APPM 5600 - Homework 8

Eappen Nelluvelil

October 22, 2021

1. (a) 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.

   The second-degree polynomial that interpolates f at x_0, x_1, and x_2, constructed via Newton's divided differences, is given by

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1),$$

   where

$$\begin{aligned}
f[x_0] &= f(x_0) \\
&= e^{-0.2}, \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} \\
&= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\
&= \frac{e^0 - e^{-0.2}}{0 + 0.2} \\
&= \frac{1 - e^{-0.2}}{0.2} \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\
&= \frac{\frac{f[x_2] - f[x_1]}{x_2 - x_1} - \frac{f[x_1] - f[x_0]}{x_1 - x_0}}{x_2 - x_0} \\
&= \frac{\frac{e^{0.2} - 1}{0.4} - \frac{1 - e^{-0.2}}{0.2}}{0.2 - 0} \\
&= \frac{e^{0.2} + e^{-0.2} - 2}{0.08}
\end{aligned}$$

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

   We define the error to be

$$E(x) = f(x) - p(x).$$

   Because our nodes are distinct and $f \in C^3([-0.2, 0.2])$, there exists an $\alpha \in [-0.2, 0.2]$ such that

$$\begin{aligned}
E(x) &= \frac{x^3 - 0.04x}{3!} f^{(3)}(\alpha) \\
&= \frac{x^3 - 0.04x}{3!} e^{\alpha}
\end{aligned}$$

for $x \in [-0.2, 0.2]$. We see that

$$\begin{aligned}
\max_{x\in[-0.2,0.2]} |E(x)| &= \max_{x\in[-0.2,0.2]} \left| \frac{x^3 - 0.04x}{3!} e^{\alpha} \right| \\
&\leq \max_{x\in[-0.2,0.2]} \left| \frac{x^3 - 0.04x}{3!} \right| \max_{\alpha\in[-0.2,0.2]} \left| e^{\alpha} \right| \\
&= \left(\frac{\left(-\sqrt{\frac{0.04}{3}}\right)^3 - 0.04\left(-\sqrt{\frac{0.04}{3}}\right)}{3!} \right) e^{0.2} \\
&\approx 6.26824187745 \times 10^{-4}
\end{aligned}$$

is an upper bound on our error E for $x \in [-0.2, 0.2]$.

(c) Compute the error $E(0.1) = f(0.1) - p_2(0.1)$. How does this compare with the error bound?
We compute $|E(0.1)| = |f(0.1) - p_2(0.1)|$ and see that it is approximately $5.13772099668 \times 10^{-4}$. This is less than the error bound we computed in 2(b), which is expected.

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

$$\begin{aligned}
p(x_0) &= f(x_0) \\
p(x_2) &= f(x_2) \\
p'(x_1) &= f'(x_1) \\
p''(x_1) &= f''(x_1),
\end{aligned}$$

where $f(x)$ is a given function and $x_0 \neq x_2$.
Without loss of generality, we assume the nodes are ordered such that $x_0 < x_1 < x_2$. For the sake of contradiction, suppose there exist two cubic polynomials, p and q, that satisfy the above conditions and are such that $p \neq q$. Define $r(x) = p(x) - q(x)$, which is not identically zero. We know that

i. $r(x_0) = 0$
ii. $r(x_2) = 0$
iii. $r'(x_1) = 0$
iv. $r''(x_1) = 0$

Since r is the difference of two cubic polynomials on $[x_0, x_2]$, we know that r is continuous on $[x_0, x_2]$ and differentiable on (x_0, x_2). By Rolle's theorem, there exists a point $\widetilde{x}_1 \in (x_0, x_2)$ such that $r'(\widetilde{x}_1) = 0$. Since r' is at most a second-degree polynomial, we can write $r'(x) = a(x - x_1)(x - \widetilde{x_1})$, where a is some constant. Then, $r''(x) = a((x - \widetilde{x}_1) + (x - x_1)) = a(2x - x_1 - \widetilde{x}_1)$. We know that $r''(x_1) = a(2x_1 - x_1 - \widetilde{x}_1) = a(x_1 - \widetilde{x}_1) = 0$, so it must be the case that either $a = 0$ or $x_1 = \widetilde{x}_1$. We consider the two cases:

i. **Case 1:** $a = 0$.
In this case, $r'(x) = 0$, which we can integrate to obtain $r(x) = b$, where b is some constant. Since we must have that $r(x_0) = 0$ and $r(x_2) = 0$, it must be the case that $b = 0$. Thus, $r(x) = 0$, which is a contradiction as we assumed earlier that r was not identically zero.

ii. **Case 2:** $x_1 = \widetilde{x}_1$.
In this case, $r'(x) = a(x - x_1)^2$, which we can integrate to obtain that $r(x) = a\frac{(x-x_1)^3}{3} + c$, where c is some constant. Since we must have that $r(x_0) = a\frac{(x_0-x_1)^3}{3} + c = 0$ and $r(x_2) = a\frac{(x_2-x_1)^3}{3} + c = 0$, we arrive at the following condition for r:

$$\begin{aligned}
a\frac{(x_0 - x_1)^3}{3} + c = a\frac{(x_2 - x_1)^3}{3} + c &\implies (x_0 - x_1)^3 = (x_2 - x_1)^3 \\
&\implies x_0 - x_1 = x_2 - x_1 \\
&\implies x_0 = x_2,
\end{aligned}$$

which is a contradiction as we assumed that $x_0 \neq x_2$.

Since we arrive at a contradiction in both cases, it must be the case that r is identically zero, i.e., $p = q$. Thus, there exists at most one polynomial that satisfies the above conditions, as desired.

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

We want to write p in the form

$$p(x) = f(x_0) H_0(x) + f(x_2) H_2(x) + f'(x_1) K_1(x) + f''(x_1) J_1(x),$$

where

i. $H_0(x_0) = 1$, $H_0(x_2) = 0$, $H_0'(x_1) = 0$, and $H_0''(x_1) = 0$,
ii. $H_2(x_0) = 0$, $H_2(x_2) = 1$, $H_2'(x_1) = 0$, and $H_2''(x_1) = 0$,
iii. $K_1(x_0) = 0$, $K_1(x_2) = 0$, $K_1'(x_1) = 1$, and $K_1''(x_1) = 0$, and
iv. $J_1(x_0) = 0$, $J_1(x_2) = 0$, $J_1'(x_1) = 0$, and $J_1''(x_1) = 1$.

We derive the polynomials as follows:

i. $H_0(x)$

From the last two conditions imposed on H_0, it must be the case that $H_0'(x) = a(x - x_1)^2$, where a is a constant. From this, we obtain that $H_0(x) = b(x - x_1)^3 + c$, where b and c are constants. Imposing the first two constraints on H_0, we have that

$$b(x_0 - x_1)^3 + c = 1,$$
$$b(x_2 - x_1)^3 + c = 0,$$

from which we obtain that $b = \frac{1}{(x_0-x_1)^3-(x_2-x_1)^3}$ and $c = -\frac{(x_2-x_1)^3}{(x_0-x_1)^3-(x_2-x_1)^3}$. Thus,

$$H_0(x) = \frac{(x - x_1)^3 - (x_2 - x_1)^3}{(x_0 - x_1)^3 - (x_2 - x_1)^3}.$$

ii. $H_2(x)$

From the last two conditions imposed on H_2, it must be the case that $H_2'(x) = a(x - x_1)^2$, where a is a constant. From this, we obtain that $H_2(x) = b(x - x_1)^3 + c$, where b and c are constants. Imposing the first two constraints on H_2, we have that

$$b(x_0 - x_1)^3 + c = 0,$$
$$b(x_2 - x_1)^3 + c = 1,$$

from which we obtain that $b = \frac{1}{(x_2-x_1)^3-(x_0-x_1)^3}$ and $c = -\frac{(x_0-x_1)^3}{(x_2-x_1)^3-(x_0-x_1)^3}$. Thus,

$$H_2(x) = \frac{(x - x_1)^3 - (x_0 - x_1)^3}{(x_2 - x_1)^3 - (x_0 - x_1)^3}.$$

iii. $K_1(x)$

From the last two conditions imposed on K_1, it must be the case that $K_1''(x) = a(x - x_1)$, where a is a constant. From this, we obtain that $K_1'(x) = b(x - x_1)^2 + c$ and $K_1(x) = d(x - x_1)^3 + cx + d$, where b, c, d, and e are constants. We see that $c = 1$, and imposing the first two constraints on K_1, we have that

$$d(x_0 - x_1)^3 + x_0 + e = 0,$$
$$d(x_2 - x_1)^3 + x_2 + e = 0,$$

from which we obtain that $d = \frac{(x_0-x_2)}{(x_2-x_1)^3-(x_0-x_1)^3}$ and $e = -x_0 - \frac{(x_0-x_2)(x_0-x_1)^3}{(x_2-x_1)^3-(x_0-x_1)^3}$.

Thus,

$$K_1(x) = \frac{(x_0 - x_2)}{(x_2 - x_1)^3 - (x_0 - x_1)^3}(x - x_1)^3 + x + \left(-x_0 - \frac{(x_0 - x_2)(x_0 - x_1)^3}{(x_2 - x_1)^3 - (x_0 - x_1)^3}\right).$$

(c) $J_1(x)$

From the last condition imposed on J_1, it must be the case that $J_1''(x) = a(x - x_1) + 1$, where a is a constant. From this, we obtain that $J_1'(x) = b(x - x_1)^2 + x + d$, where b and d are constants. We impose the third condition to obtain that $d = -x_1$. We integrate once more to obtain that $J_1(x) = e(x - x_1)^3 + \frac{x^2}{2} - x_1 x + f$, where e and f are constants. By imposing the first constraints on J_1, we have that

$$e(x_0 - x_1)^3 + \frac{x_0^2}{2} - x_1 x_0 + f = 0,$$
$$e(x_2 - x_1)^3 + \frac{x_2^2}{2} - x_1 x_2 + f = 0,$$

from which we obtain that

$$e = \frac{\frac{1}{2}\left(x_0^2 - x_2^2\right) + x_1(x_2 - x_0)}{(x_2 - x_1)^3 - (x_0 - x_1)^3},$$
$$f = x_1 x_0 - \frac{x_0^2}{2} - \left(\frac{\frac{1}{2}\left(x_0^2 - x_2^2\right) + x_1(x_2 - x_0)}{(x_2 - x_1)^3 - (x_0 - x_1)^3}\right)(x_0 - x_1)^3.$$

Thus, $J_1(x)$ is given by

$$\frac{\frac{1}{2}\left(x_0^2 - x_2^2\right) + x_1(x_2 - x_0)}{(x_2 - x_1)^3 - (x_0 - x_1)^3}(x - x_1)^3 + \frac{x^2}{2} - x_1 x + \left(x_1 x_0 - \frac{x_0^2}{2} - \left(\frac{\frac{1}{2}\left(x_0^2 - x_2^2\right) + x_1(x_2 - x_0)}{(x_2 - x_1)^3 - (x_0 - x_1)^3}\right)(x_0 - x_1)^3\right).$$

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

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

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

We define the error to be

$$E(x) = f(x) - p(x),$$

and we define $\psi(x) = x^4 - 1$ and $G(x) = E(x) - \frac{\psi(x)}{\psi(t)}E(t)$, where $t \neq x_0, x_1, x_2$ and $t \in [-1,1]$. We know that

i. G has four continuous derivatives,
ii. $G(t) = 0$, and
iii. $G(x_0) = 0$ and $G(x_2) = 0$.

Thus, G has three roots in $[-1,1]$ at x_0, t, and x_2. By Rolle's theorem, G' has two roots, s_1 and s_2, in $(-1,1)$ and are such that $s_1 \in (-1,t)$ and $s_2 \in (t,1)$. Furthermore, $G'(x_1) = 0$, so x_1 is another root of G'. Thus, G' has three roots, s_1, s_2, and x_1, in $(-1,1)$. By another application of Rolle's theorem, G'' has two roots, w_1 and w_2, and we also know that $G''(x_1) = 0$. Thus, G'' has three roots, w_1, w_2, and x_1, in $(-1,1)$. By inductively applying Rolle's theorem, we see that $G^{(3)}$ has two roots in $(-1,1)$ and $G^{(4)}$ has one root in $(-1,1)$, which we denote η_x. We know that at η_x,

$$0 = G^{(4)}(\eta_x) = f^{(4)}(\eta_x) - \frac{4!}{t^4 - 1}E(t),$$

which we can rearrange to obtain

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

as desired.

3. (a) 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 sub-intervals $\{[x_k, x_{k+1}]\}_{k=0}^{n-1}$ 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**.

This can be formatted as a linear least squares problem with a coefficient matrix that has a banded structure. The key to the structure of the data matrix for spline data fitting is the fact that cubic splines may be represented as linear combinations of B-splines which have **only** local support on the interval.

Note that $B_{-1}(t)$ and $B_{n+1}(t)$ are not shown, but they need to be included. So, written in terms of the $\{B_i(t)\}$, the cubic spline $s(t)$ appears as

$$s(t) = \sum_{i=0}^{n} \alpha_i B_i(t),$$

with

$$h^3 B_i(t) = \begin{cases} (t - x_{i-2})^3 & t \in [x_{i-2}, x_{i-1}] \\ h^3 + 3h^2 (t - x_{i-1}) + 3h (t - x_{i-1})^2 - 3 (t - x_{i-1})^3 & t \in [x_{i-1}, x_i] \\ h^3 + 3h^2 (x_{i+1} - t) + 3h (x_{i+1} - t)^2 - 3 (x_{i+1} - t)^3 & t \in [x_i, x_{i+1}] \\ (x_{i+2} - t)^3 & t \in [x_{i+1}, x_{i+2}] \\ 0 & \text{otherwise.} \end{cases}$$

Note that if $t_k \in [x_i, x_{i+1}]$, then

$$\begin{aligned} s(t_k) &= \sum_{i=0}^{n} c_i B_i(t_k) \\ &= c_{i-1} B_{i-1}(t_k) + c_i B_i(t_k) + c_{i+1} B_{i+1}(t_k) + c_{i+2} B_{i+2}(t_k). \end{aligned}$$

The data matrix will never have more than four nonzero entries per row.

The indexing of the $B_i(t)$ may be different from what you have seen. However, this will give the same results. Also note that $B_{n+2}(t)$ will not be needed since it vanishes for any $t \in [x_0, x_n]$.

i. Write a program that, given an interval $[a, b]$, will construct n equally spaced grid point $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ and data points $\{(t_i, f_i) \mid i = 1, \ldots, m\}$ with $t_i \in [a, b]$ and with $f_i = f(t_i)$ for some given function f. The routine should then set up the matrix $\mathbf{A}$ of coefficients and the right hand side $\mathbf{b}$ with $\mathbf{b}_i = f_i$. **Note:** Your $\mathbf{A}$ should be an $m \times (n+3)$ matrix with $m > n + 3$, where m is the number of data points and n is the number of intervals $[x_j, x_{j+1})$. it is probably best if you take the points t_i to be equally spaced in the interval (x_0, x_n), but with a much smaller spacing and then process these points in increasing order to construct the rows of $\mathbf{A}$ as you need them. The code should then solve the least squares problem

$$\min_{\mathbf{c}} ||\mathbf{b} - \mathbf{A}\mathbf{c}||_2.$$

You may use MATLAB's `qr` and backslash operator for this. The entries of $\mathbf{c}$ will be the coefficients $\{c_i | i = -1, \ldots, n+1\}$.
The input to the code sohuld be the interval $[a, b]$, i.e., the values of a and b, the number of intervals n (to specify $[x_{i-1}, x_i)$ for $i = 1, \ldots, n$), and a function handle f for a code to evaluate $f(t)$, where f is the function you wish to approximate.
The output of the code should be the coefficients $\mathbf{c}$ and the vector $\mathbf{x}$ of grid points x_i.
See attached code.

ii. Test your routine on data points generated from a variety of functions of a single variable plus noise (small random perturbations). Include the following example:

```
h1 = a stepsize smaller than x(j) - x(j-1)
t = [0:h1:2*pi];
y = sin(t)';
m = length(y);
b = y + 0.05*randn(m, 1);
```

Turn in the plots for this sin function, and also select two additional interesting ones to turn in. Plot the spline fit with the data points and the original (noiseless) function. Include legends, titles, and axis labels.
You will need to write a routine evaluate the spline at any argument $t \in [a, b]$ to make these plots.
See attached code for plots.

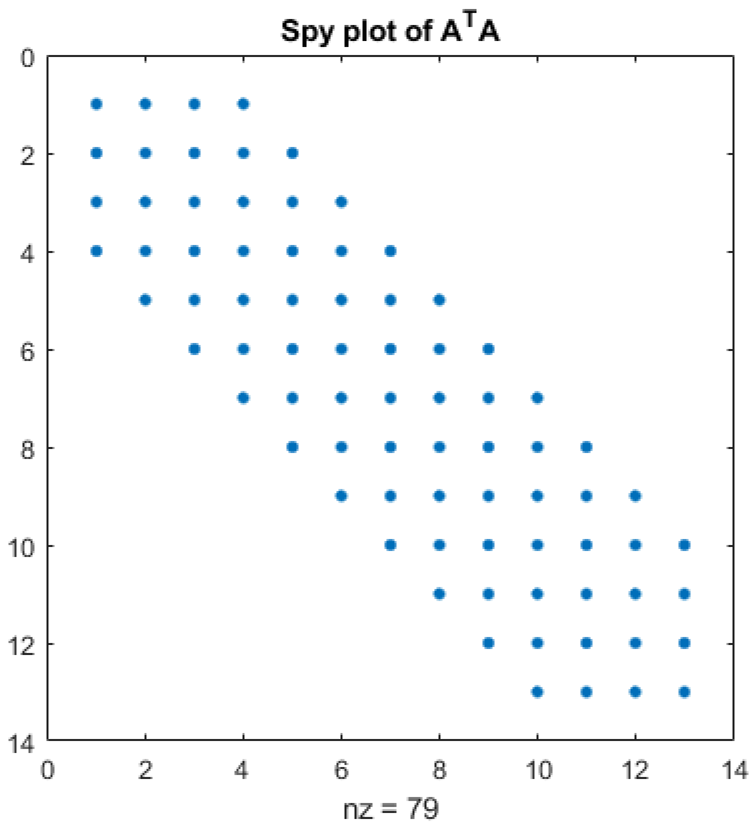
iii. Show a `spy` plot of the matrix $\mathbf{A}$ and also for the matrix $\mathbf{A}^T\mathbf{A}$. Notice the structure. We shall learn later in the course about how to take advantage of this structure.
From the `spy` plot of $\mathbf{A}^T\mathbf{A}$, we see that it is a banded matrix.

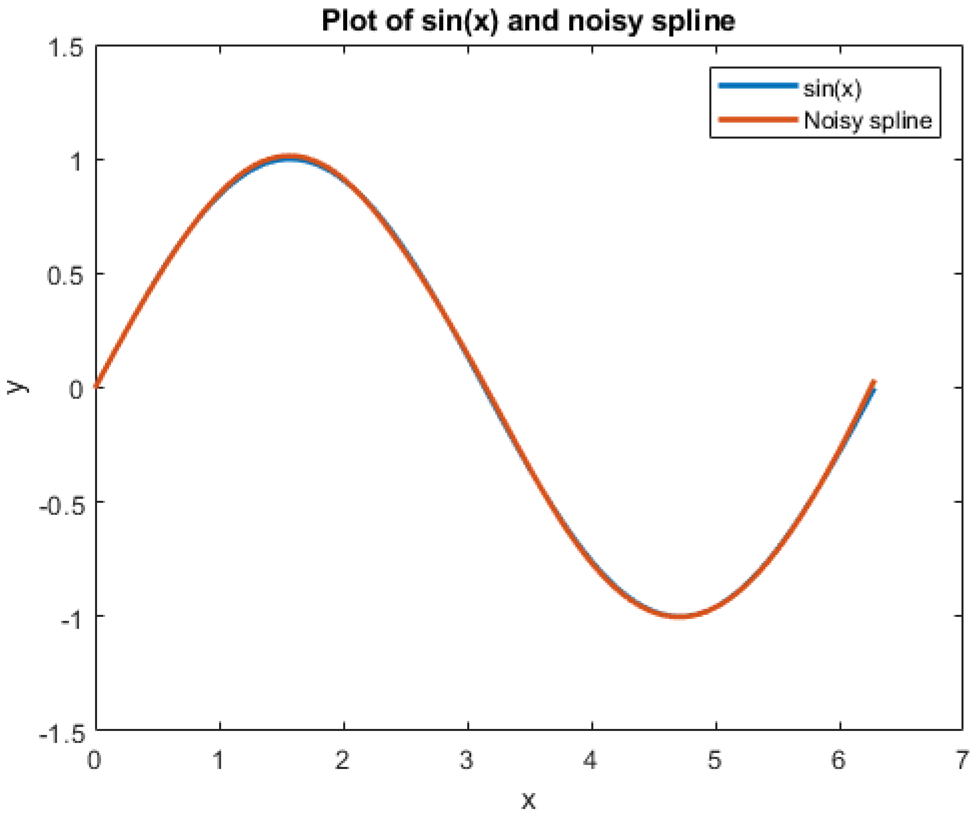
```
% Homework 8
```

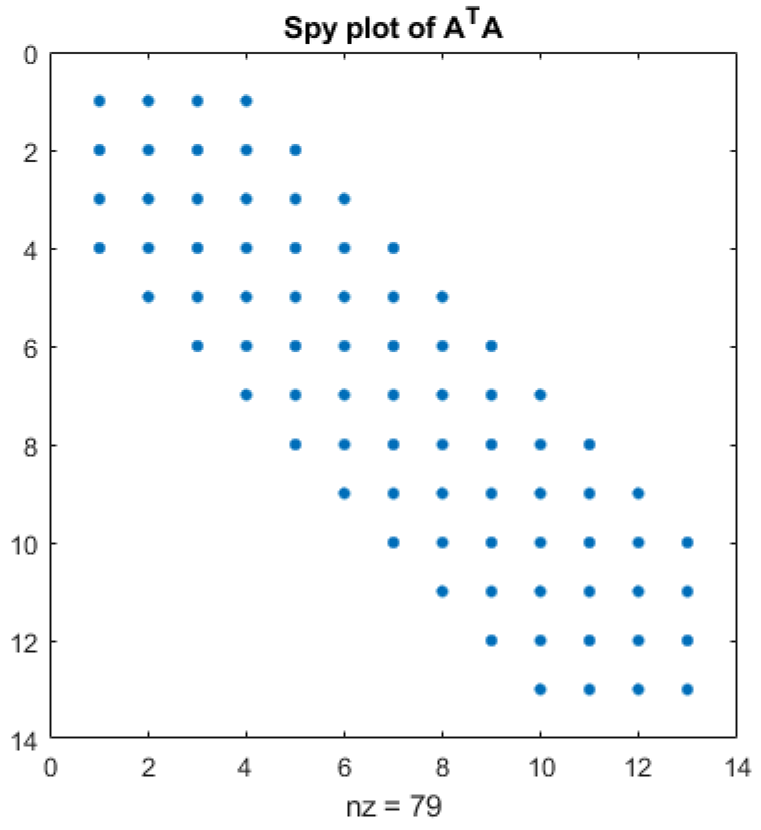
Problem 3

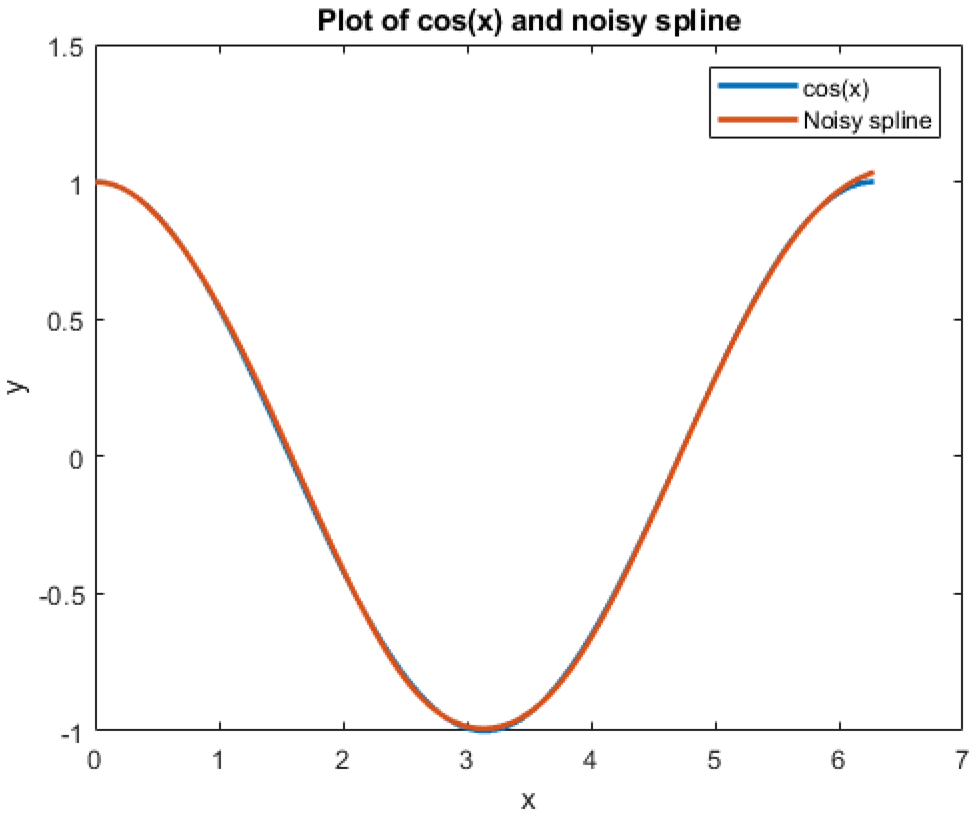
```
clc;
clear;
close all;

a = 0; b = 2*pi;
num_x_intervals = 10;
m = 200;
f = @(x) sin(x);
noise = 0.05 * randn(m, 1);

[c_2, x_2] = construct_A(a, b, num_x_intervals, m, f, noise);

t_plot_m = 1000;
[t_plot, f_t_plot_2]      = eval_spline(a, b, c_2, x_2, t_plot_m);

figure;
plot(t_plot, f(t_plot), "LineWidth", 2, "DisplayName", "sin(x)");
hold on;
plot(t_plot, f_t_plot_2, "LineWidth", 2, "DisplayName", "Noisy spline");
title("Plot of sin(x) and noisy spline");
xlabel("x");
ylabel("y");
legend;

% Second example
g = @(x) cos(x);

[c_4, x_4] = construct_A(a, b, num_x_intervals, m, g, noise);

[~, g_t_plot_2] = eval_spline(a, b, c_4, x_4, t_plot_m);

figure;
plot(t_plot, g(t_plot), "LineWidth", 2, "DisplayName", "cos(x)");
hold on;
plot(t_plot, g_t_plot_2, "LineWidth", 2, "DisplayName", "Noisy spline");
title("Plot of cos(x) and noisy spline");
xlabel("x");
ylabel("y");
legend;

% Third example
h = @(x) 1./(1+25*x.^(2));

[c_6, x_6] = construct_A(a, b, num_x_intervals, m, h, noise);

[~, h_t_plot_2] = eval_spline(a, b, c_6, x_6, t_plot_m);
```

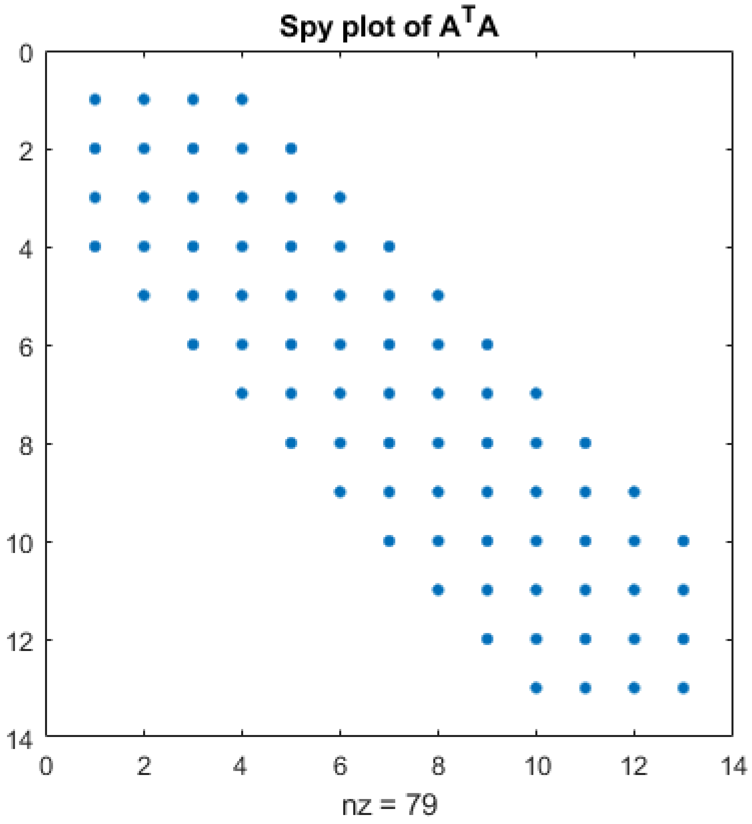
```
figure;
plot(t_plot, h(t_plot), "LineWidth", 2, "DisplayName", "1/(1+25x^{2})");
hold on;
plot(t_plot, h_t_plot_2, "LineWidth", 2, "DisplayName", "Noisy spline");
title("Plot of 1/(1+25x^{2}) and noisy spline");
xlabel("x");
ylabel("y");
legend;
```

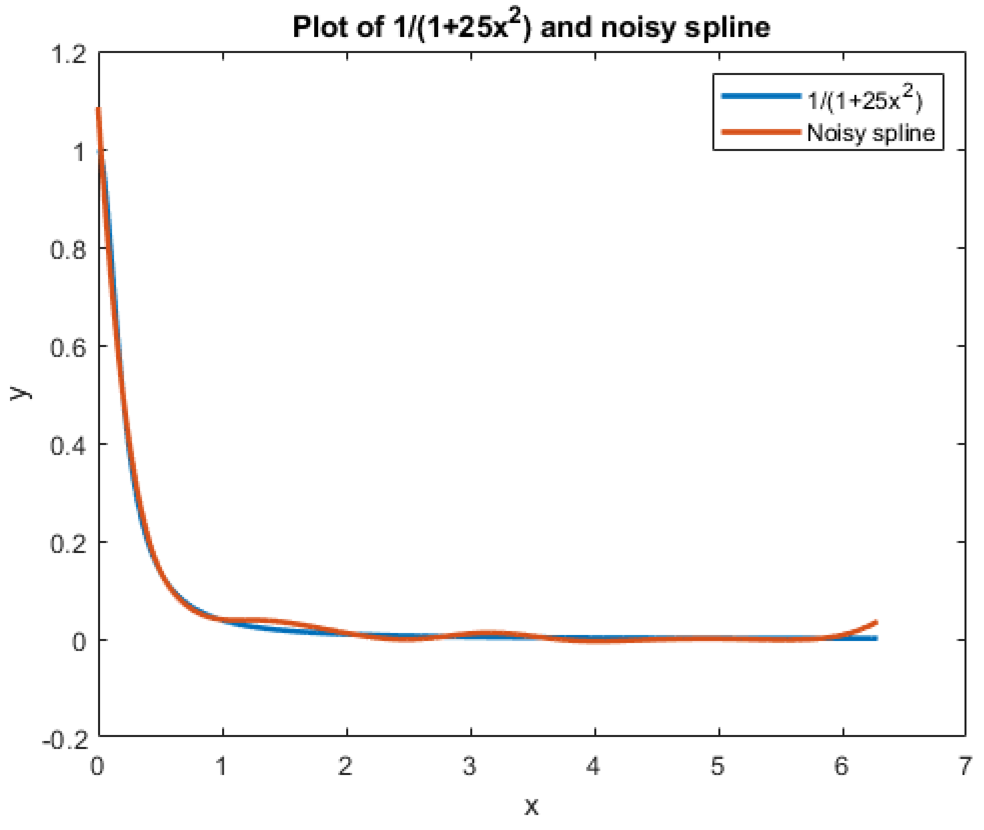



Plot of sin(x) and noisy spline
sin(x)
Noisy spline
y
x
1.5
1
0.5
0
-0.5
-1
-1.5
0
1
2
3
4
5
6
7


Spy plot of A^TA
nz = 79
0
2
4
6
8
10
12
14


Plot of cos(x) and noisy spline
cos(x)
Noisy spline
y
x
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7


Spy plot of A^TA
nz = 79
0
2
4
6
8
10
12
14



Published with MATLAB® R2021b

```
%{
%}
function [c, x] = construct_A(a, b, num_x_intervals, m, f, noise)
    x   = linspace(a, b, num_x_intervals + 1)';
    x_h = abs(x(2)-x(1));

    % Include one equally spaced point before x_0 and one equally spaced
    % point after x_n
    x = [x(1) - x_h; ...
         x; ...
         x(end) + x_h];

    t = linspace(a, b , m)';

    A = zeros(m, num_x_intervals + 3);

    for i = 1:m
        for j = 1:length(x)
            A(i, j) = eval_B_spline(x(j), t(i), x_h);
        end
    end

    figure;
    spy(A'*A);
    title("Spy plot of A^{T}A")

    rhs = f(t) + noise;
    c = A\rhs;
end
```

```
Not enough input arguments.

Error in construct_A (line 4)
    x   = linspace(a, b, num_x_intervals + 1)';
```

Published with MATLAB® R2021b

```
%{
    Function: eval_B_spline
    Inputs:
        center - center of B-spline
        t - data point to evaluate B-spline at
        h - spacing between B-spline centers
    Outputs:
        val - value of B-spline evaluated at t
%}

function val = eval_B_spline(center, t, h)
    x_i_m_2 = center - 2*h;
    x_i_m_1 = center - h;
    x_i_p_1 = center + h;
    x_i_p_2 = center + 2*h;

    B_spline_support = [x_i_m_2; x_i_m_1; center; x_i_p_1; x_i_p_2];

    % Determine which sub-interval t falls into
    bin = discretize(t, B_spline_support, 'IncludedEdge', 'left');

    if bin == 1
        val = (t - x_i_m_2)^(3);
    elseif bin == 2
        val = h^(3) + 3*h^(2)*(t - x_i_m_1) + 3*h*(t - x_i_m_1)^(2) - 3*(t -
 x_i_m_1)^(3);
    elseif bin == 3
        val = h^(3) + 3*h^(2)*(x_i_p_1 - t) + 3*h*(x_i_p_1 - t)^(2) -
 3*(x_i_p_1 - t)^(3);
    elseif bin == 4
        val = (x_i_p_2 - t)^(3);
    else
        val = 0;
    end

    val = val/(h^(3));
end
```

```
Not enough input arguments.

Error in eval_B_spline (line 12)
    x_i_m_2 = center - 2*h;
```

Published with MATLAB® R2021b

```
%{
    Function: eval_spline
    Inputs:
        a - left endpoint of interval (x(1) = a)
        b - right endpoint of interval (x(end) = b)
        c - vector of coefficients for B-splines
        x - coarse grid containing sub-intervals [x_j, x_{j+1}]
        t_plot_m - number of fine data points to use when plotting
    Outputs:
        t_plot - vector of fine data points (length(t_plot) = t_plot_m)
        f_t_plot - vector of spline evaluated at t_plot
%}
function [t_plot, f_t_plot] = eval_spline(a, b, c, x, t_plot_m)
    x_h = abs(x(2) - x(1));
    t_plot = linspace(a, b, t_plot_m)';
    f_t_plot = zeros(size(t_plot));

    for i = 1:t_plot_m
        for j = 1:length(x)
            f_t_plot(i) = f_t_plot(i) + c(j)*eval_B_spline(x(j), t_plot(i),
 x_h);
        end
    end
end
```

```
Not enough input arguments.

Error in eval_spline (line 14)
    x_h = abs(x(2) - x(1));
```

Published with MATLAB® R2021b