CME 108/MATH 114
Introduction to Scientific Computing
Summer 2018

Problem Set 5 – Solution

Due: Monday, August 6 at 11:59pm

Warm-up Questions

1. *(Conjugate powers.)* Consider an $n \times n$ matrix A such that $A = PDP^{-1}$, for some matrices P, D. Show that

$$A^m = PD^mP^{-1}.$$

Solution

The base case $m = 1$ follows by hypothesis, so suppose the statement is true for some $m \geq 1$. Then

$$A^{m+1} = A(A^m) = A(PD^mP^{-1}) = PD(P^{-1}P)D^mP^{-1} = PD^{m+1}P^{-1},$$

completing the induction.

2. *(Inverse determinant.)* Using the multiplicativity of the determinant ($\det(AB) = \det(A)\det(B)$ for any $n \times n$ A, B), show that

$$\det(A^{-1}) = (\det(A))^{-1}$$

for any invertible A.

Solution

Since A is invertible, we have $AA^{-1} = I$. Therefore

$$\det(A)\det(A^{-1}) = \det(AA^{-1}) = \det(I).$$

On a Warm-up Question in HW3, we showed that the determinant of a diagonal matrix is the product of its diagonal entries. Hence, $\det(I) = 1$ and we have $\det(A)\det(A^{-1}) = 1$, which implies

$$\det(A^{-1}) = \frac{1}{\det(A)}$$

as desired.

3. *(Polynomial eigenvalues.)* Suppose λ is an eigenvalue of an $n \times n$ matrix A and that v is a λ-eigenvector of A.

   (a) Show that $A^2 v = \lambda^2 v$.

   (b) Show that λ^m is an eigenvalue of A^m.

   (c) Consider the polynomial $p(x) = c_0 + c_1 x + \cdots + c_m x^m$, and define

   $$p(A) = c_0 I_n + c_1 A + \cdots + c_m A^m.$$

   Show that $p(\lambda)$ is an eigenvalue of $p(A)$.

Solution

(a) The definition of v and the linearity of multiplication imply that

$$A^2 v = A(Av) = A(\lambda v) = \lambda(Av) = \lambda(\lambda v) = \lambda^2 v.$$

(b) The base case $m = 1$ follows by definition λ, so suppose $A^k v = \lambda^k v$ for some $m \geq 1$. Then

$$A^{k+1} v = A(A^k v) = A(\lambda^k v) = \lambda^k (Av) = \lambda^{k+1} v,$$

completing the induction. Since $A^k v = \lambda^k v$ for every positive integer k, v is a λ^k-eigenvector of A^k and λ^k is an eigenvalue of A^k.

(c) Note that

$$\begin{aligned} p(A)v &= (a_0 I_n + a_1 A + \cdots a_n A^n)v \\ &= a_0(Iv) + a_1(Av) + \cdots + a_n(A^n v) \\ &= a_0 v + a_1 \lambda v + \cdots + a_n \lambda^n v \\ &= (a_0 + a_1 \lambda + \cdots + a_n \lambda^n)v \\ &= p(\lambda)v. \end{aligned}$$

In the third equality we used the result from part (b) above. The equality $p(A)v = p(\lambda)v$ proves that $p(\lambda)$ is an eigenvalue of the matrix $p(A)$.

4. *(Inverse eigenvalues.)* Suppose that A is an invertible matrix and that v is a λ-eigenvector of A.

   First show that $\lambda \neq 0$. That is, show that an invertible matrix cannot have the eigenvalue 0.

   Now show that v is a $\frac{1}{\lambda}$-eigenvector of A^{-1}.

   Solution

   If 0 were an eigenvalue of A, then $Av = 0v = 0$ for some non-zero vector v. Multiplying the last equality by A^{-1} yields a contradiction, since $v \neq 0$:

   $$Av = 0 \implies A^{-1}Av = A^{-1}0 \implies v = 0.$$

   By definition, $Av = \lambda v$. Multiplying by A^{-1} on both sides yields

   $$A^{-1}(Av) = \lambda A^{-1}v.$$

   Since $A^{-1}A = I$, $A^{-1}(Av) = (A^{-1}A)v = v$, implying that

   $$v = \lambda A^{-1}v,$$

   or equivalently $A^{-1}v = \lambda^{-1}v$.

5. *(Uniqueness of interpolating polynomial, Bradie 5.1.7.)* Consider the following data set:

x	-1	0	1	2
y	5	1	1	11

   (a) Show that the polynomials $f(x) = x^3 + 2x^2 - 3x + 1$ and $g(x) = \frac{1}{8}x^4 + \frac{3}{4}x^3 + \frac{15}{8}x^2 - \frac{11}{4}x + 1$ both interpolate all of the data.

   (b) Why does this not contradict the uniqueness statement of the theorem on existence and uniqueness of the interpolating polynomial?

   Solution

   (a) For this part, we explicitly compute $f(k), g(k)$ for $k = -1, \ldots, 2$, and verify that the result is the corresponding table entry. For instance,

   $$f(1) = 1^3 + 2(1)^2 - 3(1) + 1 = 1, \text{ and}$$
   $$g(1) = \frac{1}{8}(1)^4 + \frac{3}{4}(1)^3 + \frac{15}{8}(1)^2 - \frac{11}{4}(1) + 1 = 1.$$

(b) Note that $\deg(f) = 3$ and $\deg(g) = 4$. We are given $3 + 1$ points in the data table, and the theorem guarantees the uniqueness of an interpolating polynomial of *degree at most 3*. It makes no mention of higher degree polynomials.

6. *(Piecewise linear interpolation, Bradie 5.5.1.)* Consider the following data for the density ρ of water as a function of temperature T:

T (°C)	ρ (kg/m^3)
0	1000
10	1000
20	998
30	996
40	992
50	988
60	983
70	978
80	972
90	965
100	958

Estimate, using piecewise linear interpolation, the density of water when the temperature is 34 °C, 68 °C, and 91 °C.

Solution

Estimating the density when $T = 34$ requires construction of the interpolating linear polynomial on $[30, 40]$. By definition, we have

$$\begin{aligned} s_3(x) &= a_3 + b_3(x - x_3) \\ &= \rho_3 + \frac{\rho_4 - \rho_3}{x_4 - x_3}(x - x_3) \\ &= 996 + \frac{992 - 996}{40 - 30}(x - 30) \\ &= 1,008 - 0.4x. \end{aligned}$$

We estimate the density at $T = 34$ as $s_3(34) = 994.4$.

Similarly, estimating the density when $T = 68$ requires construction of the interpolating linear polynomial on $[60, 70]$. By definition, we have

$$\begin{aligned} s_6(x) &= a_6 + b_6(x - x_6) \\ &= \rho_6 + \frac{\rho_7 - \rho_6}{x_7 - x_6}(x - x_6) \\ &= 983 + \frac{978 - 983}{70 - 60}(x - 60) \\ &= 1,013 - 0.5x. \end{aligned}$$

We estimate the density at $T = 68$ as $s_6(68) = 979$.

Finally, to estimate the density when $T = 91$, we need the linear interpolant on $[90, 100]$. Following the same procedure as above, we obtain

$$s_9(x) = 1,028 - 0.7x,$$

so that our estimate becomes $s_9(91) = 964.3$.

Assignment Problems

1. (20 points) (*Computing eigenvalues.*) Consider the matrix

$$A = \begin{bmatrix} 0 & -17 & 21 \\ 0 & 13 & -15 \\ 0 & 10 & -12 \end{bmatrix}.$$

Recall that $\lambda \in \mathbb{R}$ is an *eigenvalue* of A if the equality $Av = \lambda v$ is satisfied by some non-zero vector v.

(a) It turns out that the eigenvalues of any $n \times n$ matrix M are exactly the roots of the polynomial

$$\chi(\lambda) = \det(\lambda I - M).$$

This polynomial is known as the *characteristic polynomial* of M.

Compute the eigenvalues of the given matrix A.

(b) In MATLAB, implement the following algorithm:

INPUT : $n \times n$ matrix A, n-vector q_0, tolerance ϵ, maximum number of iterations N

$q \leftarrow \frac{q_0}{\|q_0\|}$
$\lambda_{\text{curr}} \leftarrow 0$
for $k = 1, \ldots, N$ **do**
  $\lambda_{\text{prev}} \leftarrow \lambda_{\text{curr}}$
  $v \leftarrow Aq$
  $\lambda_{\text{curr}} \leftarrow q^T v$
  $q \leftarrow \frac{v}{\|v\|}$
  if $|\lambda_{prev} - \lambda_{curr}| < \epsilon$ **then**
    break
  end
end
OUTPUT: λ_{curr}

Use the MATLAB command `rand` to generate a vector q_0 with random entries and run your algorithm using as input: the given A, the generated q_0, $\epsilon = 10^{-6}$, and $N = 100$.

With your results from part (a) in mind, what can you say about the output of your algorithm?

(c) Lastly, implement the following algorithm in MATLAB:

INPUT : matrix A, maximum number of iterations N

for $k = 1, \ldots, N$ **do**
  compute LU factorization $A = LU$
  $A \leftarrow UL$
end
OUTPUT: A

Use the built-in MATLAB function `lu`.

Your earlier partial pivoting LU code **cannot** factorize the given matrix, since its first column is zero. Full pivoting, which incorporates permuting both rows and columns, is required for this factorization, and the `lu` function implements this strategy. *Full pivoting* ensures that at step k, $A(k,k)$ is the largest entry in absolute value of the entire submatrix $A(k:n, k:n)$. In contrast, partial pivoting only guarantees that $A(k,k)$ is the largest entry of the vector $A(k:n, k)$.

Run your algorithm using the given matrix A as input, and set $N = 100$.

With your results from (a) in mind, what can you say about the output of your algorithm?

Hint: Pay particular attention to the output diagonal!

Solution

(a) Note that

$$\lambda I - A = \begin{bmatrix} \lambda & 17 & -21 \\ 0 & \lambda - 13 & 15 \\ 0 & -10 & \lambda + 12 \end{bmatrix}.$$

Expanding about the first column, we compute the determinant as

$$\begin{aligned} \det(\lambda I - A) &= \lambda \det\left(\begin{bmatrix} \lambda - 13 & 15 \\ -10 & \lambda + 12 \end{bmatrix} \right) \\ &= \lambda((\lambda - 13)(\lambda + 12) + 150) \\ &= \lambda(\lambda - 3)(\lambda + 2). \end{aligned}$$

Clearly, the polynomial $\chi(\lambda) = \det(\lambda I - A)$ has roots at $\lambda = -2, 0, 3$. Hence, the eigenvalues of A are $-2, 0, 3$.

(b) The algorithm is implemented below.

```
%Power iteration

%Input matrix and define input parameters
A = [0 -17 21; 0 13 -15; 0 10 -12];
q0 = rand(length(A),1);
eps = 1e-6;
N = 100;

%Iterate
q = q0 / norm(q0);
lambda_curr = 0;
for k=1:N
    lambda_prev = lambda_curr;
    aq = A*q;
    lambda_curr = q'*aq;
    q = aq / norm(aq);
    if abs(lambda_curr-lambda_prev) < eps
```

```
            break
        end
end
lambda_curr
```

We note that the algorithm returns $\lambda_{\text{curr}} = 3.000$. This is exactly the largest (in absolute value) eigenvalue of A.

The Power Iteration algorithm is used to compute the largest eigenvalue of a matrix and a corresponding unit eigenvector. A unit eigenvector corresponding to λ_{curr} is stored in q.

(c) The implementation details are provided below.

```
%Input matrix and define parameters
A = [0 -17 21; 0 13 -15; 0 10 -12];
N = 100;

%LR iteration
for k=1:N
    [L, U] = lu(A);
    A = U * L;
end
A
```

The algorithm outputs

$$A = \begin{bmatrix} 0.0000 & -3.0000 & 21.0000 \\ & 3.0000 & -15.0000 \\ & & -2.0000 \end{bmatrix}.$$

Note that the output diagonal contains the eigenvalues of A as entries.

This algorithm is called "LR iteration." It is the predecessor of the "QR iteration," which is labeled one of the top 10 algorithms of the 20th century. The interested reader may find more at: IEEE Top 10 Algorithms of the 20th century and at: IEEE description of QR iteration.

2. (10 points) *(Eigen-theory.)* **BONUS PROBLEM!** Let A be an $n \times n$ matrix with real entries.

(a) Assuming the statement of the previous prompt, show that A can have at most n real eigenvalues. In addition, show that if μ is a complex

eigenvalue of A, then $\bar{\mu}$ is also an eigenvalue of A (the over-bar denotes complex conjugation).

(b) Suppose that P is any invertible $n \times n$ matrix. Show that A and $P^{-1}AP$ have the same eigenvalues.

(c) If D is a diagonal matrix, what are the eigenvalues of D?

(d) Use the characteristic polynomial to show that

$$\det(A) = \prod_{i=1}^{n} \lambda_i,$$

where λ_i denote the eigenvalues of A.

Solution

(a) The eigenvalues of A are roots of the characteristic polynomial $\chi(\lambda) = \det(\lambda I - A)$. This is a polynomial of degree n (this can be easily shown using induction), so A has at most n roots by the Fundamental Theorem of Algebra (FTA).

Since A has real entries, the coefficients of χ are also real. Therefore, every complex root of χ must appear in pairs of complex conjugates. Hence if μ is an eigenvalue of A then so is $\bar{\mu}$.

(b) Suppose λ is an eigenvalue of A, with corresponding eigenvector v. Set $w = P^{-1}v$ and note that

$$(P^{-1}AP)w = P^{-1}A(PP^{-1}v) = P^{-1}(Av) = P^{-1}(\lambda v) = \lambda w,$$

so that λ is an eigenvalue of $P^{-1}AP$.

Conversely, consider an eigenvalue λ of $P^{-1}AP$, with corresponding eigenvector v. Now set $w = Pv$. By definition,

$$P^{-1}APv = \lambda v.$$

Multiplying both sides by P implies

$$APv = \lambda Pv,$$

or equivalently, $Aw = \lambda w$, showing that λ is an eigenvalue of A.

(c) The eigenvalues of D are precisely the diagonal entries. Using the result of a Warm-Up exercise on HW4, we have that

$$\chi(\lambda) = \det(D - \lambda I) = \prod_{i=1}^{n} (d_{ii} - \lambda).$$

(d) Note that $A = PDP^{-1}$ implies $P^{-1}DP = A$, so that D and $A = P^{-1}DP$ have the same eigenvalues by part (b). Since the eigenvalues of D are its diagonal entries, the eigenvalues λ_i of A are precisely the diagonal entries of D. Using the result of Warm-Up exercise 3, we have that

$$\begin{aligned}
\det(A) &= \det(PDP^{-1}) \\
&= \det(P)\det(D)\det(P^{-1}) \\
&= \det(P)\det(D)\det(P)^{-1} \\
&= \det(D) \\
&= \prod_{i=1}^{n} d_{ii} \\
&= \prod_{i=1}^{n} \lambda_i,
\end{aligned}$$

as desired.

3. (15 points) *(Interpolation points, Bradie 5.1.10.)* The interpolation points influence the interpolation error through the polynomial $\prod_{i=0}^{n}(x - x_i)$. Suppose we are interpolating the function f over the interval $[-1, 1]$ using linear interpolation.

   (a) If $x_0 = -1$ and $x_1 = 1$, determine the maximum value of the expression $|(x - x_0)(x - x_1)|$ for $-1 \leq x \leq 1$.

   (b) If $x_0 = -\sqrt{2}/2$ and $x_1 = \sqrt{2}/2$, determine the maximum value of the expression $|(x - x_0)(x - x_1)|$ for $-1 \leq x \leq 1$. How does this maximum compare to the maximum found in part (a)?

   (c) Select any two numbers from the interval $[-1, 1]$ to serve as the interpolation points x_0 and x_1. Determine the maximum value of the expression $|(x - x_0)(x - x_1)|$ for $-1 \leq x \leq 1$, and compare to the maxima found in (a) and (b).

Solution

Define $g(x) = (x - x_0)(x - x_2)$ and note that the maximum value of the expression $|(x - x_0)(x - x_1)|$ for $-1 \leq x \leq 1$ corresponds to an extreme value of g. Since g is smooth, the Extreme Value Theorem (EVT) implies that the extrema of g can only occur at the critical points (when $g'(x) = 0$) or at the boundary of the interval. Setting $g'(x) = 0$ yields

$$2x - (x_0 + x_1) = 0,$$

or equivalently $x = \frac{x_0+x_1}{2}$. Therefore, the maximum value of the expression $|(x - x_0)(x - x_1)|$ for $-1 \leq x \leq 1$ is achieved either at $x = -1, 1$, or $x = (x_0 + x_1)/2$.

(a) In this case, $(x_0 + x_1)/2 = 0$ and $g(-1) = g(1) = 0$ while $g(0) = -1$. Therefore, in this case the maximum of the expression is 1.

(b) In this case, $(x_0 + x_1)/2 = 0$ and $g(-1) = g(1) = 1/2$ while $g(0) = -1/2$. Therefore, in this case the maximum of the expression is $1/2$. The maximum here is half of that found in part (a).

(c) Let $x^* = (x_0 + x_1)/2$ and note that

$$g(x^*) = -\frac{(x_1 - x_0)^2}{4}.$$

This expression is maximized when x_0 and x_1 are as far from each other as possible, which happens when $x_0 = -1$ and $x_1 = 1$. This is the situation in part (a), and the corresponding maximum is in fact 1.

However, we can do better (or worse)! Note that

$$g(1) = (1 - x_0)(1 - x_1) = x_0x_1 - (x_0 + x_1) + 1.$$

In the limit where $x_0 = x_1 = -1$, we have $g(1) = 4$. Similarly, in the limit where $x_0 = x_1 = 1$, $g(-1) = 4$. These cases result in the largest possible maximum value of the expression $|(x - x_0)(x - x_1)|$ for $-1 \leq x \leq 1$, over *any* choice of interpolating points in $[-1, 1]$.

4. (25 points) *(Cubic spline, Bradie 5.6.10.)* In MATLAB, implement the construction of the cubic spline interpolant, as described in lecture. In particular, follow these steps:

- Set up a tridiagonal linear system for c_j and solve.
- Compute the other coefficients b_j and d_j.
- Combine your coefficients to form $s_j(x)$.

Now consider the following data set:

x	0.0	0.5	1.0	1.5	2.0
y	0.500000	1.425639	2.640859	4.009155	5.305472
y'	1.500000				2.305472

(a) Construct the cubic spline interpolant for this data set using the not-a-knot boundary conditions.

(b) Construct the cubic spline interpolant for this data set using the clamped spline boundary conditions.

(c) The data for this problem is taken from the function $y = (1+x)^2 - 0.5e^x$. Plot the error $\epsilon(x) = y(x) - s(x)$ in each of the splines from (a) and (b) . Which spline produced better results?

Note: You may (should) use the built-in `spline` function to verify your computations.

Solution

The error associated to the clamped spline is smaller than that associated to the not-a-knot boundary conditions. Note the vertical scale in the plots. The clamped spline exploits additional information about the function (derivative values at the boundary of the interval) and therefore provides a closer match to the actual function values.

Implementation details and plots are provided below.

```
%Input data
h = 0.5;
x = 0:h:2;
y = [0.5 1.425639 2.640859 4.009155 5.305472]';
yprime_a = 1.5;
yprime_b = 2.305472;
n = length(x)-1;

%Define function handle
```

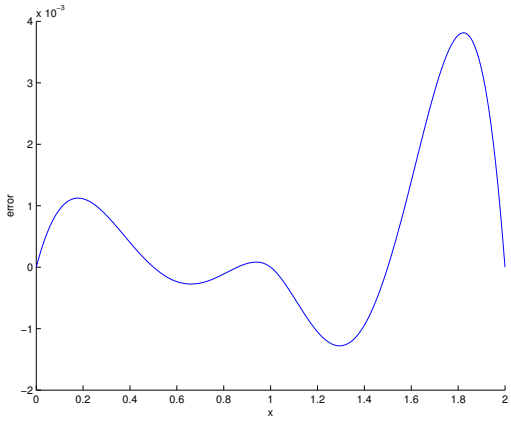

Figure 1: Error for Not-a-Knot boundary conditions

```
fext = @(x)(x+1).^2-0.5*exp(x);

%Initialize a_j
a = y;

%Assemble tridiagonal matrix
Mat = h*(4*eye(n+1)+diag(ones(n,1),1)+diag(ones(n,1),-1)
    );
rhs = zeros(n+1,1);
rhs(2:n) = 3/h*(a(3:n+1)-a(2:n))-3/h*(a(2:n)-a(1:n-1));

%Implement boundary conditions
%% Not-a-Knot
if(1)
      Mat(1,1) = h;
      Mat(1,2) = -2*h;
      Mat(1,3) = h;
      Mat(n+1,n-1) = h;
      Mat(n+1,n) = -2*h;
      Mat(n+1,n+1) = h;
      rhs(1) = 0;
      rhs(n+1) = 0;
end
```

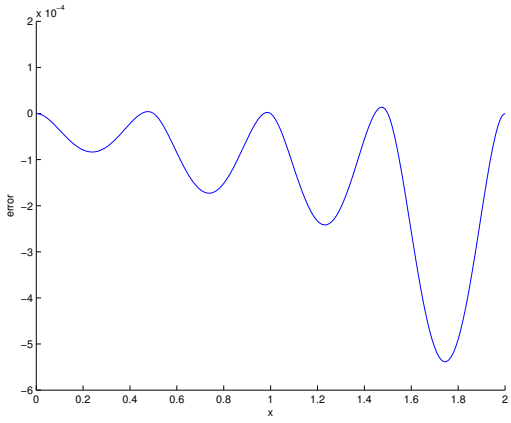

Figure 2: Error for Clamped boundary conditions

```
32
33 %% Clamped spline
34 if(0)
35     Mat(1,1) = 2*h;
36     Mat(1,2) = h;
37     Mat(n+1,n) = h;
38     Mat(n+1,n+1) = 2*h;
39     rhs(1) = 3/h*(a(2)-a(1))-3*yprime_a;
40     rhs(n+1) = 3*yprime_b-3/h*(a(n+1)-a(n));
41 end
42
43 %%
44 %Solve for c_j
45 c = Mat\rhs;
46
47 %Compute b_j
48 b = (a(2:n+1)-a(1:n))/h-(2*c(1:n)+c(2:n+1))*h/3;
49
50 %Compute d_j
51 d = (c(2:n+1)-c(1:n))/3/h;
52
53 %% Plot the error
54 figure
```

```
55 hold on
56 for i = 1:n
57     xsub = x(i):0.01:x(i+1);
58     yext = fext(xsub);
59     yapp = a(i)+b(i)*(xsub-x(i))+c(i)*(xsub-x(i)).^2+d(i
           )*(xsub-x(i)).^3;
60     plot(xsub,yext-yapp)
61 end
62 xlabel('x')
63 ylabel('error')
```

5. (20 points) *(Lake pollution, from "Fundamentals of Engineering Numerical Analysis".)* The concentration of a certain toxin in a system of lakes downwind of an industrial area has been monitored very accurately at intervals from 1978 to 1992 as shown in the table below. It is believed that the concentration has varied smoothly between these data points.

Year	Concentration (ppm)
1978	12.0
1980	12.7
1982	13.0
1984	15.2
1986	18.2
1988	19.8
1990	24.1
1992	28.1

   (a) Interpolate the data using the Lagrange polynomial. Plot the polynomial along with the data points. Use your polynomial to predict the condition of the lakes in 1994.

   (b) Use a cubic spline interpolant with not-a-knot boundary conditions to predict the toxin concentration in 1994. You may use your code from Problem 4 or the MATLAB function `spline`.

   (c) Comment on the difference between your predictions.

Solution

(a) The value predicted for the year 1994 by the Lagrange polynomial is -38.4.

(b) The value predicted for the year 1994 by the cubic spline is 26.44.

(c) As shown in Figure 3, the plot of the Lagrangian polynomial drops sharply after the last point in a unrealistic fashion. Therefore, the value predicted by this polynomial is far from the true value and occurs as a result of over-fitting the data.

In contrast, prediction from the cubic spline seems realistic. In Figure 3, we can see that the cubic spline follows the general trend of the data.

Below is the Matlab code.

```
%Input data and plot points
xi = [1978; 1980; 1982; 1984; 1986; 1988; 1990; 1992];
fi = [12.0; 12.7; 13.0; 15.2; 18.2; 19.8; 24.1; 28.1];
figure
plot(xi,fi,'b*')
hold on;

%Construct the interpolating polynomial
%Define degree of interpolating polynomial, accounting
    for Matlab indexing
n=8;
N = 801;
x = linspace(1978,1994,N);
y = zeros(1,N);
for j=1:N
    %Construct Lagrange basis polynomial L_n,i
    for i=1:n
        L = 1;
        for k=1:n
            if k~=i
                L = L*(x(j)-xi(k))/(xi(i)-xi(k));
            end
        end
        %Evaluate interpolating polynomial
        y(j) = y(j)+fi(i)*L;
    end
end
```

```
27
28 %Plot results!
29 plot(x,y,'k')
30 xlabel('x'); ylabel('f(x)')
31
32 %Compute spline interpolant and plot results
33 yspline = spline(xi,fi,x);
34 plot(x,yspline,'r')
35 legend('Data Points','Lagrange','Cubic Spline','Location
       ','NorthWest')
36 print('lake_pollution.png','-dpng')
```

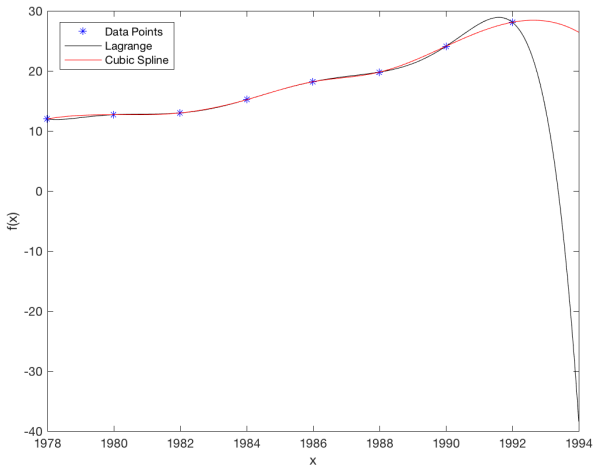

Figure 3: Extrapolation using Lagrange interpolation and Cubic spline

6. (25 points) *(Quadratic spline.)* In this problem you will explore a construction of a quadratic spline. The development is akin to that of the cubic spline, except that the interpolating polynomial is piecewice quadratic (parabolic) instead of cubic.

   Given $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ and $y_j = f(x_j)$, the quadratic spline $s(x)$ is defined by the conditions:

   - (Parabolicity) $s(x) = s_j(x)$ for $x \in [x_j, x_{j+1}]$,
   - (Interpolation) $s(x_j) = y_j$ for $j = 0, \ldots, n$,

- (Continuity) $s_j(x_{j+1}) = s_{j+1}(x_{j+1})$ for $j = 0, \ldots, n-2$,
- (Smoothness) $s'_j(x_{j+1}) = s'_{j+1}(x_{j+1})$ for $j = 0, \ldots, n-2$,

with

$$s_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2,$$

for $j = 0, \ldots, n-1$.

(a) Manipulate the defining relations to obtain a *recursive relation* for b_j, $j = 1, \ldots, n-1$.

Note: In lecture we showed how to manipulate the relations defining the cubic spline to obtain a tridiagonal system for the c_j's. The manipulation here is similar in nature, but much simpler.

(b) Note that the defining relations provide only $3n-1$ conditions on $3n$ variables. A similar situation arises in the case of the cubic spline, and we need to impose additional conditions (e.g., not-a-knot, clamped spline) in order to uniquely specify the interpolant.

How many additional conditions are needed to uniquely specify the quadratic spline? Propose a set of additional conditions.

Note: There are many valid sets of additional conditions.

Solution

(a) Starting with

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2$$
$$\text{and} \quad S_{j+1} = a_{j+1} + b_{j+1}(x - x_{j+1}) + c_{j+1}(x - x_{j+1})^2$$

and evaluating at $x = x_{j+1}$, we get

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 \qquad (S \text{ continuity}) \tag{1}$$

Similarly for S' continuity:

$$b_{j+1} = b_j + 2c_j h_j \quad \longrightarrow \quad c_j = \frac{b_{j+1} - b_j}{2h_j} \tag{2}$$

Substituting Eq. (2) into (1) yields

$$a_{j+1} = a_j + \frac{1}{2}(b_{j+1} + b_j)h_j$$

or

$$b_{j+1} = -b_j + \frac{2}{h_j}(y_{j+1} - y_j) \quad j = 0, 1, 2, \ldots n-1 \tag{3}$$

Equation (3) is the recursive formula for calculation b_j which is S'_j

(b) To use the recursive formula above, we need **one** boundary condition; either at the left-most point, $S'_0(x_0)$, or at the right-most point, $S'_{n-1}(x_n)$. It's easiest to set the left boundary condition and use the recursive formula to march to the right. One choice of left boundary condition is:

$$S'_0(x_0) = 0$$

The recursive relation in Eq. (3) is shown below:

$$\begin{aligned}
b_0 = \quad & S'_0(x_0) = 0 \qquad \text{(left boundary condition)} \\
b_1 = \quad & S'_1(x_1) = \frac{2}{h_0}(y_1 - y_0) \\
b_2 = \quad & S'_2(x_2) = \frac{2}{h_1}(y_2 - y_1) - b_1 \\
& \ldots \\
b_{n-1} = S'_{n-1}(x_{n-1}) = & \frac{2}{h_{n-2}}(y_{n-1} - y_{n-2}) - b_{n-2} \\
b_n = \quad S'_{n-1}(x_n) = & \frac{2}{h_{n-1}}(y_n - y_{n-1}) - b_{n-1}.
\end{aligned}$$