Numerical Analysis (089180) A.Y. 2022-2023
prof. Simona Perotto Luca Liverotti
simona.perotto@polimi.it luca.liverotti@polimi.it

Lab 8 – Solutions

December 2, 2022

Given a function $f(x)$, an **interpolating function** is generally defined as

$$g(x) = \sum_{k=0}^{n} \alpha_k \phi_k(x) \qquad \text{s.t.} \qquad g(x_i) = f(x_i) \quad i = 0, 1, \ldots, n$$

where $\{x_i\}_{i=0}^n$ and $\{\phi_i(x)\}_{i=0}^n$ are respectively sets of $n+1$ **nodes** and **basis functions**, and the coefficients α_k are called **weights**.

Definition 8.1 (Lagrange interpolating polynomial). *Characteristic polynomials are defined as*

$$\varphi_k(x) = \frac{\prod\limits_{i \neq k} (x - x_i)}{\prod\limits_{i \neq k} (x_k - x_i)}$$

and they are s.t. $\varphi_k(x_i) = \delta_{ik}$. Using them as basis functions, it holds $\alpha_i = f(x_i)$ so that we obtain the Lagrange

$$\Pi_n f(x) = \sum_{k=0}^{n} f(x_k) \varphi_k(x).$$

Theorem 8.1 (Interpolation error). *Let $\Pi_n f(x)$ be the interpolating polynomial of order n at the nodes $x_i \in [a, b]$. If $f(x) \in \mathcal{C}^{n+1}([a, b])$ then*

$$E_n f(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \qquad \xi \in [a, b],\ \forall x \in [a, b].$$

If the nodes are equally spaced with step h, it holds

$$\|E_n f(x)\|_{L^\infty([a,b])} \leq \frac{\|f^{(n+1)}(x)\|_{L^\infty([a,b])}}{4(n+1)} h^{n+1}$$

This does not ensure a priori the uniform convergence of $\Pi_n f(x)$ to $f(x)$ for $n \to \infty$.

Definition 8.2 (Piecewise linear interpolation). *Given of nodes: $x_0 < x_1 < \cdots < x_n$, we denote by I_i the interval $[x_i, x_{i+1}]$ and by H the maximum length of these intervals. We denote by $\Pi_1^H f$ the piecewise linear interpolating polynomial of f given by:*

$$\Pi_1^H f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} (x - x_i) \quad \forall x \in I_i.$$

Theorem 8.2 (Interpolation error). *If $f \in C^2(I)$, where $I = [x_0, x_n]$, then*

$$\|f(x) - \Pi_1^H f(x)\|_{L^\infty(I)} \leqslant \frac{H^2}{8} \|f''(x)\|_{L^\infty(I)}$$

This bound allows us to ensure the uniform convergence of the interpolating polynomial to f.

MATLAB commands

Lagrange interpolation:
```
>> help polyfit
>> help polyval
```
Piecewise polynomial interpolation:
```
>> help interp1
```
Cubic spline:
```
>> help spline
```
Notice that the commands polyfit and polyval can be used to compute both the interpolation and a least-square approximation
```
polyfit(x_nodes, y_nodes, N)
```

$$\begin{array}{rcll} N+1 & = & \#x\_nodes \implies & \text{Lagrange interpolation;} \\ N+1 & != & \#x\_nodes \implies & \text{least square interpolation.} \end{array}$$

Exercise 8.1. Approximate the function $f(x) = x\sin(x)$ in the interval $[-2, 6]$.

a. Consider a set of equally spaced nodes: find the interpolating polynomial using Lagrange basis polynomials (`polyfit`, `polyval` commands) of degree $4, 6, 8$. Plot the function $f(x)$ and the associate interpolating polynomials.

b. Estimate the interpolation error when resorting to a quadratic interpolation (n=2), using the theoretical results. Then evaluate the interpolating polynomial in 1000 equally spaced points and plot the interpolation error. Motivate the result.

c. Compute with Matlab the piecewise linear interpolant (`interp1` command) and the cubic spline (`spline` command) associated with $n + 1 = 6, 11, 21$ equally spaced nodes.

Solution Exercise 8.1.

ex_8_1.m

```
clc
clear all
close all


f = @(x) x.*sin(x);
a = -2;
b = 6;

% a) Lagrange interpolation
% Set the order of lagrange interpolation
n = 4; %n = 6; %n = 8;

% Set the equally spaced nodes position in the interval a b
x_nodes = linspace(a, b, n+1);
% Evaluation of the funtion in the equally spaced nodes
y_nodes = f(x_nodes);
% Lagrange interpolation using polyfit function
interp_coeff = polyfit(x_nodes, y_nodes, n);
% Plot the funtion and its polynomial interpolation
x_plot = linspace(a, b, 1000);
f_plot = f(x_plot);
interp_plot = polyval(interp_coeff, x_plot);
figure;
plot(x_plot, f_plot, '--',x_plot, interp_plot, x_nodes, y_nodes, 'o','LineWidth',2,'MarkerSize
',12);
xlim([a-0.1, b+0.1]);

% b) Estimation of the interpolation error using what reported in slides
% (Remember the Infinite norm
% corresponds to take the maximum of the ads value)

% calculation of the maximum in the derivative term
```

```
if (n == 4)
  % the n+1-th derivative is 5 sin(x) + x cos(x). An upper bound of its absolute value is
  deriv_upper_bound = 11 ;
elseif (n == 6)
  % the n+1-th derivative is -7 sin(x) - x cos(x). An upper bound of its absolute value is
  deriv_upper_bound = 13 ;
elseif (n == 8)
  % the n+1-th derivative is 9 sin(x) + x cos(x). An upper bound of its absolute value is
  deriv_upper_bound = 15 ;
else
  error('Wrong n');
end
% evaluation of the upper bound of the error (Remember the Infinite norm
% corresponds to take the maximum of the ads value)
err_est = (x_nodes(2)-x_nodes(1))^(n+1)/(4*(n+1)) * deriv_upper_bound;

% Calculation of the error
% err_norm = norm((f_plot - interp_plot),'inf');
err = abs(f_plot - interp_plot);
% Plotting the
figure;
plot(x_plot, err, '-b','LineWidth',2,'MarkerSize',12);
hold on;
plot(x_plot, err_est*ones(size(x_plot)), '-r','LineWidth',2,'MarkerSize',12)

% The interpolation on equally spaced nodes produces a non uniform error
% in the interval, which grows towards the extreme points.

% c-1) Computation of the piecewise linear interpolant using interp1 Matlab command
% on equally spaced nodes

% Setting the posion of the nodes
n = 5; % n=10, n=20
xint = linspace(a, b, n+1);
% Evaluation of the function in the interpolating nodes
yint = f(xint);
% Calculation of the piecewise linear interpolant and evaluation of
% the interpolant in the x_plot node to visualize the function in the plot
y_plot = interp1(xint, yint, x_plot);
% Plotting
figure;
plot(x_plot, f(x_plot), 'k-', x_plot, y_plot, 'r-', xint, yint, 'rx', 'LineWidth', 2, '
MarkerSize', 8)
title('Piecewise Linear interpolation') ;

% c-2) Computation of the spine line interpolant using spline Matlab command
% on equally spaced nodes

xsp = linspace(a, b, n+1);
ysp = f(xsp);
y_plot = spline(xsp, ysp, x_plot);

figure;
plot(x_plot, f(x_plot), 'k-', x_plot, y_plot, 'r-', xsp, ysp, 'rx', 'LineWidth', 2, '
MarkerSize', 8)
title ('Spline interpolation')
```

Exercise 8.2 (Runge's counterexample). Consider the function

$$f(x) = \frac{1}{1+x^2} \qquad -5 \leq x \leq 5.$$

a. Verify with Matlab that the interpolating polynomials $\Pi_n f(x)$ using equally spaced nodes are such that

$$\lim_{n\to\infty} |f(x) - \Pi_n f(x)| \neq 0.$$

Check this statement graphycally and by computing the infinity norm

b. Compute the piecewise linear interpolation (`interp1` command) with $n = 1, 2, 4, 8, 16, 32$ subintervals and plot the result. Find the error with the respect to the infinity norm and verify that it converges quadratically as a function of the distance between two nodes.

c. Interpolate the Runge function using a piecewise cubic spline (`spline` command) with $n = 1, 2, 4, 8, 16, 32$ subintervals. Plot the result.

d. Compute $\Pi_n f(x)$ using the Chebyshev nodes:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\hat{x}_i, \quad \hat{x}_i = -cos(\pi i/n), i = 0, \cdots, n$$

Solution Exercise 8.2.

ex_8_2.m

```
clc
clear all
close all

% Set the Runge function
f = @(x) 1./(1 + x.^2);
% Set the point to build the plot of the Rung function
xx = linspace(-5, 5, 1000);
fxx = f(xx);

% Plot the Runge function
figure
subplot(1,2,1)
hold on, box on
plot(xx, fxx, 'k-', 'LineWidth',2)
axis([-5.1 5.1 -0.4 1.2])
set(gca,'FontSize',16)
set(gca,'LineWidth',1.5)
xlabel('x','FontSize',16)
ylabel('f(x)','FontSize',16)

% Plot the Linear interpolant polynomial
for k = [2:2:10]
  x = linspace(-5, 5, k);
  fx = f(x);
  coef = polyfit(x, fx, k-1);
  yy = polyval(coef, xx);
  plot(xx, yy, 'r-', 'LineWidth',2)
  norm((fxx-yy),'inf')
  pause
end

% Plot the Runge function
subplot(1,2,2)
hold on, box on
plot(xx, fxx, 'k-', 'LineWidth',2)
axis([-5.1 5.1 -1.2 2.4])
set(gca,'FontSize',16)
set(gca,'LineWidth',1.5)
xlabel('x','FontSize',16)
ylabel('f(x)','FontSize',16)

% Plot the Linear interpolant polynomial
for k = [12:2:16]
  x = linspace(-5, 5, k);
  fx = f(x);
  coef = polyfit(x, fx, k-1);
  yy = polyval(coef, xx);
  plot(xx, yy, 'r-', 'LineWidth',2)
  norm((fxx-yy),'inf')
  pause
end
```

```
% The interpolation polynomials approach the function in the middle
% of the interval, but close to the boundaries increasing oscillations appear.

% b) Compute the piecewise linear interpolation (interp1 command) with
% n = 1, 2, 4, 8, 16, 32 elements and plot the result. Find the error

clear all
a = -5;
b = 5;
f = @(x) 1./(1+x.^2);
x_plot = linspace(a, b, 101);

n_vect = [1 2 4 8 16 32 64 128];

figure
for (i = 1:numel(n_vect))
  n = n_vect(i)

  x = linspace(a, b, n+1);
  y = f(x);

  y_plot = interp1(x, y, x_plot);
  plot(x_plot, f(x_plot), 'k-', x_plot, y_plot, 'r-', x, y, 'rx', 'LineWidth', 2, 'MarkerSize'
  , 8)
  axis([a-0.2 b+0.2 -0.1+min(f(x_plot)) 0.1+max(f(x_plot))])
  set(gca,'FontSize', 16)
  set(gca,'LineWidth', 1.5)

  error(i) = max(abs(f(x_plot) - y_plot));
  pause
end

H = 10./n_vect
pause

figure
loglog(H, error, 'bx-', 'LineWidth', 2, 'MarkerSize', 8)
hold on, box on
loglog(H, H.^(2), 'g-', 'LineWidth', 2)
axis([1e-1 1e1 1e-3 1e2])
set(gca,'FontSize',16)
set(gca,'LineWidth',1.5)
xlabel('h','FontSize',16)
ylabel('error','FontSize',16)
legend('error', 'O(h^2)', 'Location', 'ne');

% From the graph we can conclude that the convergence order is 2,
% as predicted by the theory.
% Calculation of the error
order = (log(error(1:end-1) ./ error(2:end))/log(2))'
% or using the diff command build in in matlab
p = -diff(log(error)) / log(2)

% c) Interpolation of the Runge function using a piecewise cubic spline on
% the same number of iterpolating points

figure
for (i = 1:numel(n_vect))
  n = n_vect(i)

  x = linspace(a, b, n+1);
  y = f(x);

  y_plot = spline(x, y, x_plot);
  plot(x_plot, f(x_plot), 'k-', x_plot, y_plot, 'r-', x, y, 'rx', 'LineWidth', 2, 'MarkerSize'
  , 8)
  axis([a-0.2 b+0.2 -0.1+min(f(x_plot)) 0.1+max(f(x_plot))])
  set(gca,'FontSize', 16)
  set(gca,'LineWidth', 1.5)
```

```
  pause
end

% d) Computation of the linera interpolant by using the Chebyshev nodes

xx = linspace(-5, 5, 1000);

figure;
for (i = 1:numel(n_vect)-2)
  n = n_vect(i) ;

  % Calculation of the Chebyshev nodes
  ii = 0:n;
  x_cap = -cos(pi*ii/n);
  x = 0.5*(a+b) + 0.5*(b-a)*x_cap;
  y = f(x);

  coef = polyfit(x, y, n);
  yy = polyval(coef, xx);

  plot(xx, f(xx), 'k-', xx, yy, 'r-', x, y, 'rx', 'LineWidth', 2, 'MarkerSize', 8)
  axis([a-0.2 b+0.2 -0.1+min(f(xx)) 0.1+max(f(xx))])
  set(gca,'FontSize', 16)
  set(gca,'LineWidth', 1.5)

  error(i) = max(abs(f(xx) - yy));


  pause

end

order = (log(error(1:end-1) ./ error(2:end))/log(2))'
```

Exercise 8.3. Consider the function

$$f(x) = \left|x - \frac{\pi}{12}\right| \qquad -1 \leq x \leq 1.$$

a. Verify that the interpolating polynomials $\Pi_n f(x)$ based on equally spaced nodes exibit the Runge's phenomenon.

b. Compute the piecewise linear interpolation (`interp1` command) based on $n = 1, 2, 4, 8, 16, 32$ intervals and compute the associated error with the respect to infinity norm. Is the quadratic convergence ensured in such a case?

c. Interpolate the function using a piecewise cubic spline (`spline` command) with $n = 1, 2, 4, 8, 16, 32$ subintervals. Plot the result.

d. Interpolate the function using the least square approach bu setting the number of nodes $n = 10$ and varying the degree of the interpolant. Plot the results and check when the least-square approximation exactly matches the values y_nodes in correspondence with the x_nodes.

Solution Exercise 8.3.

ex_8_3.m

```
clc
clear all
close all

% Setting the function and its visualization
a = -1;
b = 1;
f = @(x) abs(x - pi/12);

xx = linspace(a, b, 1000);
fxx = f(xx);

figure
subplot(1,2,1)
hold on, box on
plot(xx, fxx, 'k-', 'LineWidth',2)
set(gca,'FontSize',16)
set(gca,'LineWidth',1.5)
xlabel('x','FontSize',16)
ylabel('f(x)','FontSize',16)

% a) The interpolating polynomials  n f (x ) using equally
% spaced nodes show the same Runges phenomenon

for k = [2:2:10]
  x = linspace(a, b, k);
  fx = f(x);
  coef = polyfit(x, fx, k-1);
  yy = polyval(coef, xx);
  plot(xx, yy, 'r-', 'LineWidth',2)
  axis([a-0.2 b+0.2 -0.1+min(f(xx)) 0.1+max(f(xx))])
  pause
end

subplot(1,2,2)
hold on, box on
plot(xx, fxx, 'k-', 'LineWidth',2)
set(gca,'FontSize',16)
set(gca,'LineWidth',1.5)
xlabel('x','FontSize',16)
ylabel('f(x)','FontSize',16)

for k = [12:2:16]
  x = linspace(a, b, k);
```

```
  fx = f(x);
  coef = polyfit(x, fx, k-1);
  yy = polyval(coef, xx);
  plot(xx, yy, 'r-', 'LineWidth',2)
  axis([a-0.2 b+0.2 -0.1+min(f(xx)) 0.1+max(f(xx))])
  pause
end

% The interpolation polynomials approach the function in the middle of the
% interval, but close to the boundaries increasing oscillations appears.

% b) Computation of the piecewise linear interpolation (interp1 command)
% and calculation of the convergence order ( f(x) is not C2(I) function!!)

clear all
a = -1;
b = 1;
f = @(x) abs(x - pi/12);
x_plot = linspace(a, b, 101);

n_vect = [1 2 4 8 16 32 64 128];

figure
for (i = 1:numel(n_vect))
  n = n_vect(i)

  x = linspace(a, b, n+1);
  y = f(x);

  y_plot = interp1(x, y, x_plot);
  plot(x_plot, f(x_plot), 'k-', x_plot, y_plot, 'r-', x, y, 'rx', 'LineWidth', 2, 'MarkerSize'
  , 8)
  axis([a-0.2 b+0.2 -0.1+min(f(x_plot)) 0.1+max(f(x_plot))])
  set(gca,'FontSize', 16)
  set(gca,'LineWidth', 1.5)

  %error(i) = max(abs(f(x_plot) - y_plot));
  error(i) = norm((f(x_plot) - y_plot),'inf');
  pause
end

% detetmination of the element width
H = (b-a)./n_vect;

figure
loglog(H, error, 'bx-', 'LineWidth', 2, 'MarkerSize', 8)
hold on, box on
loglog(H, H.^2, 'k-', 'LineWidth', 2)
%axis([0.7 600 1e-4 3])
set(gca,'FontSize',16)
set(gca,'LineWidth',1.5)
xlabel('n','FontSize',16)
ylabel('error','FontSize',16)
legend('error', 'O(h^2)', 'Location', 'ne');

% A second order convergence cannot be expected, since the function is not smooth enough.
% Calculation of the error
order = (log(error(1:end-1) ./ error(2:end))/log(2))'
% or using the diff command build in in matlab
p = -diff(log(error)) / log(2)

% c) Interpolate the function using a piecewise cubic spline (spline
% command) with n = 1, 2, 4, 8, 16, 32 elements.

figure
for (i = 1:numel(n_vect))
  n = n_vect(i)

  x = linspace(a, b, n+1);
  y = f(x);
```

```
  y_plot = spline(x, y, x_plot);
  plot(x_plot, f(x_plot), 'k-', x_plot, y_plot, 'r-', x, y, 'rx', 'LineWidth', 2, 'MarkerSize'
  , 8)
  axis([a-0.2 b+0.2 -0.1+min(f(x_plot)) 0.1+max(f(x_plot))])
  set(gca,'FontSize', 16)
  set(gca,'LineWidth', 1.5)

  pause
end

% d) Interpolation of the points generated with the above function with nodes n = 10
% and using the least square approach with varying the degree of the interpolant.

close all

a = -1;
b = 1;
f = @(x) abs(x - pi/12);
x_plot = linspace(a, b, 101);
x_nodes = linspace(a, b, 10);
y_nodes = f(x_nodes);

plot(x_nodes, y_nodes, 'rO');
hold on;

for ii = 2:1:15
   v = polyfit(x_nodes, y_nodes, ii);
   y_plot = polyval(v, x_plot);
   plot(x_plot, y_plot)
   pause
end
```