d12-d11)/(x3-x1)
d22 = (d13-d12)/(x4-x2)
d23 = (d14-d13)/(x5-x3)
d31 = (d22 - d21)/(x4-x1)
```



```
d32 = (d23 - d22)/(x5-x2)
d41 = (d32 - d31_/(x5-x1)
return d41*T1
```

Algorithm 11. The fourth Derivative Approximation based on a quartic Lagrange polynomial.

Code Implementation for the Lagrangian Version

This section presents implementation of the derivative functions for 3, 4, and 5 points in Excel VB, MATLAB, and Python.

Implementation in Excel VBA

This section presents the implementation of the algorithms presented earlier in Excel VBA:

```
Function deriv1_3(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv1_3 = 1E+99
  N = UBound(Xarr)
  If N < 3 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 2
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 2 Then nStartIdx = N - 2
  End If


  Sum = 0
  For I = nStartIdx To nStartIdx + 2
    T1 = 0
    T2 = 1
    For J = nStartIdx To nStartIdx + 2
      If I <> J Then
        T1 = T1 + X - Xarr(J)
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv1_3 = Sum
End Function
```



```
Function deriv2_3(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv2_3 = 1E+99
  N = UBound(Xarr)
  If N < 3 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 2
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 2 Then nStartIdx = N - 2
  End If

  Sum = 0
  For I = nStartIdx To nStartIdx + 2
    T1 = 2
    T2 = 1
    For J = nStartIdx To nStartIdx + 2
      If I <> J Then
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If

    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv2_3 = Sum
End Function


Function deriv1_4(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv1_4 = 1E+99
  N = UBound(Xarr)
  If N < 4 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 3
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
```



```
    If nStartIdx > N - 3 Then nStartIdx = N - 3
  End If


  Sum = 0

  For I = nStartIdx To nStartIdx + 3
    T1 = 3 * X ^ 2
    T2 = 1
    For K = nStartIdx To nStartIdx + 2
      If K <> I Then
        For J = K + 1 To nStartIdx + 3
          If K <> J And I <> J Then
            T1 = T1 + Xarr(K) * Xarr(J)
          End If
        Next J
      End If
    Next K
    For J = nStartIdx To nStartIdx + 3
      If I <> J Then
        T1 = T1 - 2 * X * Xarr(J)
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv1_4 = Sum
End Function

Function deriv2_4(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv2_4 = 1E+99
  N = UBound(Xarr)
  If N < 4 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 3
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 3 Then nStartIdx = N - 3
  End If

  Sum = 0
  For I = nStartIdx To nStartIdx + 3
    T1 = 6 * X
    T2 = 1
     For J = nStartIdx To nStartIdx + 3
      If I <> J Then
        T1 = T1 - 2 * Xarr(J)
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
```



```
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv2_4 = Sum
End Function

Function deriv3_4(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv3_4 = 1E+99
  N = UBound(Xarr)
  If N < 4 Or N <> UBound(Yarr) Then Exit Function


  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 3
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 3 Then nStartIdx = N - 3
  End If

  Sum = 0
  For I = nStartIdx To nStartIdx + 3
    T1 = 6
    T2 = 1
     For J = nStartIdx To nStartIdx + 3
      If I <> J Then
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv3_4 = Sum
End Function


Function deriv1_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim nStartIdx As Integer


  deriv1_5 = 1E+99
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 4
```

```
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)

  Sum = 0

  For I = nStartIdx To nStartIdx + 4
    J = I - nStartIdx + 1
    Select Case J
      Case 1:
        T1 = 4 * X ^ 3 - 3 * X ^ 2 * (x2 + x3 + x4 + x5)
        T1 = T1 + 2 * X * (x2 * x3 + x2 * x4 + x2 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
        T1 = T1 - (x2 * x3 * x4 + x2 * x3 * x5 + x2 * x4 * x5 + x3 * x4 * x5)
      Case 2:
        T1 = 4 * X ^ 3 - 3 * X ^ 2 * (x1 + x3 + x4 + x5)
        T1 = T1 + 2 * X * (x1 * x3 + x1 * x4 + x1 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
        T1 = T1 - (x1 * x3 * x4 + x1 * x3 * x5 + x1 * x4 * x5 + x3 * x4 * x5)
      Case 3:
        T1 = 4 * X ^ 3 - 3 * X ^ 2 * (x1 + x2 + x4 + x5)
        T1 = T1 + 2 * X * (x1 * x4 + x1 * x5 + x4 * x5 + x1 * x2 + x2 * x5 + x2 * x4)
        T1 = T1 - (x1 * x2 * x4 + x1 * x2 * x5 + x1 * x4 * x5 + x2 * x4 * x5)
      Case 4:
        T1 = 4 * X ^ 3 - 3 * X ^ 2 * (x1 + x2 + x3 + x5)
        T1 = T1 + 2 * X * (x1 * x2 + x1 * x3 + x1 * x5 + x2 * x3 + x2 * x5 + x3 * x5)
        T1 = T1 - (x1 * x2 * x3 + x1 * x2 * x5 + x1 * x3 * x5 + x2 * x3 * x5)
      Case 5:
        T1 = 4 * X ^ 3 - 3 * X ^ 2 * (x1 + x2 + x3 + x4)
        T1 = T1 + 2 * X * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4)
        T1 = T1 - (x1 * x2 * x3 + x1 * x2 * x4 + x1 * x3 * x4 + x2 * x3 * x4)
    End Select
    T2 = 1
    For J = nStartIdx To nStartIdx + 4
      If I <> J Then
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv1_5 = Sum
End Function

Function deriv2_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim nStartIdx As Integer

  deriv2_5 = 1E+99
  N = UBound(Xarr)
```



```
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 4
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)

  Sum = 0

  For I = nStartIdx To nStartIdx + 4
    J = I - nStartIdx + 1
    Select Case J
      Case 1:
        T1 = 12 * X ^ 2 - 6 * X * (x2 + x3 + x4 + x5)
        T1 = T1 + 2 * (x2 * x3 + x2 * x4 + x2 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
      Case 2:
        T1 = 12 * X ^ 2 - 6 * X * (x1 + x3 + x4 + x5)
        T1 = T1 + 2 * (x1 * x3 + x1 * x4 + x1 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
      Case 3:
        T1 = 12 * X ^ 2 - 6 * X * (x1 + x2 + x4 + x5)
        T1 = T1 + 2 * (x1 * x4 + x1 * x5 + x4 * x5 + x1 * x2 + x2 * x5 + x2 * x4)
      Case 4:
        T1 = 12 * X ^ 2 - 6 * X * (x1 + x2 + x3 + x5)
        T1 = T1 + 2 * (x1 * x2 + x1 * x3 + x1 * x5 + x2 * x3 + x2 * x5 + x3 * x5)
      Case 5:
        T1 = 12 * X ^ 2 - 6 * X * (x1 + x2 + x3 + x4)
        T1 = T1 + 2 * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4)
    End Select
    T2 = 1
    For J = nStartIdx To nStartIdx + 4
      If I <> J Then
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv2_5 = Sum
End Function

Function deriv3_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim nStartIdx As Integer

  deriv3_5 = 1E+99
```



```
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 4
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)

  Sum = 0

  For I = nStartIdx To nStartIdx + 4
    J = I - nStartIdx + 1
    Select Case J
      Case 1:
        T1 = 24 * X - 6 * (x2 + x3 + x4 + x5)
      Case 2:
        T1 = 24 * X - 6 * (x1 + x3 + x4 + x5)
      Case 3:
        T1 = 24 * X - 6 * (x1 + x2 + x4 + x5)
      Case 4:
        T1 = 24 * X - 6 * (x1 + x2 + x3 + x5)
       Case 5:
        T1 = 24 * X - 6 * (x1 + x2 + x3 + x4)
    End Select
    T2 = 1
    For J = nStartIdx To nStartIdx + 4
      If I <> J Then
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv3_5 = Sum
End Function

Function deriv4_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal X As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim Sum As Double, T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv4_5 = 1E+99
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If X < Xarr(1) Then
    nStartIdx = 1
```



```
  ElseIf X > Xarr(N) Then
    nStartIdx = N - 4
  Else
    For I = 1 To N - 1
      If X > Xarr(I) And X < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStartIdx too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  Sum = 0
  For I = nStartIdx To nStartIdx + 4
    T1 = 24
    T2 = 1
    For J = nStartIdx To nStartIdx + 4
      If I <> J Then
        T2 = T2 * (Xarr(I) - Xarr(J))
      End If
    Next J
    Sum = Sum + Yarr(I) * T1 / T2
  Next I
  deriv4_5 = Sum
End Function
```

Each of the above functions have the same parameter lists, which are:

- The reference array Xarr() that passes the elements of array X. You can have more array elements than you need. The function returns 1E99 if the array size is insufficient.
- The reference array Yarr() that passes the elements of array Y. You can have more array elements than you need. The function returns 1E99 if the array sizes of arrays Yarr() and Xarr() are not equal.
- The parameter X is the value at which a derivative is estimated.

The functions compare the value of X with the first and last elements in array Xarr to determine the first of the index values to use in calculating the derivative. If that first index is too close to the last element of array Xarr, the functions decrease the value of the starting index by an appropriate value. By contrast, if X is less than the first element in array Xarr, the functions set the starting index as the first array index.

The next two sections present the translation of the above VBA code into MATLAB and Python. The comments I made about the VBA functions applies equally to the MATLAB and Python code.

Implementation in MATLAB

This section presents the implementation of the algorithms presented earlier in MATLAB:

```
function d = deriv1_3(xarr, yarr, x)
%DERIV1_3 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 3 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 2;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 2, nStartIdx = n - 2; end
  end

  zsum = 0;
  for i = nStartIdx:nStartIdx + 2
    t1 = 0;
    t2 = 1;
    for j = nStartIdx:nStartIdx + 2
      if i ~= j
        t1 = t1 + x - xarr(j);
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end

function d = deriv2_3(xarr, yarr, x)
%DERIV2_3 Summary of this function goes here
%   Detailed explanation goes here
  d=1e+99;
  n = length(xarr);
  if n < 3 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 2;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
```



```
    if nStartIdx > n - 2,  nStartIdx = n - 2; end
  end

  zsum = 0
  for i = nStartIdx:nStartIdx + 2
    t1 = 2;
    t2 = 1;
    for j = nStartIdx:nStartIdx + 2
      if i ~= j
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end

function d = deriv1_4(xarr, yarr, x)
%DERIV1_4 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 4 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 3;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3,  nStartIdx = n - 3; end
  end

  zsum = 0;
  for i = nStartIdx:nStartIdx + 3
    t1 = 3 * x ^ 2;
    t2 = 1;
    for k = nStartIdx:nStartIdx + 2
      if k ~= i
        for j = k + 1:nStartIdx + 3
          if k ~= j && i ~= j
            t1 = t1 + xarr(k) * xarr(j);
          end
        end
      end
    end
    for j = nStartIdx:nStartIdx + 3
      if i ~= j
        t1 = t1 - 2 * x * xarr(j);
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end
```



```
function d = deriv2_4(xarr, yarr, x)
%DERIV2_4 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 4 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 3;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3, nStartIdx = n - 3; end
  end

  zsum = 0;
  for i = nStartIdx:nStartIdx + 3
    t1 = 6 * x;
    t2 = 1;
    for j = nStartIdx:nStartIdx + 3
      if i ~= j
        t1 = t1 - 2 * xarr(j);
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end

function d = deriv3_4(xarr, yarr, x)
%DERIV3_4 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 4 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 3;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3,  nStartIdx = n - 3; end
  end

  zsum = 0;
  for i = nStartIdx:nStartIdx + 3
    t1 = 6;
```



```
    t2 = 1;
    for j = nStartIdx:nStartIdx + 3
      if i ~= j
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end

function d = deriv1_5(xarr, yarr, x)
%DERIV1_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  x1 = xarr(nStartIdx);
  x2 = xarr(nStartIdx + 1);
  x3 = xarr(nStartIdx + 2);
  x4 = xarr(nStartIdx + 3);
  x5 = xarr(nStartIdx + 4);

  zsum = 0;

  for i = nStartIdx:nStartIdx + 4
    j = i - nStartIdx + 1;
    switch(j)
      case 1
        t1 = 4 * x ^ 3 - 3 * x ^ 2 * (x2 + x3 + x4 + x5);
        t1 = t1 + 2 * x * (x2 * x3 + x2 * x4 + x2 * x5 + x3 * x4 + x3 * x5 + x4 * x5);
        t1 = t1 - (x2 * x3 * x4 + x2 * x3 * x5 + x2 * x4 * x5 + x3 * x4 * x5);
      case 2
        t1 = 4 * x ^ 3 - 3 * x ^ 2 * (x1 + x3 + x4 + x5);
        t1 = t1 + 2 * x * (x1 * x3 + x1 * x4 + x1 * x5 + x3 * x4 + x3 * x5 + x4 * x5);
        t1 = t1 - (x1 * x3 * x4 + x1 * x3 * x5 + x1 * x4 * x5 + x3 * x4 * x5);
      case 3
        t1 = 4 * x ^ 3 - 3 * x ^ 2 * (x1 + x2 + x4 + x5);
        t1 = t1 + 2 * x * (x1 * x4 + x1 * x5 + x4 * x5 + x1 * x2 + x2 * x5 + x2 * x4);
        t1 = t1 - (x1 * x2 * x4 + x1 * x2 * x5 + x1 * x4 * x5 + x2 * x4 * x5);
      case 4
        t1 = 4 * x ^ 3 - 3 * x ^ 2 * (x1 + x2 + x3 + x5);
        t1 = t1 + 2 * x * (x1 * x2 + x1 * x3 + x1 * x5 + x2 * x3 + x2 * x5 + x3 * x5);
        t1 = t1 - (x1 * x2 * x3 + x1 * x2 * x5 + x1 * x3 * x5 + x2 * x3 * x5);
      case 5
        t1 = 4 * x ^ 3 - 3 * x ^ 2 * (x1 + x2 + x3 + x4);
        t1 = t1 + 2 * x * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4);
```



```
        t1 = t1 - (x1 * x2 * x3 + x1 * x2 * x4 + x1 * x3 * x4 + x2 * x3 * x4);
    end
    t2 = 1;
    for j = nStartIdx:nStartIdx + 4
      if i ~= j
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end

function d = deriv2_5(xarr, yarr, x)
%DERIV2_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  x1 = xarr(nStartIdx);
  x2 = xarr(nStartIdx + 1);
  x3 = xarr(nStartIdx + 2);
  x4 = xarr(nStartIdx + 3);
  x5 = xarr(nStartIdx + 4);

  zsum = 0;

  for i = nStartIdx:nStartIdx + 4
    j = i - nStartIdx + 1;
    switch(j)
      case 1
        t1 = 12*x ^ 2 - 6*x*(x2 + x3 + x4 + x5);
        t1 = t1 + 2 * (x2 * x3 + x2 * x4 + x2 * x5 + x3 * x4 + x3 * x5 + x4 * x5);
      case 2
        t1 = 12*x ^ 2 - 6*x*(x1 + x3 + x4 + x5);
        t1 = t1 + 2 * (x1 * x3 + x1 * x4 + x1 * x5 + x3 * x4 + x3 * x5 + x4 * x5);
      case 3
        t1 = 12*x ^ 2 - 6*x*(x1 + x2 + x4 + x5);
        t1 = t1 + 2 * (x1 * x4 + x1 * x5 + x4 * x5 + x1 * x2 + x2 * x5 + x2 * x4);
      case 4
        t1 = 12*x ^ 2 - 6*x*(x1 + x2 + x3 + x5);
        t1 = t1 + 2 * (x1 * x2 + x1 * x3 + x1 * x5 + x2 * x3 + x2 * x5 + x3 * x5);
      case 5
        t1 = 12*x ^ 2 - 6*x*(x1 + x2 + x3 + x4);
        t1 = t1 + 2 * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4);
     end
    t2 = 1;
```



```
    for j = nStartIdx:nStartIdx + 4
      if i ~= j
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end

function d = deriv3_5(xarr, yarr, x)
%DERIV3_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  x1 = xarr(nStartIdx);
  x2 = xarr(nStartIdx + 1);
  x3 = xarr(nStartIdx + 2);
  x4 = xarr(nStartIdx + 3);
  x5 = xarr(nStartIdx + 4);

  zsum = 0;

  for i = nStartIdx:nStartIdx + 4
    j = i - nStartIdx + 1;
    switch(j)
      case 1
        t1 = 24*x - 6*(x2 + x3 + x4 + x5);
      case 2
        t1 = 24*x - 6*(x1 + x3 + x4 + x5);
      case 3
        t1 = 24*x - 6*(x1 + x2 + x4 + x5);
      case 4
        t1 = 24*x - 6*(x1 + x2 + x3 + x5);
      case 5
        t1 = 24*x - 6*(x1 + x2 + x3 + x4);
     end
    t2 = 1;
    for j = nStartIdx:nStartIdx + 4
      if i ~= j
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
```



```
end

function d = deriv4_5(xarr, yarr, x)
%DERIV4_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  zsum = 0;
  for i = nStartIdx:nStartIdx + 4
    t1 = 24;
    t2 = 1;
    for j = nStartIdx:nStartIdx + 4
      if i ~= j
        t2 = t2 * (xarr(i) - xarr(j));
      end
    end
    zsum = zsum + yarr(i) * t1 / t2;
  end
  d = zsum;
end
```

Implementation in Python

This section presents the implementation of the algorithms presented earlier in Python:

```
def deriv1_3(xarr, yarr, x):
#DERIV1_3 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 3
  else:
    for i in range(n):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3:
      nStartIdx = n - 3
```



```
  zsum = 0
  for i in range(nStartIdx,nStartIdx + 3):
    t1 = 0
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 3):
      if i != j:
        t1 += x - xarr[j]
        t2 *= xarr[i] - xarr[j]
    zsum += yarr[i] * t1 / t2
  return zsum


def deriv2_3(xarr, yarr, x):
#DERIV2_3 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 3
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3:
      nStartIdx = n - 3


  zsum = 0
  for i in range(nStartIdx,nStartIdx + 3):
    t1 = 2
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 3):
      if i != j:
        t2 *= (xarr[i] - xarr[j])
    zsum += yarr[i] * t1 / t2
  return zsum

def deriv1_4(xarr, yarr, x):
#DERIV1_4 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 4
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4:
```



```
      nStartIdx = n - 4

  zsum = 0
  for i in range(nStartIdx,nStartIdx + 4):
    t1 = 3 * x ** 2
    t2 = 1
    for k in range( nStartIdx,nStartIdx + 3):
      if k != i:
        for j in range(k + 1,nStartIdx + 4):
          if k != j and i != j:
            t1 = t1 + xarr[k] * xarr[j]
    for j in range(nStartIdx,nStartIdx + 4):
      if i != j:
        t1 -= 2 * x * xarr[j]
        t2 *= xarr[i] - xarr[j]
    zsum += yarr[i] * t1 / t2
  return zsum

def deriv2_4(xarr, yarr, x):
#DERIV2_4 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 4
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4:
      nStartIdx = n - 4

  zsum = 0
  for i in range(nStartIdx,nStartIdx + 4):
    t1 = 6 * x
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 4):
      if i != j:
        t1 -= 2 * xarr[j]
        t2 *= xarr[i] - xarr[j]
    zsum += yarr[i] * t1 / t2
  return zsum

def deriv3_4(xarr, yarr, x):
#DERIV3_4 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 4
  else:
```



```
    for i in range(0,n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4:
      nStartIdx = n - 4

  zsum = 0
  for i in range(nStartIdx,nStartIdx + 4):
    t1 = 6
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 4):
      if i != j:
        t2 = t2 * (xarr[i] - xarr[j])
    zsum = zsum + yarr[i] * t1 / t2
  return zsum

def  deriv1_5(xarr, yarr, x):
#DERIV1_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 5
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

  x1 = xarr[nStartIdx]
  x2 = xarr[nStartIdx + 1]
  x3 = xarr[nStartIdx + 2]
  x4 = xarr[nStartIdx + 3]
  x5 = xarr[nStartIdx + 4]

  zsum = 0

  for i in range(nStartIdx,nStartIdx + 5):
    j = i - nStartIdx + 1
    if j==1:
      t1 = 4 * x ** 3 - 3 * x ** 2 * (x2 + x3 + x4 + x5)
      t1 += 2 * x * (x2 * x3 + x2 * x4 + x2 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
      t1 -= x2 * x3 * x4 + x2 * x3 * x5 + x2 * x4 * x5 + x3 * x4 * x5
    elif j==2:
      t1 = 4 * x ** 3 - 3 * x ** 2 * (x1 + x3 + x4 + x5)
      t1 += 2 * x * (x1 * x3 + x1 * x4 + x1 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
      t1 -= x1 * x3 * x4 + x1 * x3 * x5 + x1 * x4 * x5 + x3 * x4 * x5
    elif j==3:
      t1 = 4 * x ** 3 - 3 * x ** 2 * (x1 + x2 + x4 + x5)
      t1 += 2 * x * (x1 * x4 + x1 * x5 + x4 * x5 + x1 * x2 + x2 * x5 + x2 * x4)
      t1 -= x1 * x2 * x4 + x1 * x2 * x5 + x1 * x4 * x5 + x2 * x4 * x5
    elif j==4:
      t1 = 4 * x ** 3 - 3 * x ** 2 * (x1 + x2 + x3 + x5)
```



```
      t1 +=  2 * x * (x1 * x2 + x1 * x3 + x1 * x5 + x2 * x3 + x2 * x5 + x3 * x5)
      t1 -= x1 * x2 * x3 + x1 * x2 * x5 + x1 * x3 * x5 + x2 * x3 * x5
    elif j==5:
      t1 = 4 * x ** 3 - 3 * x ** 2 * (x1 + x2 + x3 + x4)
      t1 +=  2 * x * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4)
      t1 -= x1 * x2 * x3 + x1 * x2 * x4 + x1 * x3 * x4 + x2 * x3 * x4
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 5):
      if i != j:
        t2 *= xarr[i] - xarr[j]
    zsum += yarr[i] * t1 / t2
  return zsum

def deriv2_5(xarr, yarr, x):
#DERIV2_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[1]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 5
  else:
    for i in range(0,n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

  x1 = xarr[nStartIdx]
  x2 = xarr[nStartIdx + 1]
  x3 = xarr[nStartIdx + 2]
  x4 = xarr[nStartIdx + 3]
  x5 = xarr[nStartIdx + 4]

  zsum = 0

  for i in range(nStartIdx,nStartIdx + 5):
    j = i - nStartIdx + 1
    if j==1:
      t1 = 12 * x ** 2 - 6 * x * (x2 + x3 + x4 + x5)
      t1 +=  2 *  (x2 * x3 + x2 * x4 + x2 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
    elif j==2:
      t1 = 12 * x ** 2 - 6 * x * (x1 + x3 + x4 + x5)
      t1 += 2 *  (x1 * x3 + x1 * x4 + x1 * x5 + x3 * x4 + x3 * x5 + x4 * x5)
    elif j==3:
      t1 = 12 * x ** 2 - 6 * x * (x1 + x2 + x4 + x5)
      t1 += 2 * (x1 * x4 + x1 * x5 + x4 * x5 + x1 * x2 + x2 * x5 + x2 * x4)
    elif j==4:
      t1 = 12 * x ** 2 - 6 * x * (x1 + x2 + x3 + x5)
      t1 += 2 * (x1 * x2 + x1 * x3 + x1 * x5 + x2 * x3 + x2 * x5 + x3 * x5)
    elif j==5:
      t1 = 12 * x ** 2 - 6 * x * (x1 + x2 + x3 + x4)
      t1 += 2 * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4)
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 5):
      if i != j:
        t2 *= xarr[i] - xarr[j]
```



```
    zsum += yarr[i] * t1 / t2
  return zsum


def deriv3_5(xarr, yarr, x):
#DERIV3_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 5
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nstart too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

  x1 = xarr[nStartIdx]
  x2 = xarr[nStartIdx + 1]
  x3 = xarr[nStartIdx + 2]
  x4 = xarr[nStartIdx + 3]
  x5 = xarr[nStartIdx + 4]

  zsum = 0

  for i in range(nStartIdx,nStartIdx + 5):
    j = i - nStartIdx + 1
    if j==1:
      t1 = 24 * x - 6 * (x2 + x3 + x4 + x5)
    elif j==2:
      t1 = 24 * x - 6 * (x1 + x3 + x4 + x5)
    elif j==3:
      t1 = 24 * x - 6 *(x1 + x2 + x4 + x5)
    elif j==4:
      t1 = 24 * x - 6 * (x1 + x2 + x3 + x5)
    elif j==5:
      t1 = 24 * x - 6 *(x1 + x2 + x3 + x4)
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 5):
      if i != j:
        t2 *= xarr[i] - xarr[j]
    zsum += yarr[i] * t1 / t2
  return zsum

def deriv4_5(xarr, yarr, x):
#DERIV4_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
```



```
    nStartIdx = n - 5
  else:
    for i in range(0,n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nstart too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

  zsum = 0
  for i in range(nStartIdx,nStartIdx + 5):
    t2 = 1
    for j in range(nStartIdx,nStartIdx + 5):
      if i != j:
        t2 = t2 * (xarr[i] - xarr[j])
    zsum = zsum + yarr[i] *24 / t2
  return zsum
```

Code Implementation for the Newton Version

This section presents implementation of the derivative functions for 3, 4, and 5 points in Excel VB, MATLAB, and Python.

Implementation in Excel VBA

This section presents the implementation of the algorithms presented earlier in Excel VBA:

```
Function deriv1_3(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, N As Integer
  Dim T1 As Double, T2 As Double
  Dim nStartIdx As Integer

  deriv1_3 = 1E+99
  N = UBound(Xarr)
  If N < 3 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 2
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 2 Then nStartIdx = N - 2
  End If

  J = nStartIdx
  T1 = (Yarr(J + 1) - Yarr(J)) / (Xarr(J + 1) - Xarr(J))
  T2 = (Yarr(J + 2) - Yarr(J + 1)) / (Xarr(J + 2) - Xarr(J + 1)) / (Xarr(J + 2) -
Xarr(J))
  T2 = T2 - (Yarr(J + 1) - Yarr(J)) / (Xarr(J + 1) - Xarr(J)) / (Xarr(J + 2) -
Xarr(J))
```



```
  deriv1_3 = T1 + T2 * (2 * x - Xarr(J) - Xarr(J + 1))
End Function


Function deriv2_3(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, N As Integer
  Dim T2 As Double
  Dim nStartIdx As Integer

  deriv2_3 = 1E+99
  N = UBound(Xarr)
  If N < 3 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 2
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 2 Then nStartIdx = N - 2
  End If

  J = nStartIdx
  T2 = (Yarr(J + 2) - Yarr(J + 1)) / (Xarr(J + 2) - Xarr(J + 1)) / (Xarr(J + 2) -
Xarr(J))
  T2 = T2 - (Yarr(J + 1) - Yarr(J)) / (Xarr(J + 1) - Xarr(J)) / (Xarr(J + 2) -
Xarr(J))
  deriv2_3 = 2 * T2
End Function


Function deriv1_4(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double
  Dim d11 As Double, d12 As Double, d13 As Double, d21 As Double, d22 As Double, d31
As Double

  deriv1_4 = 1E+99
  N = UBound(Xarr)
  If N < 4 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 3
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
```



```
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 3 Then nStartIdx = N - 3
  End If

  J = nStartIdx
  x1 = Xarr(J)
  x2 = Xarr(J + 1)
  x3 = Xarr(J + 2)
  x4 = Xarr(J + 3)
  y1 = Yarr(J)
  y2 = Yarr(J + 1)
  y3 = Yarr(J + 2)
  y4 = Yarr(J + 3)
  T1 = 2 * x - x1 - x2
  T2 = 3 * x ^ 2 - 2 * x * (x1 + x2 + x3) + x1 * x2 + x1 * x3 + x2 * x3
  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d31 = (d22 - d21) / (x4 - x1)
  deriv1_4 = d11 + d21 * T1 + d31 * T2
End Function

Function deriv2_4(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double
  Dim d11 As Double, d12 As Double, d13 As Double, d21 As Double, d22 As Double, d31
As Double

  deriv2_4 = 1E+99
  N = UBound(Xarr)
  If N < 4 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 3
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 3 Then nStartIdx = N - 3
  End If

  J = nStartIdx
  x1 = Xarr(J)
  x2 = Xarr(J + 1)
  x3 = Xarr(J + 2)
  x4 = Xarr(J + 3)
  y1 = Yarr(J)
  y2 = Yarr(J + 1)
  y3 = Yarr(J + 2)
  y4 = Yarr(J + 3)
  T1 = 2
```



```
  T2 = 6 * x - 2 * (x1 + x2 + x3)
  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d31 = (d22 - d21) / (x4 - x1)
  deriv2_4 = d21 * T1 + d31 * T2

End Function

Function deriv3_4(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double
  Dim d11 As Double, d12 As Double, d13 As Double, d21 As Double, d22 As Double, d31
As Double

  deriv3_4 = 1E+99
  N = UBound(Xarr)
  If N < 4 Or N <> UBound(Yarr) Then Exit Function


  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 3
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 3 Then nStartIdx = N - 3
  End If

  J = nStartIdx
  x1 = Xarr(J)
  x2 = Xarr(J + 1)
  x3 = Xarr(J + 2)
  x4 = Xarr(J + 3)
  y1 = Yarr(J)
  y2 = Yarr(J + 1)
  y3 = Yarr(J + 2)
  y4 = Yarr(J + 3)
  T2 = 6
  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d31 = (d22 - d21) / (x4 - x1)
  deriv3_4 = d31 * T2
End Function


Function deriv1_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
```



```
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double, T3 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double, y5 As Double
  Dim d11 As Double, d12 As Double, d13 As Double, d14 As Double
  Dim d21 As Double, d22 As Double, d23 As Double
  Dim d31 As Double, d32 As Double, d41 As Double

  deriv1_5 = 1E+99
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 4
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)
  y1 = Yarr(nStartIdx)
  y2 = Yarr(nStartIdx + 1)
  y3 = Yarr(nStartIdx + 2)
  y4 = Yarr(nStartIdx + 3)
  y5 = Yarr(nStartIdx + 4)
  T1 = 2 * x - x1 - x2
  T2 = 3 * x ^ 2 - 2 * x * (x1 + x2 + x3) + x1 * x2 + x1 * x3 + x2 * x3
  T3 = 4 * x ^ 3 - 3 * x ^ 2 * (x1 + x2 + x3 + x4) _
      + 2 * x * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4) _
      - x * (x1 * x2 * x3 + x1 * x2 * x4 + x1 * x3 * x4 + x2 * x3 * x4)
  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d14 = (y5 - y4) / (x5 - x4)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d23 = (d14 - d13) / (x5 - x3)
  d31 = (d22 - d21) / (x4 - x1)
  d32 = (d23 - d22) / (x5 - x2)
  d41 = (d32 - d31) / (x5 - x1)
  deriv1_5 = d11 + d21 * T1 + d31 * T2 + d41 * T3
End Function

Function deriv2_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double, T3 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double, y5 As Double
```



```
  Dim d11 As Double, d12 As Double, d13 As Double, d14 As Double
  Dim d21 As Double, d22 As Double, d23 As Double
  Dim d31 As Double, d32 As Double, d41 As Double

  deriv2_5 = 1E+99
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 4
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)
  y1 = Yarr(nStartIdx)
  y2 = Yarr(nStartIdx + 1)
  y3 = Yarr(nStartIdx + 2)
  y4 = Yarr(nStartIdx + 3)
  y5 = Yarr(nStartIdx + 4)
  T1 = 2
  T2 = 6 * x - 2 * (x1 + x2 + x3)
  T3 = 12 * x ^ 2 - 6 * x * (x1 + x2 + x3 + x4) _
       + 2 * (x1 * x2 + x1 * x3 + x1 * x4 + x2 * x3 + x2 * x4 + x3 * x4)

  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d14 = (y5 - y4) / (x5 - x4)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d23 = (d14 - d13) / (x5 - x3)
  d31 = (d22 - d21) / (x4 - x1)
  d32 = (d23 - d22) / (x5 - x2)
  d41 = (d32 - d31) / (x5 - x1)
  deriv2_5 = d21 * T1 + d31 * T2 + d41 * T3
End Function

Function deriv3_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double, T3 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double, y5 As Double
  Dim d11 As Double, d12 As Double, d13 As Double, d14 As Double
  Dim d21 As Double, d22 As Double, d23 As Double
  Dim d31 As Double, d32 As Double, d41 As Double

  deriv3_5 = 1E+99
```



```
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 4
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)
  y1 = Yarr(nStartIdx)
  y2 = Yarr(nStartIdx + 1)
  y3 = Yarr(nStartIdx + 2)
  y4 = Yarr(nStartIdx + 3)
  y5 = Yarr(nStartIdx + 4)
  T1 = 6
  T2 = 24 * x - 6 * (x1 + x2 + x3 + x4)

  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d14 = (y5 - y4) / (x5 - x4)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d23 = (d14 - d13) / (x5 - x3)
  d31 = (d22 - d21) / (x4 - x1)
  d32 = (d23 - d22) / (x5 - x2)
  d41 = (d32 - d31) / (x5 - x1)
  deriv3_5 = d31 * T1 + d41 * T2
End Function

Function deriv4_5(ByRef Xarr() As Double, ByRef Yarr() As Double, ByVal x As Double)
As Double
  Dim I As Integer, J As Integer, K As Integer, N As Integer
  Dim T1 As Double, T2 As Double, T3 As Double
  Dim nStartIdx As Integer
  Dim x1 As Double, x2 As Double, x3 As Double, x4 As Double, x5 As Double
  Dim y1 As Double, y2 As Double, y3 As Double, y4 As Double, y5 As Double
  Dim d11 As Double, d12 As Double, d13 As Double, d14 As Double
  Dim d21 As Double, d22 As Double, d23 As Double
  Dim d31 As Double, d32 As Double, d41 As Double

  deriv4_5 = 1E+99
  N = UBound(Xarr)
  If N < 5 Or N <> UBound(Yarr) Then Exit Function

  If x < Xarr(1) Then
    nStartIdx = 1
  ElseIf x > Xarr(N) Then
    nStartIdx = N - 4
```



```
  Else
    For I = 1 To N - 1
      If x > Xarr(I) And x < Xarr(I + 1) Then
        nStartIdx = I
        Exit For
      End If
    Next I
    ' nStart too close to the last Xarr array element?
    If nStartIdx > N - 4 Then nStartIdx = N - 4
  End If

  x1 = Xarr(nStartIdx)
  x2 = Xarr(nStartIdx + 1)
  x3 = Xarr(nStartIdx + 2)
  x4 = Xarr(nStartIdx + 3)
  x5 = Xarr(nStartIdx + 4)
  y1 = Yarr(nStartIdx)
  y2 = Yarr(nStartIdx + 1)
  y3 = Yarr(nStartIdx + 2)
  y4 = Yarr(nStartIdx + 3)
  y5 = Yarr(nStartIdx + 4)
  T1 = 24
  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d14 = (y5 - y4) / (x5 - x4)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d23 = (d14 - d13) / (x5 - x3)
  d31 = (d22 - d21) / (x4 - x1)
  d32 = (d23 - d22) / (x5 - x2)
  d41 = (d32 - d31) / (x5 - x1)
  deriv4_5 = d41 * T1

End Function
```

Each of the above functions have the same parameter lists, which are:

- The reference array Xarr() that passes the elements of array X. You can have more array elements than you need. The function returns 1E99 if the array size is insufficient.
- The reference array Yarr() that passes the elements of array Y. You can have more array elements than you need. The function returns 1E99 if the array sizes of arrays Yarr() and Xarr() are not equal.
- The parameter X is the value at which a derivative is estimated.

The functions compare the value of X with the first and last elements in array Xarr to determine the first of the index values to use in calculating the derivative. If that first index is too close to the last element of array Xarr, the functions decrease the value of the starting index by an appropriate value. By contrast, if X is less than the first element in array Xarr, the functions set the starting index as the first array index.

The next two sections present the translation of the above VBA code into MATLAB and Python. The comments I made about the VBA functions applies equally to the MATLAB and Python code.

Implementation in MATLAB

This section presents the implementation of the algorithms presented earlier in MATLAB:

```
function d = deriv1_3(xarr, yarr, x)
%DERIV1_3 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 3 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 2;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 2, nStartIdx = n - 2; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  d11 = (y2 - y1) / (x2 - x1);
  d12 = (y3 - y2) / (x3 - x2);
  d21 = (d12 - d11) / (x3 - x1);
  d=d11+d21*(2*x-x1-x2);
end

function d = deriv2_3(xarr, yarr, x)
%DERIV2_3 Summary of this function goes here
%   Detailed explanation goes here
  d=1e+99;
  n = length(xarr);
  if n < 3 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 2;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
```



```
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 2,  nStartIdx = n - 2; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d21=(d12-d11)/(x3-x1);
  d=2*d21;
end

function d = deriv1_4(xarr, yarr, x)
%DERIV1_4 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 4 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 3;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3,  nStartIdx = n - 3; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  x4=xarr(j+3);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  y4=yarr(j+3);
  T1=2*x-x1-x2;
  T2=3*x^2-2*x*(x1+x2+x3)+x1*x2+x1*x3+x2*x3;
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d13=(y4-y3)/(x4-x3);
  d21=(d12-d11)/(x3-x1);
  d22=(d13-d12)/(x4-x2);
  d31=(d22-d21)/(x4-x1);
  d=d11+d21*T1+d31*T2;
end
```



```
function d = deriv2_4(xarr, yarr, x)
%DERIV2_4 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 4 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 3;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3, nStartIdx = n - 3; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  x4=xarr(j+3);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  y4=yarr(j+3);
  T1=2;
  T2=6*x-2*(x1+x2+x3);
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d13=(y4-y3)/(x4-x3);
  d21=(d12-d11)/(x3-x1);
  d22=(d13-d12)/(x4-x2);
  d31=(d22-d21)/(x4-x1);
  d=d21*T1+d31*T2;
end

function d = deriv3_4(xarr, yarr, x)
%DERIV3_4 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 4 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 3;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3,  nStartIdx = n - 3; end
  end
```



```
  j = nStartIdx;
  x1 = xarr(j);
  x2 = xarr(j + 1);
  x3 = xarr(j + 2);
  x4 = xarr(j + 3);
  y1 = yarr(j);
  y2 = yarr(j + 1);
  y3 = yarr(j + 2);
  y4 = yarr(j + 3);
  T2 = 6;
  d11 = (y2 - y1) / (x2 - x1);
  d12 = (y3 - y2) / (x3 - x2);
  d13 = (y4 - y3) / (x4 - x3);
  d21 = (d12 - d11) / (x3 - x1);
  d22 = (d13 - d12) / (x4 - x2);
  d31 = (d22 - d21) / (x4 - x1);
  d = d31 * T2;
end

function d = deriv1_5(xarr, yarr, x)
%DERIV1_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  x4=xarr(j+3);
  x5=xarr(j+4);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  y4=yarr(j+3);
  y5=yarr(j+4);
  T1=2*x-x1-x2;
  T2=3*x^2-2*x*(x1+x2+x3)+x1*x2+x1*x3+x2*x3;
  T3=4*x^3-3*x^2*(x1+x2+x3+x4)+2*x*(x1*x2+x1*x3+x1*x4+x2*x3+x2*x4+x3*x4)-
x*(x1*x2*x3+x1*x2*x4+x1*x3*x4+x2*x3*x4)+x1*x2*x3*x4;
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d13=(y4-y3)/(x4-x3);
  d14=(y5-y4)/(x5-x4);
  d21=(d12-d11)/(x3-x1);
  d22=(d13-d12)/(x4-x2);
```



```
  d23=(d14-d13)/(x5-x3);
  d31=(d22-d21)/(x4-x1);
  d32=(d23-d22)/(x5-x2);
  d41=(d32-d31)/(x5-x1);
  d=d11+d21*T1+d31*T2+d41*T3 ;
end

function d = deriv2_5(xarr, yarr, x)
%DERIV2_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  x4=xarr(j+3);
  x5=xarr(j+4);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  y4=yarr(j+3);
  y5=yarr(j+4);
  T1=2;
  T2=6*x-2*(x1+x2+x3);
  T3=12*x^2-6*x*(x1+x2+x3+x4)+2*(x1*x2+x1*x3+x1*x4+x2*x3+x2*x4+x3*x4);
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d13=(y4-y3)/(x4-x3);
  d14=(y5-y4)/(x5-x4);
  d21=(d12-d11)/(x3-x1);
  d22=(d13-d12)/(x4-x2);
  d23=(d14-d13)/(x5-x3);
  d31=(d22-d21)/(x4-x1);
  d32=(d23-d22)/(x5-x2);
  d41=(d32-d31)/(x5-x1);
  d=d21*T1+d31*T2+d41*T3;
end

function d = deriv3_5(xarr, yarr, x)
%DERIV3_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end
```



```
  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end

  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  x4=xarr(j+3);
  x5=xarr(j+4);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  y4=yarr(j+3);
  y5=yarr(j+4);
  T1=6;
  T2=24*x-6*(x1+x2+x3+x4);
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d13=(y4-y3)/(x4-x3);
  d14=(y5-y4)/(x5-x4);
  d21=(d12-d11)/(x3-x1);
  d22=(d13-d12)/(x4-x2);
  d23=(d14-d13)/(x5-x3);
  d31=(d22-d21)/(x4-x1);
  d32=(d23-d22)/(x5-x2);
  d41=(d32-d31)/(x5-x1);
  d=d31*T1+d41*T2;
end

function d = deriv4_5(xarr, yarr, x)
%DERIV4_5 Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(xarr);
  if n < 5 || n ~= length(yarr), return; end

  if x < xarr(1)
    nStartIdx = 1;
  elseif x > xarr(n)
    nStartIdx = n - 4;
  else
    for i = 1:n - 1
      if x > xarr(i) && x < xarr(i + 1)
        nStartIdx = i;
        break;
      end
    end
    % nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4, nStartIdx = n - 4; end
  end
```



```
  j=nStartIdx;
  x1=xarr(j);
  x2=xarr(j+1);
  x3=xarr(j+2);
  x4=xarr(j+3);
  x5=xarr(j+4);
  y1=yarr(j);
  y2=yarr(j+1);
  y3=yarr(j+2);
  y4=yarr(j+3);
  y5=yarr(j+4);
  T1=24;
  T2=24*x-6*(x1+x2+x3+x4);
  d11=(y2-y1)/(x2-x1);
  d12=(y3-y2)/(x3-x2);
  d13=(y4-y3)/(x4-x3);
  d14=(y5-y4)/(x5-x4);
  d21=(d12-d11)/(x3-x1);
  d22=(d13-d12)/(x4-x2);
  d23=(d14-d13)/(x5-x3);
  d31=(d22-d21)/(x4-x1);
  d32=(d23-d22)/(x5-x2);
  d41=(d32-d31)/(x5-x1);
  d=d41*T1;
end
```

Implementation in Python

This section presents the implementation of the algorithms presented earlier in Python:

```
def deriv1_3(xarr, yarr, x):
#DERIV1_3 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 3
  else:
    for i in range(n):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3:
      nStartIdx = n - 3

  j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  y1=yarr[j]
  y2=yarr[j+1]
  y3=yarr[j+2]
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d21=(d12-d11)/(x3-x1)
  return d11+d21*(2*x-x1-x2)
```



```
def deriv2_3(xarr, yarr, x):
#DERIV2_3 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 3
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 3:
      nStartIdx = n - 3


  j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  y1=yarr[j]
  y2=yarr[j+1]
  y3=yarr[j+2]
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d21=(d12-d11)/(x3-x1)
  return 2*d21

def deriv1_4(xarr, yarr, x):
#DERIV1_4 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 4
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4:
      nStartIdx = n - 4

  j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  x4=xarr[j+3]
  y1=yarr[j]
```



```
  y2=yarr[j+1]
  y3=yarr[j+2]
  y4=yarr[j+3]
  T1=2*x-x1-x2
  T2=3*x**2-2*x*(x1+x2+x3)+x1*x2+x1*x3+x2*x3
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d13=(y4-y3)/(x4-x3)
  d21=(d12-d11)/(x3-x1)
  d22=(d13-d12)/(x4-x2)
  d31=(d22-d21)/(x4-x1)
  d=d11+d21*T1+d31*T2
  return d

def deriv2_4(xarr, yarr, x):
#DERIV2_4 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 4
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4:
      nStartIdx = n - 4

  j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  x4=xarr[j+3]
  y1=yarr[j]
  y2=yarr[j+1]
  y3=yarr[j+2]
  y4=yarr[j+3]
  T1=2
  T2=6*x-2*(x1+x2+x3)
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d13=(y4-y3)/(x4-x3)
  d21=(d12-d11)/(x3-x1)
  d22=(d13-d12)/(x4-x2)
  d31=(d22-d21)/(x4-x1)
  d=d21*T1+d31*T2
  return d

def deriv3_4(xarr, yarr, x):
#DERIV3_4 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d
```



```
  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 4
  else:
    for i in range(0,n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 4:
      nStartIdx = n - 4

  j = nStartIdx
  x1 = xarr[j]
  x2 = xarr[j + 1]
  x3 = xarr[j + 2]
  x4 = xarr[j + 3]
  y1 = yarr[j]
  y2 = yarr[j + 1]
  y3 = yarr[j + 2]
  y4 = yarr[j + 3]
  T2 = 6
  d11 = (y2 - y1) / (x2 - x1)
  d12 = (y3 - y2) / (x3 - x2)
  d13 = (y4 - y3) / (x4 - x3)
  d21 = (d12 - d11) / (x3 - x1)
  d22 = (d13 - d12) / (x4 - x2)
  d31 = (d22 - d21) / (x4 - x1)
  d = d31 * T2
  return d

def  deriv1_5(xarr, yarr, x):
#DERIV1_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 5
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

  j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  x4=xarr[j+3]
  x5=xarr[j+4]
  y1=yarr[j]
  y2=yarr[j+1]
  y3=yarr[j+2]
  y4=yarr[j+3]
```



```
  y5=yarr[j+4]
  T1=2*x-x1-x2
  T2=3*x**2-2*x*(x1+x2+x3)+x1*x2+x1*x3+x2*x3
  T3=4*x**3-3*x**2*(x1+x2+x3+x4)+2*x*(x1*x2+x1*x3+x1*x4+x2*x3+x2*x4+x3*x4)-
x*(x1*x2*x3+x1*x2*x4+x1*x3*x4+x2*x3*x4)+ x1*x2*x3*x4
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d13=(y4-y3)/(x4-x3)
  d14=(y5-y4)/(x5-x4)
  d21=(d12-d11)/(x3-x1)
  d22=(d13-d12)/(x4-x2)
  d23=(d14-d13)/(x5-x3)
  d31=(d22-d21)/(x4-x1)
  d32=(d23-d22)/(x5-x2)
  d41=(d32-d31)/(x5-x1)
  d=d11+d21*T1+d31*T2+d41*T3
  return d

def deriv2_5(xarr, yarr, x):
#DERIV2_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[1]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 5
  else:
    for i in range(0,n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nStartIdx too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

    j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  x4=xarr[j+3]
  x5=xarr[j+4]
  y1=yarr[j]
  y2=yarr[j+1]
  y3=yarr[j+2]
  y4=yarr[j+3]
  y5=yarr[j+4]
  T1=2
  T2=6*x-2*(x1+x2+x3)
  T3=12*x**2-6*x*(x1+x2+x3+x4)+2*(x1*x2+x1*x3+x1*x4+x2*x3+x2*x4+x3*x4)
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d13=(y4-y3)/(x4-x3)
  d14=(y5-y4)/(x5-x4)
  d21=(d12-d11)/(x3-x1)
  d22=(d13-d12)/(x4-x2)
  d23=(d14-d13)/(x5-x3)
  d31=(d22-d21)/(x4-x1)
  d32=(d23-d22)/(x5-x2)
  d41=(d32-d31)/(x5-x1)
```



```
  d=d21*T1+d31*T2+d41*T3
  return d


def deriv3_5(xarr, yarr, x):
#DERIV3_5 Summary of this function goes here
#   Detailed explanation goes here
  d = 1e+99
  n = len(xarr)
  if n < 3 or n != len(yarr):
    return d

  if x < xarr[0]:
    nStartIdx = 0
  elif x > xarr[n-1]:
    nStartIdx = n - 5
  else:
    for i in range(n - 1):
      if x > xarr[i] and x < xarr[i + 1]:
        nStartIdx = i
        break
    # nstart too close to the last xarr array element?
    if nStartIdx > n - 5:
      nStartIdx = n - 5

    j=nStartIdx
  x1=xarr[j]
  x2=xarr[j+1]
  x3=xarr[j+2]
  x4=xarr[j+3]
  x5=xarr[j+4]
  y1=yarr[j]
  y2=yarr[j+1]
  y3=yarr[j+2]
  y4=yarr[j+3]
  y5=yarr[j+4]
  T1=6
  T2=24*x-6*(x1+x2+x3+x4)
  d11=(y2-y1)/(x2-x1)
  d12=(y3-y2)/(x3-x2)
  d13=(y4-y3)/(x4-x3)
  d14=(y5-y4)/(x5-x4)
  d21=(d12-d11)/(x3-x1)
  d22=(d13-d12)/(x4-x2)
  d23=(d14-d13)/(x5-x3)
  d31=(d22-d21)/(x4-x1)
  d32=(d23-d22)/(x5-x2)
  d41=(d32-d31)/(x5-x1)
  d=d31*T1+d41*T2
  return d

# test code
n=10
x=[]
y=[]
z=[0,0.5,1.0,1.2,1.7,2.2,3.2,3.7,4,4.2,4.5,4.8, 5, 5.5]
for t in z:
  x.append(t)
  y.append(t**4)

xint=1.5
print("Testing first derivatives")
print("f1=(", xint, ")=", 4*xint**3)
```



```
print("f1,3=(", xint, ")=", deriv1_3(x,y,xint))
print("f1,4=(", xint, ")=", deriv1_4(x,y,xint))
print("f1,5=(", xint, ")=",  deriv1_5(x,y,xint))
print("")
print("Testing second derivatives")
print("f1=(", xint, ")=",  12*xint**2)
print("f2,3=(", xint, ")=", deriv2_3(x,y,xint))
print("f2,43=(", xint, ")=", deriv2_4(x,y,xint))
print("f2,5=(", xint, ")=", deriv2_5(x,y,xint))
print("")
print("Testing third derivatives")
print("f3=(", xint, ")=", 24*xint)
print("f3,4=(", xint, ")=", deriv3_4(x,y,xint))
print("f3,5=(", xint, ")=", deriv3_5(x,y,xint))
```

Final Comments

Estimating the derivatives of unequally spaced points offers a practical advantage over its equally spaced counterpart calculations. The accuracy of the derivatives depends on the gap between the x values and the errors in measuring the x and y values. In addition, higher derivatives (like the third and fourth derivative) are more prone to deviation from the true values. Nevertheless, estimating the derivatives of unequally spaced points is handy.

One general case that comes to mind (as a chemical engineer myself) is determining the chemical reaction rates based on measuring the concentration of a chemical reactant or product over time. To determine the reaction rate, one typically estimates the first derivative of the concentration with respect to time. Accuracy is not really paramount in this case. If the first derivative is closer to 1 than to 2 or to 3, then the reaction is a first rate one. Likewise, if the first derivative is closer to 2 than to 1 or to 3, then the reaction is a second rate one, and so on. The researchers are the one who ultimately make the call for the reaction rate (which could also be multiples of 0.5).

Document History

Version	*Release Date*	*Comments*
1.0.0	June 22, 2021	Initial release.
2.0.0	July 17, 2021	Updated with Newton polynomial version

