

MATH 4640 Numerical Analysis - HW 2 Solutions

Austin Barton

February 27, 2025

Problem 1:

$$y_0 = -1 \quad y_4 = 1$$

$$\Delta x_i = 1 \quad \forall i \in \{0, 1, 2, 3, 4\}$$

$$\Delta x_i + \Delta x_{i+1} = 2 \quad \forall i \in \{0, 1, 2, 3\}$$

Note: This is important to understand the simplifications to the general form below

$$A = \begin{bmatrix} \frac{1}{3} & \frac{1}{6} & 0 & 0 & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \\ 0 & 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ 0 & 0 & 0 & \frac{1}{6} & \frac{1}{3} \end{bmatrix}$$

$$M = \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix}$$

$$D = \begin{bmatrix} f_1 - f_0 - y_0 \\ f_2 - f_1 - (f_1 - f_0) \\ f_3 - f_2 - (f_2 - f_1) \\ f_4 - f_3 - (f_3 - f_2) \\ y_4 - (f_4 - f_3) \end{bmatrix} = \begin{bmatrix} -0.3 + 1 \\ -0.3 + 0.3 \\ -0.2 + 0.3 \\ 0.2 + 0.2 \\ 1 - 0.2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0 \\ 0.1 \\ 0.4 \\ 0.8 \end{bmatrix}$$

We want to find M s.t.

$AM = D$, below is it expressed as a linear system of equations

$$\textcircled{1} \quad \frac{1}{3}M_0 + \frac{1}{6}M_1 = 0.7$$

$$\textcircled{2} \quad \frac{1}{6}M_0 + \frac{2}{3}M_1 + \frac{1}{6}M_2 = 0$$

$$\textcircled{3} \quad \frac{1}{6}M_1 + \frac{2}{3}M_2 + \frac{1}{6}M_3 = 0.1$$

$$\textcircled{4} \quad \frac{1}{6}M_2 + \frac{2}{3}M_3 + \frac{1}{6}M_4 = 0.4$$

$$\textcircled{5} \quad \frac{1}{6}M_3 + \frac{1}{3}M_4 = 0.8$$

$$AM = \begin{bmatrix} \frac{2M_0 + M_1}{6} \\ \frac{4M_1 + M_0 + M_2}{6} \\ \frac{4M_2 + M_1 + M_3}{6} \\ \frac{4M_3 + M_2 + M_4}{6} \\ \frac{2M_4 + M_3}{6} \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0 \\ 0.1 \\ 0.4 \\ 0.8 \end{bmatrix}$$

$$\frac{2M_4 + M_3}{6} = 0.8 \Rightarrow 2M_4 + M_3 = 4.8$$

$$\Rightarrow M_4 = 2.4 - \frac{1}{2}M_3$$

$$\frac{4M_3 + M_2 + M_4}{6} = 0.4 \Rightarrow 4M_3 + M_2 + M_4 = 2.4$$

$$\begin{aligned} \Rightarrow M_2 &= 2.4 - M_4 - 4M_3 = 2.4 - (2.4 - \frac{1}{2}M_3) - 4M_3 \\ &= \frac{1}{2}M_3 - 4M_3 = -\frac{7}{2}M_3 \\ M_2 &= -\frac{7}{2}M_3 \end{aligned}$$

$$\frac{4M_2 + M_1 + M_3}{6} = 0.1 \Rightarrow 4M_2 + M_1 + M_3 = 0.6$$

$$\begin{aligned} \Rightarrow M_1 &= 0.6 - 4M_2 - M_3 = 0.6 - 4(-\frac{7}{2}M_3) - M_3 \\ &= 0.6 + 14M_3 - M_3 = 0.6 + 13M_3 \\ M_1 &= 0.6 + 13M_3 \end{aligned}$$

$$\frac{4M_1 + M_0 + M_2}{6} = 0 \Rightarrow M_0 = -4M_1 - M_2$$

$$= -4(0.6 + 13M_3) - (-\frac{7}{2}M_3) = -2.4 - 52M_3 + \frac{7}{2}M_3$$

$$M_0 = -2.4 - 52M_3 + \frac{7}{2}M_3 = -2.4 - \frac{97}{2}M_3$$

$$\begin{aligned} \frac{2M_0 + M_1}{6} &= 0.7 = \frac{2(-2.4 - \frac{97}{2}M_3) + (0.6 + 13M_3)}{6} \\ &= \frac{-4.8 - 97M_3 + 0.6 + 13M_3}{6} \end{aligned}$$

$$\Rightarrow 4.2 = -4.2 - 84M_3 \Rightarrow 8.4 = -84M_3$$

$$\Rightarrow -0.1 = M_3$$

$$M_3 = -0.1, \text{ thus,}$$

$$M_0 = -2.4 - 52(-0.1) + \frac{7}{2}(-0.1) = 2.45$$

$$M_1 = 0.6 + 13(-0.1) = -0.7$$

$$M_2 = -\frac{7}{2}(-0.1) = 0.35$$

$$M_4 = 2.4 - \frac{1}{2}(-0.1) = 2.45$$

$$M = \begin{bmatrix} 2.45 \\ -0.7 \\ 0.35 \\ -0.1 \\ 2.45 \end{bmatrix}$$

$$p_0(x) = \frac{(1-x)^3 M_0 + x^3 M_1}{6} + (1-x)(1.1) + x(0.8)$$

$$- \frac{1}{6} \left((1-x)M_0 + xM_1 \right), \quad x \in [0, 1]$$

$$p_1(x) = \frac{(2-x)^3 M_1 + (x-1)^3 M_2}{6} + (2-x)(0.8) + (x-1)(0.5)$$

$$- \frac{1}{6} \left((2-x)M_1 + (x-1)M_2 \right), \quad x \in [1, 2]$$

$$p_2(x) = \frac{(3-x)^3 M_2 + (x-2)^3 M_3}{6} + (3-x)(0.5) + (x-2)(0.3)$$

$$- \frac{1}{6} \left((3-x)M_2 + (x-2)M_3 \right), \quad x \in [2, 3]$$

$$p_3(x) = \frac{(4-x)^3 M_3 + (x-3)^3 M_4}{6} + (4-x)(0.3) + (x-3)(0.5)$$

$$- \frac{1}{6} \left((4-x)M_3 + (x-3)M_4 \right), \quad x \in [3, 4]$$

where $M_0 = 2.45$

$M_1 = -0.7$

$M_2 = 0.35$

$M_3 = -0.1$

$M_4 = 2.45$

Problem 2:

Problem 3:

Problem 4:



```
1 // Newton's Divided Difference Algorithm
2 pub fn nddx(x: &[f64], y: &[f64]) -> Vec<f64> {
3     println!("Beginning Newton's Divided Difference algorithm...");
4     let n = x.len();
5     assert_eq!(n, y.len(), "x and y must have the same length.");
6
7     // Div Diff table
8     let mut d = vec![vec![0.0; n]; n];
9
10    // First column is y
11    for i in 0..n {
12        d[i][0] = y[i];
13    }
14
15    println!("Calculating divided difference table...");
16    for j in 1..n {
17        for i in 0..(n - j) {
18            let numerator: f64 = d[i + 1][j - 1] - d[i][j - 1];
19            let denominator: f64 = x[i + j] - x[i];
20            d[i][j] = numerator / denominator;
21            println!(
22                "f[x(), ..., x()] = ({} - {}) / ({} - {}) = {}^{}",
23                i + j,
24                d[i + 1][j - 1],
25                d[i][j - 1],
26                x[i + j],
27                x[i],
28                d[i][j]
29            );
30        }
31    }
32    println!("Success! Divided difference table:");
33    for row in 0..n {
34        println!("{}", row);
35    }
36    d[0].clone()
37 }
38
39 // Interpolation given NDD coefficients
40 // p(x) = d[0] + (x - x[0])*d[1] + (x - x[0])*(x - x[1])*d[2] + ...
41 pub fn interpolate_polynomial(x: &[f64], d: &[f64], t: f64, n: usize) -> f64 {
42     println!("Beginning interpolation of polynomial at point (t):");
43     let mut p = d[0];
44     for i in 0..(n - 1).rev() {
45         p = d[i] + (t - x[i]) * p;
46     }
47     println!("Interpolation succeeded.");
48     p
49 }
50
51 #[cfg(test)]
52 mod tests {
53     use super::*;
54
55     #[test]
56     fn test_nddx() {
57         let x = [1.0, 2.0, 3.0, 4.0];
58         let y = [1.0, 8.0, 27.0, 64.0];
59         let d = nddx(&x, &y);
60         println!("Divided difference table:");
61         for row in 0..d.len() {
62             println!("{}", row);
63         }
64         let p = interpolate_polynomial(&x, &d, 5.0, d.len());
65         println!("Interpolated value at x=5.0: {}", p);
66     }
67 }
```

Figure 1: Screenshot of code for the Newton's Divided Difference (NDD) algorithm (upper) and the interpolate using the NDD coefficients (lower).

Problem 5:

Problem 6:

Problem 7:

Problem 8:

```

Thursday February 27, 08:59
ta
2 use crate::math::approx::round::coefficients;
3 use crate::math::interpolation::interpolate::polynomial, ndd;
4
5
6
7 impl DataPoints {
8     fn new(x: Vec<f64>, y: Vec<f64>) -> Result<Self, &'static str> {
9         if !Self::is_valid(&x, &y) {
10             return Err("x and y must have the same length");
11         }
12         Ok(DataPoints { x, y })
13     }
14
15     fn is_valid(x: &Vec<f64>, y: &Vec<f64>) -> bool {
16         x.len() == y.len()
17     }
18 }
19
20 fn solve_p4(data: &DataPoints) {
21     let d = ndd(&data.x, &data.y);
22     let d_r = round::coefficients(&d, 1e-3);
23
24     let m = data.x.len();
25
26     println!("Newton's Divided Difference Coefficients: {:?}", &d_r[0..]);
27
28     let t_values = vec![-3., -2., -1., 0., 0.5, 1., 1.5, 2., 3., 4.];
29     println!("Interpolating values: {:?}", t_values);
30     for &t in t_values.iter() {
31         let f_t = interpolate::polynomial(&data.x, &d_r, &t, 6n);
32         println!("p({}) = {:.3}", t, f_t);
33     }
34 }
35
36 pub fn main() {
37     // GTID: 903742722
38     // w=7, y=2, z=2
39     let x: Vec<f64> = vec![-2., -1., 0., 1., 2., 3.];
40     // l=7, y=1, z=2
41     let y: Vec<f64> = vec![7., 2., 1., 0., 2., 2.];
42
43     match DataPoints::new(x, y) {
44         Ok(data) => solve_p4(&data),
45         Err(e) => println!("Error creating DataPoints: {}", e),
46     }
47 }

```

Normal P rust 0x0+ - /Documents/Academics/Math/Numerical/numerical_analysis_math4640/solutions/rust/hw2/src/q4.rs
 "hw2/src/q4.rs" 50L, 1340B written
 numerical@nvim: 1:nvim- "cabin" 08:59 27-Feb-28

Figure 2: Screenshot of code using the DivDif and Interpolate algorithms.

```

Thursday February 27, 09:06
ta
42 fn evaluate_on_rand(
43     x_k: &f64,
44     d: &f64,
45     num_points: i32,
46 ) -> (Vec<f64>, Vec<f64>, Vec<f64>) {
47     let mut p_random_points: Vec<f64> = Vec::new();
48     let mut error: Vec<f64> = Vec::new();
49
50     let mut rng = rand::rng();
51     let random_points: Vec<f64> = (0..num_points)
52         .map(|_| rng.random_range(-5.0..5.0))
53         .collect();
54
55     let f_random_points: Vec<f64> = random_points.iter().map(|&r| function.f(r)).collect();
56
57     for (i, &t) in random_points.iter().enumerate() {
58         p_random_points.push(interpolate::polynomial(&x_k, &d, &t, &x_k.len()));
59         error.push((p_random_points[i] - f_random_points[i]).abs());
60     }
61
62     (random_points, f_random_points, p_random_points, error)
63 }
64
65 fn solve_q5(a: f64, b: f64, n: i32) {
66     let step = (b - a) / (n + 2) as f64;
67     println!("Polynomial will have degree of {}", n);
68     let x_k: Vec<f64> = (1..n + 2).map(|i| a + (i as f64) * step).collect();
69     println!("Evenly spaced points on [a, b]: {:?}", a, b, x_k);
70
71     let f_x_k: Vec<f64> = x_k.iter().map(|&x| function.f(x)).collect();
72     println!("Data Table: \n x = {:?} \n f(x) = {:?}", x_k, f_x_k);
73
74     let d = ndd(&x_k, &f_x_k);
75     println!(
76         "Newton Divided Difference coefficients for polynomial of degree {}: {:?}",
77         n, d
78     );
79
80     let (random_points, f_random_points, p_random_points, error) =
81         evaluate_on_rand(&x_k, &d, 2000);
82
83     match plot_interpolation_results(
84         random_points,
85         &f_random_points,
86         &p_random_points,
87         &error,
88         &x_k,
89         &f_x_k,
90     ) {
91         Ok(_) => println!("Plot saved as 'interpolation_plot_1.png'", n),
92         Err(e) => println!("Failed to create plot: {}", e),
93     }
94 }
95
96 pub fn main() {
97     solve_q5(-5., 5., 2);
98     solve_q5(-5., 5., 4);
99 }

```

N P rust 0x0+ - /Documents/Academics/Math/Numerical/numerical_analysis_math4640/solutions/rust/hw2/src/q5.rs
 "hw2/src/q5.rs" 50L, 1340B written
 numerical@nvim: 1:nvim- "cabin" 09:06 27-Feb-28

Figure 3: Screenshot of code using DivDif and Interpolate.

```

Thursday February 27, 09:05
ta
33 fn plot_interpolation_results(
32   r: &f64,
31   f: &f64,
30   p: &f64,
29   error: &f64,
28   x: &f64,
27   y: &f64,
26   degree: i32,
25 ) -> Result<(), Box

```

Figure 4: Screenshot of code plotting the true function, the interpolated polynomial, the interpolated points, and the error between the interpolated polynomial and true function.