Ahmed Muhammad

MATH 4315

November 1, 2020

Project 2: Natural Cubic Spline and Discrete Fourier Series

This project will be exploring the Natural Cubic Spline and Discrete Fourier Series for the analysis of the average temperature in a city X. The natural cubic spline is a piecewise interpolation method that differs from polynomial interpolation methods because it is the summation of multiple lower order polynomials that piece together the entire approximation rather than a single, large degree polynomial. This is advantageous because of some problems with polynomial interpolation such as over-fitting, which can lead to unhelpful oscillating behavior in the interpolant. In this project, we are given 12 points for the average temperatures within a year, given as:

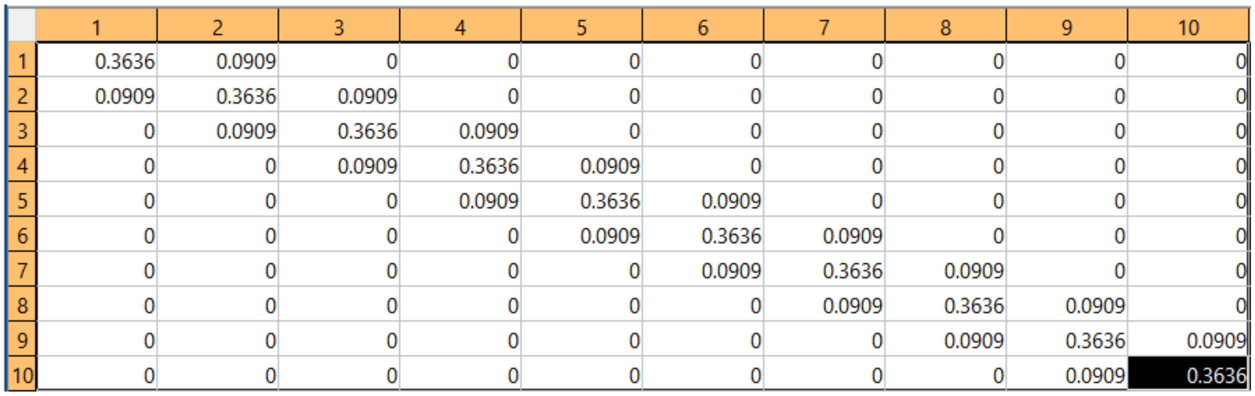
Data = [43.5, 48, 57.5, 65.5, 73, 81, 85.5, 85, 78, 67.5, 56, 47.5]

With these, we can begin the process of finding the natural cubic spline. The natural cubic spline requires solving a linear system for the coefficients of the spline. This linear system is shown below:

$$\begin{bmatrix} 2(h_1+h_2) & h_2 & & & \\ h_2 & 2(h_2+h_3) & h_3 & & \Large 0 \\ & \ddots & \ddots & \ddots & \\ \Large 0 & & \ddots & \ddots & h_{n-1} \\ & & & h_{n-1} & 2(h_n+h_{n-1}) \end{bmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{n-1} \end{pmatrix} = 6 \begin{pmatrix} \left(\frac{f_2-f_1}{h_2} - \frac{f_1-f_0}{h_1}\right) \\ \left(\frac{f_3-f_2}{h_3} - \frac{f_2-f_1}{h_2}\right) \\ \vdots \\ \left(\frac{f_n-f_{n-1}}{h_n} - \frac{f_{n-1}-f_{n-2}}{h_{n-1}}\right) \end{pmatrix}$$

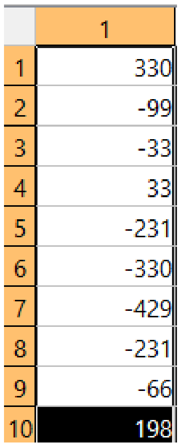
Figure 1

In my code, I solved the linear system as **A**(sigma) = **RHS**. The function generate_matrix() that generates **A** was written such that it can compute any **A** given any h and n—it is not specific to this problem. In our case, n = 12 and h = 1/11, and passing n-2 and h as arguments for the function generate_matrix() gives the following **A**:



	1	2	3	4	5	6	7	8	9	10
1	0.3636	0.0909	0	0	0	0	0	0	0	0
2	0.0909	0.3636	0.0909	0	0	0	0	0	0	0
3	0	0.0909	0.3636	0.0909	0	0	0	0	0	0
4	0	0	0.0909	0.3636	0.0909	0	0	0	0	0
5	0	0	0	0.0909	0.3636	0.0909	0	0	0	0
6	0	0	0	0	0.0909	0.3636	0.0909	0	0	0
7	0	0	0	0	0	0.0909	0.3636	0.0909	0	0
8	0	0	0	0	0	0	0.0909	0.3636	0.0909	0
9	0	0	0	0	0	0	0	0.0909	0.3636	0.0909
10	0	0	0	0	0	0	0	0	0.0909	0.3636

This matches the guidelines given in the project guide. Furthermore, I wrote another function called rhs() that computes the right-hand side of the linear system. This function, as seen in Figure 1, will require the data points as well and n and h as inputs. Thus, we pass the data, n, and h as arguments to the function rhs() and get the following output:

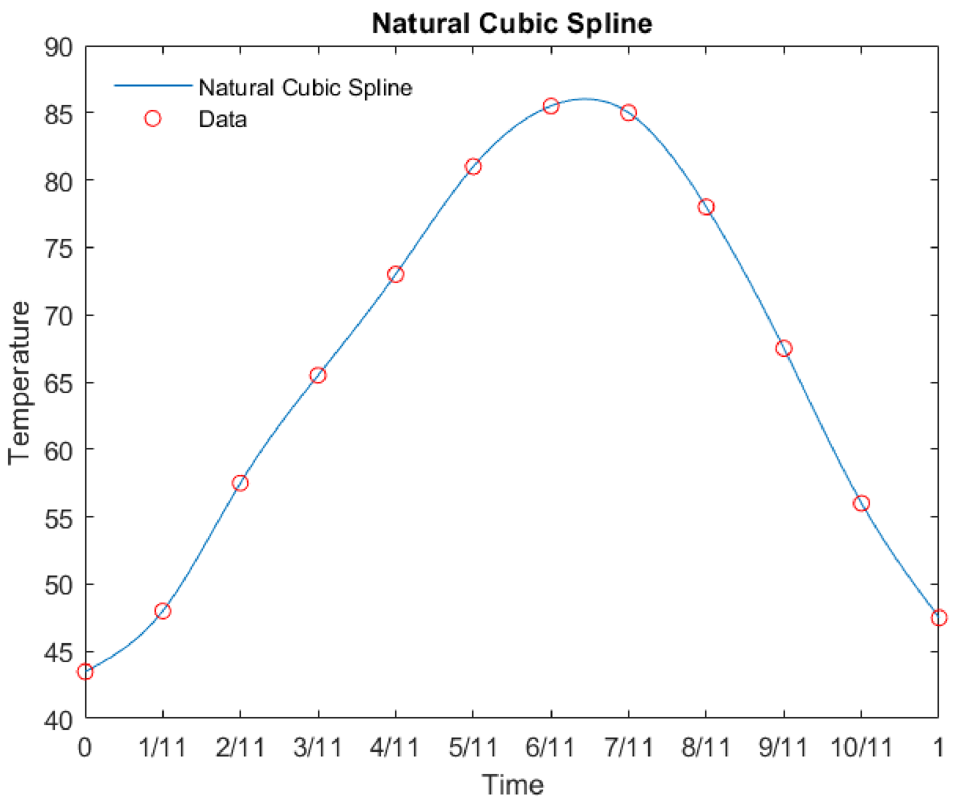


	1
1	330
2	-99
3	-33
4	33
5	-231
6	-330
7	-429
8	-231
9	-66
10	198

This output also matches guidelines given in the project guide, so we can continue with the next part of finding the natural cubic spline.

The linear system only solves for the sigma values sigma(1) until sigma(n-1). Sigma(0) and sigma(n) are both 0. Thus, in solving for sigma, we must be sure to only input the solution of the linear system from index 2 until index n-1. This is accomplished with the backslash solver as follows: sigma(2 : n - 1) = A \ RHS;. Sigma is pre-allocated as a 12x1 matrix of zeros beforehand so our first and last index will be zero, no matter what n we have.

Now that we have our sigma coefficient values, we are ready to use the provided function feval_spline() to generate the spline at an array of nodes. Our array of nodes will be 100 equally spaced points between 0 and 1, which we can generate with linspace(). linspace() will also generate the array of knots needed. In our project, this will be [0, 1/11, 2/11, 3/11, …10/11, 1]. Thus, we can pass the arguments knots, nodes, data values, sigma coefficients, and h (h = 1/11, simply the space between our knots) through feval_spline() to output the values of the spline at the array of 100 nodes. We can now plot this approximation, which is shown below:



We can see that our spline interpolates all the data points, shown in red circles. This approximation does reflect what I would expect the average temperature to look like over the entire year because there is not a lot of oscillation between the data points. This is because of the stiffer nature of cubic splines, and it does a good job interpolating the data. We can assume that summer falls in the middle of the plot, and the beginning and ending of the plot must be the beginnings and end of the year, as lower temperatures suggest winter.

Problem 2 of the project involves the discrete truncated Fourier series. The Fourier Series can be computed as follows:

$$f(x) \approx \frac{a_0}{2} + \sum_{i=1}^{N} (a_i \cos 2i\pi x + b_i \sin 2i\pi x)$$

Figure 2

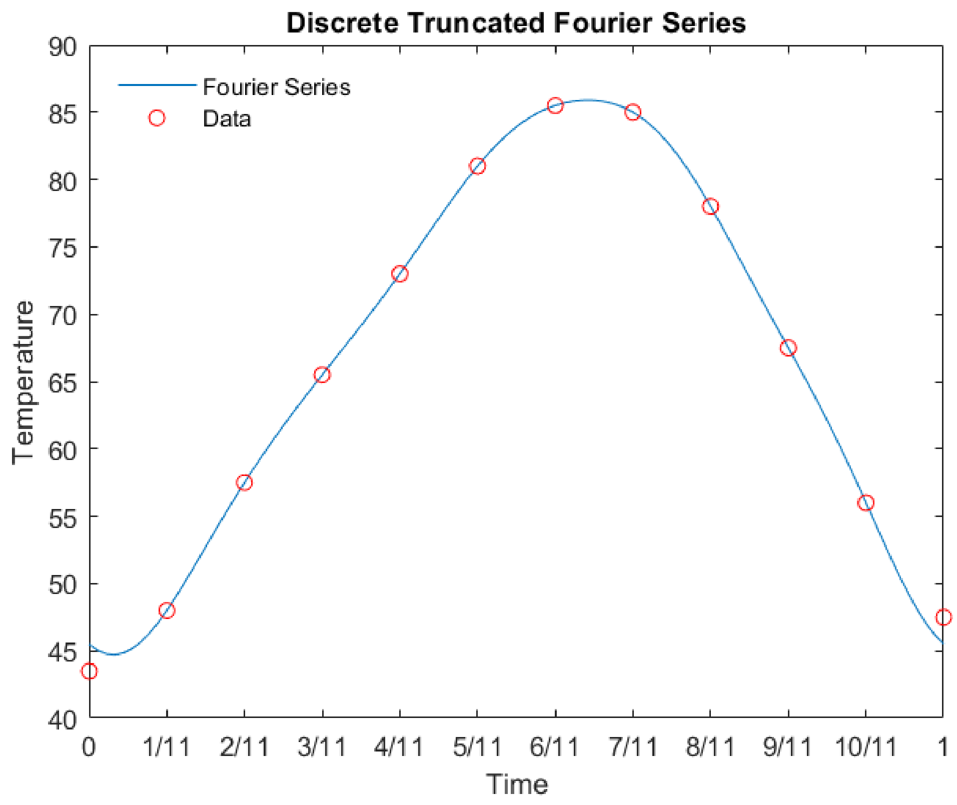
where:

$$a_i = 2\int_0^1 f(x) \cos 2i\pi x \, dx\,, \qquad b_i = 2\int_0^1 f(x) \sin 2i\pi x \, dx.$$

a(i) and b(i) can be approximated using a numerical integration procedure, which is provided in the project as fourier_coeff(). Once we have the coefficients, we can compute the right-hand series to approximate f(x). The project gives us N = 5, so we need 5 a(i) coefficients, 5 b(i) coefficients, and a0. fourier_coeff() takes the arguments:

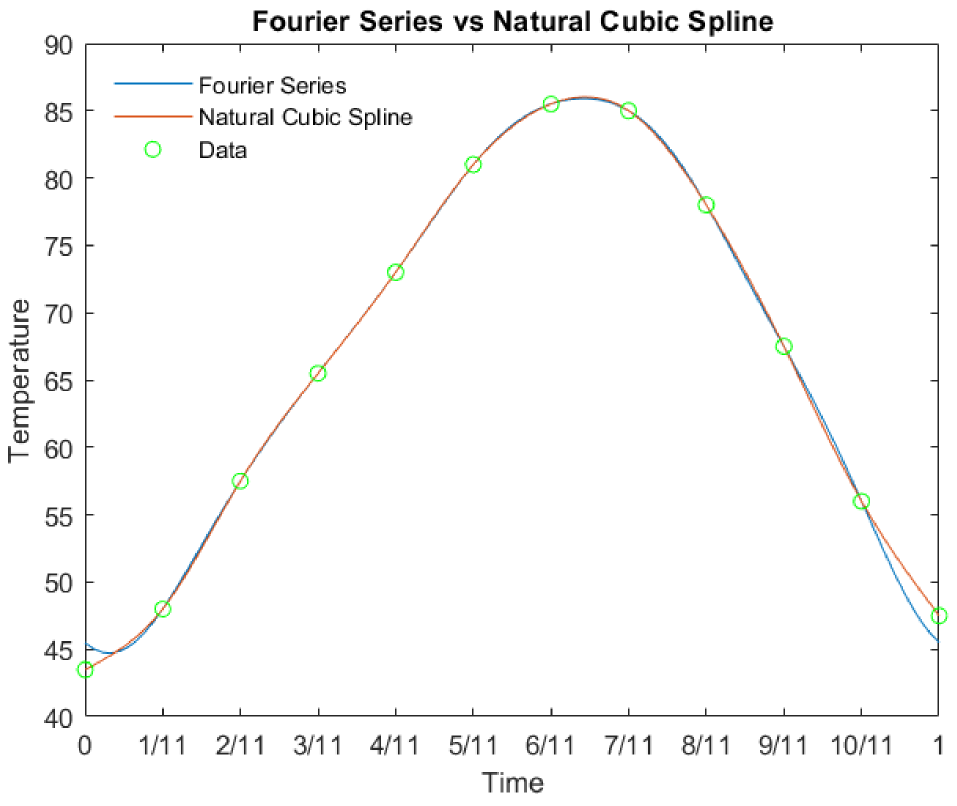
- Array of function values (our data points)
- i, the coefficient index (will range from 1 – 5 for a(i) and b(i)
- type 0, 1 (1 for a(i) coefficient and 0 for b(i) coefficient)

Thus, we can use two for loops to output a(i) and b(i) of size 5x1, as well as a0, all of which are needed to compute f(x) at 100 equally spaced nodes. I wrote another function named ahmed() which takes the arguments of a(i), b(i), nodes (100 equally spaced points between 0 and 1, as called for by the project guidelines), N (N = 5 per project guidelines), and a0. ahmed() outputs the Fourier Series values at each of the 100 nodes, which I stored into 'fx' in my code. To be clear, ahmed() essentially programs Figure 2, and fourier_coeff() programs the computation of the coefficients needed by ahmed(). The resulting Fourier Series values can be plotted with the data points as shown below:



As we can see, the Fourier Series values do reflect what I expect the average temperatures should look like throughout the year. There is not too much oscillation between data points. The Fourier Series values also interpolate all data points except for the first and last.

Finally, the last plot in my code shows the Fourier Series, the natural cubic spline, and the data points on one single graph. It is shown below:



Clearly, both methods yield similar results. The spline interpolates all data points while the Fourier series interpolates all but the first and last data point.

In conclusion, this project explored the Natural Cubic Spline and Discrete Fourier Series for the analysis of the average temperature in a city X. Both methods seemed to be good approximations of the data, as they did not give issues of over-fitting. I would like to expand the scope of this project with this topic and look into extrapolating methods, because I think those methods have interesting applications in numerical forecasting.

Program Listings

Problem 1 part 1:

Driver:

```
h = 1 / 11;
n = 12;

A = generate_matrix(n - 2, h);

disp(A)
```

Function:

```
function A = generate_matrix(n, h)

if n < 0
    disp('Error: n must be greater than 0!');
else
    A = ( 2 * ( h + h ) )  * eye(n);

    for i = 1 : n - 1
        A(i + 1, i) = h;
    end
    for i = 1 : n - 1
        A(i, i + 1) = h;
    end

end
```

Problem 1 part 2:

Driver:

```
data = [43.5, 48, 57.5, 65.5, 73, 81, 85.5, 85, 78, 67.5, 56, 47.5]';

RHS = rhs(data, h , n - 2);

disp(RHS)
sigma = zeros(n, 1);

sigma(2 : n - 1) = A \ RHS;
```

Function:

```
function rhs = rhs(data, h, n)
```

```
if h < 0
    disp('Error: h cannot be less than 0!');
else

    rhs = zeros(n,1);

    for i = 1 : n
        rhs(i) = ( ( data( i + 2 ) - data( i + 1 ) ) / h ) - ( ( data
( i + 1 ) - data ( i ) ) / h );
    end
    rhs = 6 * rhs ;
end
```

Problem 1 part 3:

Driver:

```
n = 12;
knots = linspace(0, 1, n)';

n = 100;
nodes = linspace(0, 1, n)';

values = feval_spline(knots, nodes, data, sigma, h);

figure(1)
plot(nodes, values)
title('Natural Cubic Spline')
xlabel('Time')
ylabel('Temperature')
hold on
scatter(knots, data, 'r')
xticks([0 1/11 2/11 3/11 4/11 5/11 6/11 7/11 8/11 9/11 10/11 1])
xticklabels({'0', '1/11', '2/11', '3/11', '4/11', '5/11', '6/11',
'7/11', '8/11', '9/11', '10/11', '1'})
legend('Natural Cubic Spline', 'Data')
legend('Location','northwest')
legend('boxoff')
hold off
```

Function:

```
function g= feval_spline(knots, xnodes, fvals, coeff, xdelta)

%  input:
%         knots       column array of knots the spline is generated on
%         xnodes      column array of x values that the spline is to
be evaluated on
%         fvals       function values at interpolating nodes; size
(n+2)
```

```
%          coeff       column array of coefficients for generated
spline
%          xdelta      spatial meshsize
%  output:
%          g           spline values evaluated at the xnodes

m= length(xnodes);
n= length(knots);
g= zeros(m,1);

beg= 1;
for i= 1:m
   x= xnodes(i);

   % find the knot subinterval that x lands in
   for j= beg:n
      if ( x >= knots(j) && x <= knots(j+1) )
         found= j+1;
         beg  = j;
         break;
      end
   end
   g(i)= (knots(found)-x)^3/(6*xdelta)*coeff(found-1) + (x-
knots(found-1))^3/(6*xdelta)*coeff(found) + (fvals(found)/xdelta -
xdelta/6.*coeff(found)) ...
         *(x-knots(found-1)) + (fvals(found-1)/xdelta -
xdelta/6.*coeff(found-1))*(knots(found)-x);
end
```

__Problem 2:__

__Driver:__

```
N = 5;
a = zeros(N, 1);
for i = 1 : N
    a(i) = fourier_coeff(data, i, 1);
end
b = zeros(N, 1);
for i = 1 : N
    b(i) = fourier_coeff(data, i, 0);
end
a0 = fourier_coeff(data, 0, 1);

fx = ahmed(a, b, nodes, N, a0);

figure(2)
plot(nodes, fx)
title('Discrete Truncated Fourier Series')
xlabel('Time')
```

```
ylabel('Temperature')
hold on
scatter(knots, data, 'r')
xticks([0 1/11 2/11 3/11 4/11 5/11 6/11 7/11 8/11 9/11 10/11 1])
xticklabels({'0', '1/11', '2/11', '3/11', '4/11', '5/11', '6/11',
'7/11', '8/11', '9/11', '10/11', '1'})
legend('Fourier Series', 'Data')
legend('Location','northwest')
legend('boxoff')
hold off


figure(3)
plot(nodes, fx)
title('Fourier Series vs Natural Cubic Spline')
xlabel('Time')
ylabel('Temperature')
hold on
plot(nodes, values)
hold on
scatter(knots, data, 'g')
legend('Fourier Series','Natural Cubic Spline', 'Data')
legend('Location','northwest')
legend('boxoff')
xticks([0 1/11 2/11 3/11 4/11 5/11 6/11 7/11 8/11 9/11 10/11 1])
xticklabels({'0', '1/11', '2/11', '3/11', '4/11', '5/11', '6/11',
'7/11', '8/11', '9/11', '10/11', '1'})
hold off
```

Functions:

```
function coeff= fourier_coeff(fvals, n, type)

% input
%       fvals  column array of function values evaluated at equally-
spaced nodes in [0,1].
%       n      index of coefficient to be determined
%       type   flag to determine cosine or sine coefficient
%
% output
%       coeff  Fourier coefficient a_n or b_n

% trapezoid rule to approximate Fourier coefficients

nvals= length(fvals);
h    = 1.0/(nvals-1);
xnodes= 0:h:1;
xnodes= xnodes';

if type > 0
   g= cos(n*2*pi*xnodes);
else
```

```
   g= sin(n*2*pi*xnodes);
end

integrand= fvals.*g;

coeff= 2*h*( sum(integrand,1) - 0.5*integrand(1) -
0.5*integrand(nvals) );  % trapezoid

if n < 1
   coeff= 0.5*coeff;
end




function fx = ahmed(a, b, x, N, a0)

fx = zeros(100, 1);

for i = 1 : length(x)
    for j = 1 : N
        fx(i) =  a(j) * cos( 2 * j * pi * x(i) ) + b(j) * sin ( 2 * j
* pi * x(i) ) + fx(i) ;
    end
    fx(i) = fx(i) + a0;
end
```