Problems

1.

(i) Given $x_0 = -0.2, x_1 = 0$, and $x_2 = 0.2$ construct a second degree polynomial to approximate $f(x) = e^x$ via Newton's divided differences.

We want to derive a polynomial of the form

$$p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$

where $a_i = [x_0, \ldots, x_{i-1}]$ are the Newton Divided differences. For this problem, we have

$$\begin{aligned} a_0 &= f[x_0] = e^{x_0} = e^{-0.2}, \\ a_1 &= f[x_0, x_1] = \frac{e^{x_1} - e^{x_0}}{x_1 - x_0} = \frac{1 - e^{-0.2}}{0.2}, \end{aligned}$$

and

$$a_2 = f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{e^{0.2}-1}{0.2} - \frac{1-e^{-0.2}}{0.2}}{0.4}$$

which makes our polynomial

$$\begin{aligned} p(x) &= e^{-0.2} + \frac{1 - e^{-0.2}}{0.2}(x + 0.2) + \frac{\frac{e^{0.2}-1}{0.2} - \frac{1-e^{-0.2}}{0.2}}{0.4}(x + 0.2)(x) \\ &= 1 + 1.00668x + 0.501669x^2. \end{aligned}$$

(ii) Derive an error bound for $p_2(x)$ when $x \in [-0.2, 0.2]$.

First, note that the third derivative of f is maximized over $[-0.2, 0.2]$ when $x = 0.2$. Then, we can obtain a bound on our error as

$$\begin{aligned} E(t) &\leq \max_{t\in[-0.2,0.2]} E(t) \\ &= \max_{t\in[-0.2,0.2]} \frac{(t + 0.2)(t)(t - 0.2)}{6} e^{0.2} \\ &= \frac{(-\frac{\sqrt{3}}{15} + 0.2)(-\frac{\sqrt{3}}{15})(-\frac{\sqrt{3}}{15} - 0.2)}{6} e^{0.2} \\ &= 6.26824 \cdot 10^{-4}. \end{aligned}$$

(iii) Compute the error $E(0.1) = f(0.1) - p_2(0.1)$. How does this compare with the error bound?

Our error is

$$E(0.1) = |1.10517 - 1.10568| = 5.136621 \cdot 10^{-4}$$

which is within our error bound! So our error bound holds $x = 0.1$.

2.

(i) Show there is a unique cubic polynomial $p(x)$ for which

$$p(x_0) = f(x_0) \qquad p(x_2) = f(x_2)$$
$$p'(x_1) = f'(x_1) \qquad p''(x_1) = f''(x_1)$$

where $f(x)$ is a given function and $x_0 \neq x_2$.
Suppose $p(x)$ and $q(x)$ are two cubic polynomials satisfying

$$p(x_0) = q(x_0) = f(x_0) \qquad p(x_2) = q(x_2) = f(x_2)$$
$$p'(x_1) = q(x_1) = f'(x_1) \qquad p''(x_1) = q''(x_1) = f''(x_1).$$

Now let $v(x) + p(x) - q(x)$. Then, by linearity, $v(x)$ is a cubic polynomial that satisfies

$$v(x_0) = 0 \qquad v(x_2) = 0$$
$$v'(x_1) = 0 \qquad v''(x_1) = 0.$$

Furthermore, because $v(x)$ is a cubic polynomial, there exists constants $a_0, a_1, a_2,$ and a_3 such that

$$\begin{aligned} v(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 \\ v'(x) &= a_1 + 2a_2 x + 3a_3 x^2 \\ v''(x) &= 2a_2 + 6a_3 x. \end{aligned}$$

Then,

$$v''(x_1) = 2a_2 + 6a_3 x_1 = 0$$

which implies

$$a_2 = -3a_3.$$

Using this expression in our first derivative yields

$$v'(x_1) = a_1 - 6a_3 x_1^2 + 3a_3 x_1^2 = 0$$

which implies

$$a_1 = 3a_3 x_1^2.$$

Finally,

$$\begin{aligned} v(x) &= a_0 + 3a_3 x_1^2 x - 3a_3 x_1 x^2 + a_3 x^3 \\ &= a_0 + 3a_3 x_1^2 x - 3a_3 x_1 x^2 + a_3 x^3 - a_3 x_1^3 + a_3 x_1^3 \\ &= (a_0 + a_3 x_1^3) - a_3 (x - x_1)^3. \end{aligned}$$

Then, using our last constraints, we have the system

$$\begin{aligned} v(x_0) &= (a_0 + a_3 x_1^3) - a_3 (x_0 - x_1)^3 = 0 \\ v(x_2) &= (a_0 + a_3 x_1^3) - a_3 (x_2 - x_1)^3 = 0 \end{aligned}$$

which yields

$$a_3 (x_0 - x_1)^3 = a_3 (x_2 - x_1)^3.$$

Then, because $x_0 \neq x_2$, we must have $a_3 = 3$ which implies $a_0 = 0$. Therefore,

$$v(x) = 0$$

and so we must have

$$p(x) = q(x)$$

showing the uniqueness of our polynomial.

(ii) Derive a formula for $p(x)$.

We know $p(x)$ has the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

for some constants a_0, a_1, a_2, and a_3. Then, using our second derivative information, we have

$$p''(x_1) = 2a_2 + 6a_3 x_1 = f''(x_1)$$

which implies

$$a_2 = \frac{1}{2} f''(x_1) - 3a_3 x_1.$$

Then, using our first derivative information, we have

$$p'(x_1) = a_1 + 2\left(\frac{1}{2} f''(x_1) - 3a_3 x_1\right) x_1 + 3a_3 x_1^2 = f'(x_1)$$

which implies

$$a_1 = f'(x_1) - f''(x_1)x_1 + 3a_3 x_1^2.$$

Next, we can use our function information to get the system

$$p(x_0) = a_0 + f'(x_1)x_0 - f''(x_1)x_0 x_1 + 3a_3 x_0 x_1^2 + \frac{1}{2} f''(x_1)x_0^2 - 3a_3 x_0^2 x_1 + a_3 x_0^3 = f(x_0)$$
$$p(x_2) = a_0 + f'(x_1)x_2 - f''(x_1)x_1 x_2 + 3a_3 x_1^2 x_2 + \frac{1}{2} f''(x_1)x_2^2 - 3a_3 x_1 x_2^2 + a_3 x_2^3 = f(x_2)$$

which implies

$$a_3 = \frac{f(x_2) - f(x_0) + f'(x_1)(x_0 - x_2) - f''(x_1)x_1(x_0 - x_2) + \frac{1}{2} f''(x_1)(x_0^2 - x_2^2)}{3x_1^2(x_2 - x_0) + 3x_1(x_0^2 - x_2^2) + x_2^3 - x_0^3}$$
$$a_0 = f(x_0) - f'(x_1)x_0 + f''(x_1)x_0 x_1 + 3a_3 x_0 x_1^2 - \frac{1}{2} f''(x_1)x_0^2 + 3a_3 x_0^2 x_1 - a_3 x_0^3.$$

Now, we construct our polynomial by first computing a_3, a_0, a_1, and a_2 in that order and then plugging them into our polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

(iii) Let $x_0 = -1, x_1 = 0$, and $x_2 = 1$. Assuming $f(x) \in C^4[-1, 1]$, show that for $x \in [-1, 1]$,

$$f(x) - p(x) = \frac{x^4 - 1}{4!} f^4(\eta_x)$$

for some $\eta_x \in [-1, 1]$.

Suppose we have a fixed $t \in [-1, 1]$ such that $t \neq x_0$ and $t \neq x_2$. Next, define

$$G(x) = E(x) - \frac{x^4 - 1}{t^4 - 1} E(t)$$

where $E(x) = f(x) - p(x)$. Then, by construction, we have $G(x_0) = G(x_2) = G(t) = 0$. Next, using Rolle's theorem, we have must have at least two roots to $G'(x)$. But, $G'(x_1) = 0$ by construction and so as long as the two roots from Rolle's theorem do not coincide with x_1, $G'(x)$ has three roots (Note it is highly unlikely that x_1 will be the point given from Rolle's theorem). Then, using Rolle's theorem again implies that $G''(x)$ also has two roots that are not equal to x_1. So, because $G''(x_1) = 0$, we know $G''(x)$ also has at least three roots. Using Rolle's theorem twice more tells us that $G^{(3)}(x)$ has two roots which implies $G^{(4)}(x)$ has one root; call this root $\eta_x \in [-1, 1]$. Then, we have

$$\begin{aligned} G^{(4)} &= E^{(4)}(\eta_4) - \frac{4!}{t^4 - 1} E(t) \\ &= f^{(4)}(\eta_x) - \frac{4!}{t^4 - 1} E(t) = 0 \end{aligned}$$

which implies

$$E(t) = \frac{t^4 - 1}{4!} f^{(4)}(\eta_x).$$

Then, if we rename t to x, we have

$$E(x) = f(x) - p(x) = \frac{x^4 - 1}{4!} f^{(4)}(\eta_x).$$

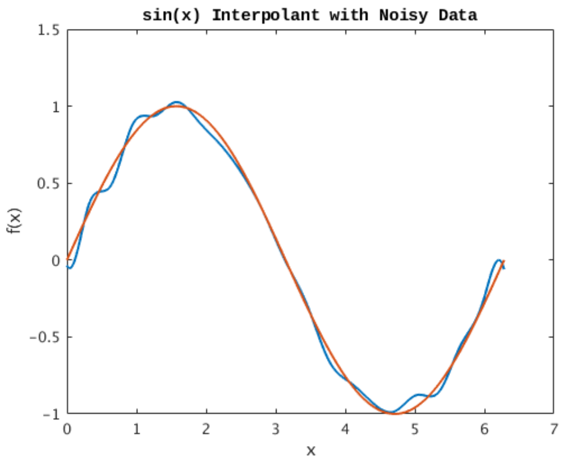
3. Suppose we have m data points $\{(t_i, y_i)\}_{i=1}^m$, where the t-values all occur in some interval $[x_0, x_n]$. We subdivide the interval $[x_0, x_n]$ into n subintervals $\{[x_k, x_{k+1}]_{k=0}^{n-1}\}$ of equal length h and attempt to choose a spline function $s(x)$ with nodes at $\{x_k\}_{k=0}^n$ in such a way so that

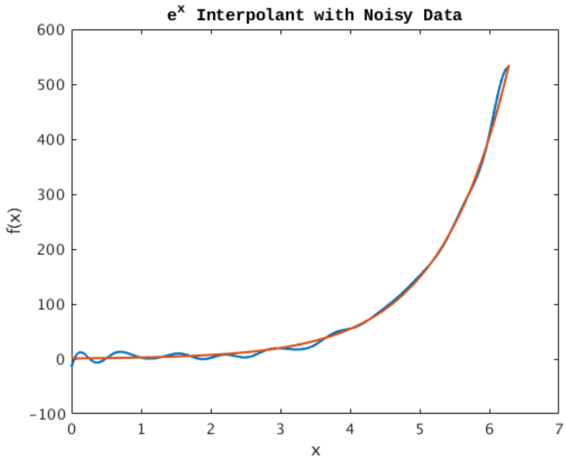
$$\sum_{i=1}^{m} |y_i - s(t_i)|^2$$

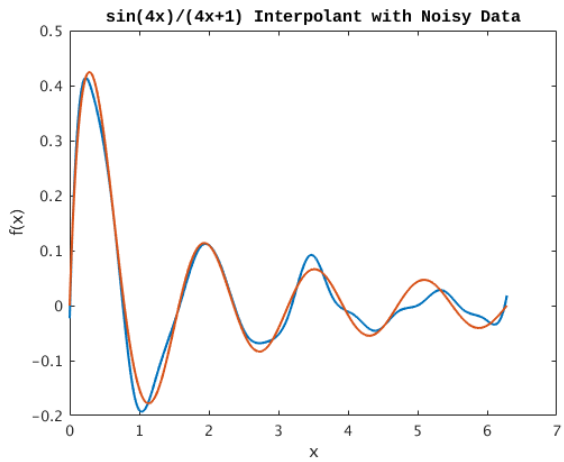
is *minimized*.

(i) My B-Spline code is given at the end of the document

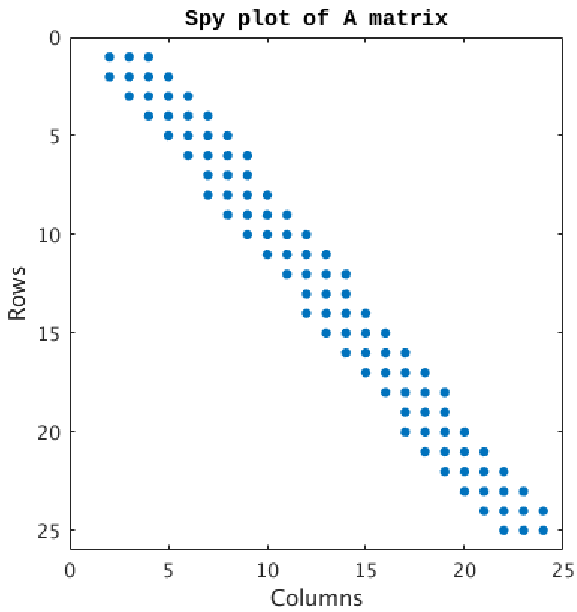
(ii) Using my code, I was able to produce the following noisy plots:







(iii) My spy plot is given below. We can see that we have nice and relatively uniform sparse structure throughout the matrix. Each row has at most 4 nonzero entries that are all consecutive. Furthermore, each new row either shifts the 4 consecutive non entries to the right by 1 or not at all.



Code Used

```
%%
% B-Spline interpolation routine for Problem 3 of HW08
% Author: Caleb Jacobs
% Date last modified: 20-10-2021

%% Clean workspace
clear

%% Settings
format longE
a = 0;             % Left side of interval
b = 2*pi;          % Right side of interval
n = 20;            % Number of intervals
m = 25;            % Number of data points

%% Initialize data
h  = (b - a) / m;                % Step size
x  = linspace(a, b, n + 1)       % Interval boundaries
t  = linspace(a, b, m);          % Nodes

A = constructA(t, x);            % Find A matrix for computing weights

%% Interpololate sin(x)
f = @(x) sin(x);                 % Function definition
bb = f(t)' + 0.05*randn(m, 1);   % Data function values
c = A \ bb;                      % Interpolation weights

figure(1)
T = linspace(a, b, 1000);
plot(T, S(T, x, c), T, f(T), 'LineWidth', 1.5)
title('sin(x) Interpolant with Noisy Data')
xlabel('x')
ylabel('f(x)')

%% Interpolate exp(x)
f = @(x) exp(x);                 % Function definition
bb = f(t)' + 6*randn(m, 1);      % Data function values
c = A \ bb                       % Interpolation weights

figure(2)
T = linspace(a, b, 1000);
plot(T, S(T, x, c), T, f(T), 'LineWidth', 1.5)
title('e^x Interpolant with Noisy Data')
xlabel('x')
ylabel('f(x)')

%% Interpolate sin(x)/x
```

```
f = @(x) sin(4*x) ./ (4*x + 1); % Function definition
bb = f(t)' + 0.02*randn(m, 1);  % Data function values
c = A \ bb                      % Interpolation weights

figure(3)
T = linspace(a, b, 1000);
plot(T, S(T, x, c), T, f(T), 'LineWidth', 1.5)
title('sin(4x)/(4x+1) Interpolant with Noisy Data')
xlabel('x')
ylabel('f(x)')

%% Plot matrix structure
figure(4)
spy(A)
title('Spy plot of A matrix')
xlabel('Columns')
ylabel('Rows')

%% Routine definitions
% Basis spline
function val = B(t, x, i)
    h = x(2) - x(1);
    n = length(x);

    if i - 2 > 0 && t < x(i - 2)
        val = 0;
    elseif i - 2 > 0 && x(i - 2) <= t && t <= x(i - 1)
        val = (t - x(i - 2)).^3 / h^3;
    elseif i - 1 > 0 && x(i - 1) <= t && t <= x(i)
        xi  = x(i - 1);
        val = 1 + 3*(t - xi) / h + 3*(t - xi).^2 / h^2 - 3*(t - xi)^3 / h^3;
    elseif i + 1 <= n && x(i) <= t && t <= x(i + 1)
        xi  = x(i + 1);
        val = 1 + 3*(xi - t) / h + 3*(xi - t).^2 / h^2 - 3*(xi - t)^3 / h^3;
    elseif i + 2 <= n && x(i + 1) <= t && t <= x(i + 2)
        val = (x(i + 2) - t).^3 / h^3;
    else
        val = 0;
    end
end

% Get interval indices of data
function idx = getIdx(t, x)
    m = length(t);          % Number of nodes
    n = length(x);          % Number of intervals + 1

    idx = ones(m, 1);
    for i = 1 : m
        for j = 2 : n
            if t(i) < x(j)
                break
```

```
            end
        end
        idx(i) = j - 1;
    end
end

% Constuct A matrix
function A = constructA(t, x)
    m = length(t);                  % Number of nodes
    n = length(x);                  % Number of intervals + 1
    h = x(2) - x(1);                % Data spacing

    idx = getIdx(t, x) + 2;         % Interval indices of data

    x = [(x(1) - h*[2 1]) x (x(n) + h*[1 2])];  % Append extra intervals

    A = spalloc(m, n + 2, 4 * m);   % Initialize sparse A
    for i = 1 : m
        for j = -1 : 2
            A(i, idx(i) + j) = B(t(i), x, idx(i) + j);
        end
    end
end

% Evaluate Interpolant
function f = S(t, x, c)
    m = length(t);
    h = x(2) - x(1);
    n = length(x);
    x = [(x(1) - h*[2 1]) x (x(n) + h*[1 2])];  % Append extra intervals
    n = length(x);

    f = zeros(m, 1);
    for j = 1 : m
        for i = 1 : n - 1
            f(j) = f(j) + c(i) * B(t(j), x, i);
        end
    end
end
```