MAT237 Numerical Analysis

Manuel Loaiza Vasquez

October 2021

Pontificia Universidad Católica del Perú
Lima, Perú
manuel.loaiza@pucp.edu.pe

Solutions to midterm exam. Instructor: Ruben Agapito.

1 Roots of nonlinear equations

1. Use a plot to approximately locate all the roots of $f(x) = x^{-2} - \sin(x)$ in the interval $[1/2, 4\pi]$. Then find a pair of initial points for each root such that

$$q(x) = f(x_k) + \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k).$$

converges to that root.

Solution. We want to find roots of the function f

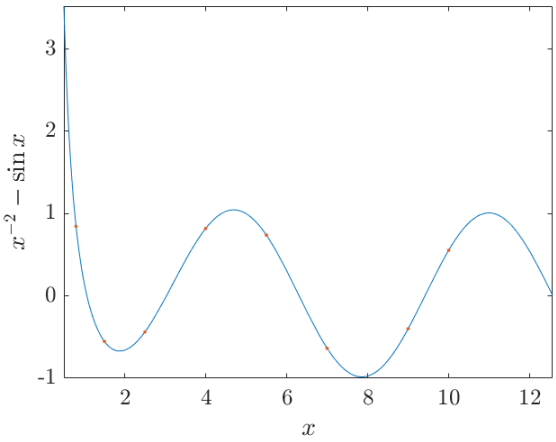

Figure 1: Plot of f in $[1/2, 4\pi]$

From the graph, it is clear that there are roots in the intervals $[0.8, 1.5]$, $[2.5, 4]$, $[5.5, 7]$ and $[9, 10]$. Using the secant method

```
function x = secant(f, x1, x2)
% Secant method for a scalar equation.
% Input:
%   f: objective function
%   x1, x2: initial root approximations
% Output
%   x: vector of root approximations (last is best)

% Operating parameters.
funtol = 100 * eps;
xtol = 100 * eps;
maxiter = 40;

x = [x1 x2];
dx = Inf;
y1 = f(x1);
k = 2;
y2 = 100;

while (abs(dx) > xtol) && (abs(y2) > funtol) && (k <
    maxiter)
  y2 = f(x(k));
  dx = -y2 * (x(k) - x(k - 1)) / (y2 - y1); % secant step
  x(k + 1) = x(k) + dx;
  k = k + 1;
  y1 = y2; % current f-value becomes the old one next
      time
end

if k == maxiter, warning('Maximum number of iterations
    reached.'), end
```

the following code

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

f = @(x) x.^(-2) - sin(x);
interval = [1 / 2, 4 * pi];
```

```
fplot(f, interval);

x_axis = [0.8, 1.5, 2.5, 4, 5.5, 7, 9, 10];
y_axis = f(x_axis);
hold on, plot(x_axis, y_axis, '.');

for i = 1 : 4
    root = secant(f, x_axis(2 * i - 1), x_axis(2 * i));
    fprintf('root %d : %.10f\n', i, root(end));
end
```

outputs the approximated roots

```
root 1 : 1.0682235442
root 2 : 3.0326454184
root 3 : 6.3083168253
root 4 : 9.4134928032
```

□

2. Write a function that performs inverse quadratic interpolation for finding a root of f, give three initial estimates. To find the quadratic polynomial $q(y)$ passing through the three most recent points, use the built-in `polyfit` command. Test your function on the first problem from this section.

Solution. The following function performs inverse quadratic interpolation

```
function x = iqi(f, x1, x2, x3)
  % Inverse quadratic interpolation method for a scalar
      equation.
  % Input:
  %   f: objective function
  %   x1, x2, x3: initial estimates
  % Output
  %   x: vector of root approximations (last is best)

  % Operating parameters.
  funtol = 100 * eps;
  xtol = 100 * eps;
  maxiter = 40;

  x = [x1 x2 x3];
  y = f(x);
  k = 0;
  dx = abs(x3 - x2);

  while (abs(dx) > xtol) && (abs(y(end)) > funtol) && (k
     < maxiter)
```

```
        coefficients = polyfit(y(k + 1 : k + 3), x(k + 1 :
            k + 3), 2);
        x(k + 4) = polyval(coefficients, 0);
        y(k + 4) = f(x(k + 4));
        k = k + 1;
        dx = abs(x(end) - x(end - 1));
    end

    if k == maxiter, warning('Maximum number of iterations
        reached.'), end
```

Examples 4.4.1 and 4.4.2 analyze the function $f(x) = xe^x - 2$ and Example 4.4.3 the function $f(x) = x + \cos(10x)$ with real domain. We are going to test our function with these examples

```
clc;
f = @(x) x.*exp(x) -2;
x = [0.5 0.8 1];
estimates = iqi(f, x(1), x(2), x(3));
fprintf('Example 4.4.1 root : %.10f\n', estimates(end));

f = @(x) x + cos(10 * x);
x = [0.8 1.2 1];
estimates = iqi(f, x(1), x(2), x(3));
fprintf('Example 4.4.3 root : %.10f\n', estimates(end));
```

obtaining the following roots:

```
Example 4.4.1 root : 0.8526055020
Example 4.4.3 root : 0.9678884018
```

□

2 Polynomial interpolation

3. In each part, use

$$\Phi(x) = \prod_{i=0}^{n} (x - t_i)$$

to interpolate the given function using $n + 1$ evenly spaced nodes in the given interval. Plot each interpolant together with the exact function.

Solution. We are going to use the barycentric formula for the interpolating polynomial

```
function p = polyinterp(t, y)
  % Polynomial interpolation by the barycentric formula.
  % Input:
```

```
%   t: interpolation nodes (vector, length n + 1)
%   y: interpolation values (vector, length n + 1)
% Output:
%   p: polynomial interpolant (function)

t = t(:);  % column vector
n = length(t) - 1;
C = (t(end) - t(1)) / 4;  % scaling factor to ensure
    stability
tc = t / C;

% Adding one node at a time, compute inverses of the
    weights.
omega = ones(n + 1, 1);
for m = 1 : n
  d = (tc(1 : m) - tc(m + 1));  % vector of node
      differences
  omega(1 : m) = omega(1 : m).*d;  % update previous
  omega(m + 1) = prod(-d);  % compute the new one
end
w = 1./omega;  % go from inverses to weights

p = @evaluate;

function f = evaluate(x)
  % Compute interpolant, one value of x at a time.
  f = zeros(size(x));
  for j = 1:numel(x)
    terms = w ./ (x(j) - t );
    f(j) = sum(y.*terms) / sum(terms);
  end

  % Apply L'Hopital's Rule exactly.
  for j = find(isnan(f(:)))'  % divided by zero here
    [~,idx] = min(abs(x(j) - t));  % node closest to x(
        j)
    f(j) = y(idx);  % value at node
  end
end

end
```

□

b. $f(x) = \tanh x$, $n = 2, 3, 4$, $x \in [1, 2]$.

Solution. With the following code

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

f = @(x) tanh(x);
interval = [0.5 2];
fplot(f, interval), hold on

for n = 2 : 4
  x = linspace(interval(1), interval(2), n + 1)';
  p = polyinterp(x, f(x));
  fplot(p, interval), hold on
end

hold off
```

we obtain

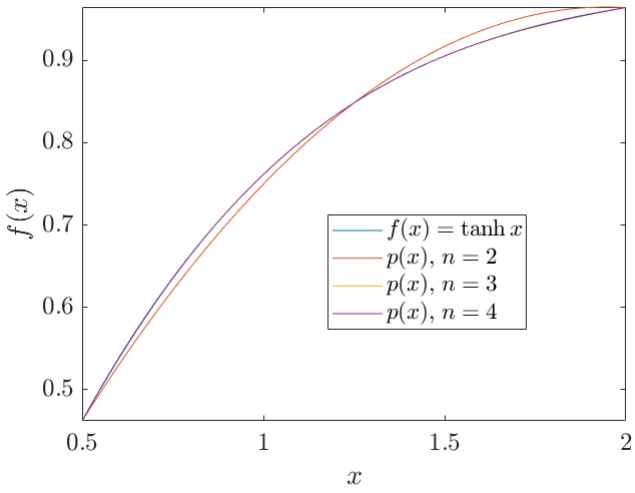

Figure 2: Polynomial interpolations of $\tanh x$ in $[0.5, 2]$

□

d. $f(x) = |x|$, $n = 2, 3, 4$, $x \in [-2, 1]$.

Solution. With the following code

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

f = @(x) abs(x);
interval = [-2 1];
fplot(f, interval), hold on

for n = 2 : 4
  x = linspace(interval(1), interval(2), n + 1)';
  p = polyinterp(x, f(x));
  fplot(p, interval), hold on
end

hold off
```

we obtain

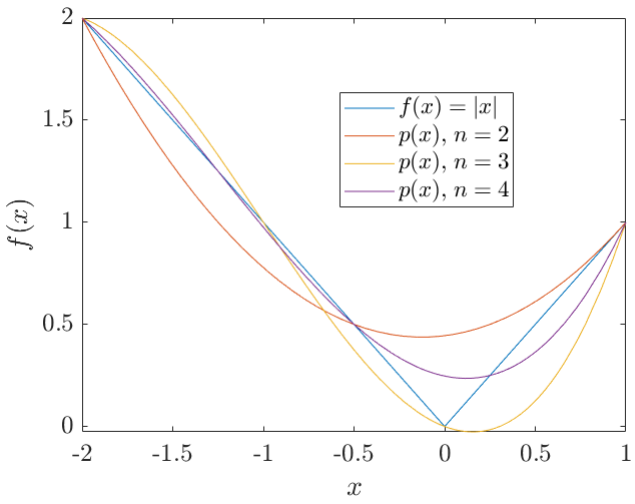

Figure 3: Polynomial interpolations of $|x|$ in $[-2, 1]$

□

4. Compute the barycentric weights numerically using $n + 1$ equally spaced nodes in $[-1, 1]$ for $n = 30, 60, 90$. On a single graph, plot their absolute values

as a function of t_i (which always ranges between -1 and 1) using a log-linear scale.

Solution. Modifying the barycentric interpolation function to return only the vector of weights

```
function w = barycentric_weights(t)
  % Polynomial interpolation by the barycentric formula.
  % Input:
  %   t: interpolation nodes (vector, length n + 1)
  % Output:
  %   w: weights (vector, length n + 1)

  t = t(:);  % column vector
  n = length(t) - 1;
  C = (t(end) - t(1)) / 4;  % scaling factor to ensure
      stability
  tc = t / C;

  % Adding one node at a time, compute inverses of the
      weights.
  omega = ones(n + 1, 1);
  for m = 1 : n
    d = (tc(1 : m) - tc(m + 1));  % vector of node
        differences
    omega(1 : m) = omega(1 : m).*d;  % update previous
    omega(m + 1) = prod(-d);  % compute the new one
  end
  w = 1./omega;  % go from inverses to weights
end
```

we plot the absolute values of the weights using a log-linear scale.

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

interval = [-1 1];

for n = 30:30:90
    x = linspace(interval(1), interval(2), n + 1);
    w = barycentric_weights(x);
    semilogy(x, abs(w)), hold on
end
```

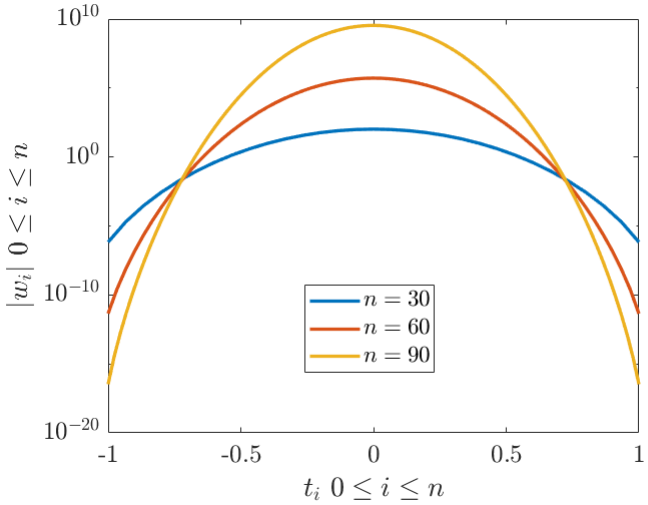

Figure 4: Absolute value of barycentric weights

□

5. For each case, compute the polynomial interpolant using n second-kind Chebyshev nodes in $[-1, 1]$ for $n = 4, 8, 12, \ldots, 60$. At each value of n, compute the infinity-norm error (that is, $\max |p(x) - f(x)|$ evaluated for at least 4000 values of x). Using a log-linear scale, plot the error as a function of n, then determine a good approximation to the constant k in

$$\max_{x \in [-1,1]} |f(x) - p(x)| \leq ck^{-n},$$

where f is analytic in an open real interval containing $[-1, 1]$, $c > 0$ and $k > 1$.

a. $f(x) = 1/(25x^2 + 1)$.

Solution. Computing the infinity-norm error of 4001 evenly spaced points between -1 and 1 of each polynomial interpolant, a good approximation for k is 1.85 using $c = 1$.

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

f = @(x) 1./(25 * x.^2 + 16);
```

```
10
11 interval = [-1 1];
12 x = linspace(interval(1), interval(2), 4001);
13 nodes = linspace(4, 60, 4);
14 len = length(nodes);
15 ines = zeros(len);
16
17 for k = 1 : len
18   n = nodes(k);
19   p = chebinterp(f, n);
20   % Infinite norm error: max|p(x) - f(x)|
21   ine = 0;
22   for value = x
23     ine = max(ine, abs(p(value) - f(value)));
24   end
25   ines(k) = ine;
26 end
27
28 g = @(x) 1./((1.85).^x);
29 semilogy(nodes, ines), hold on
30 semilogy(nodes, g(nodes));
```

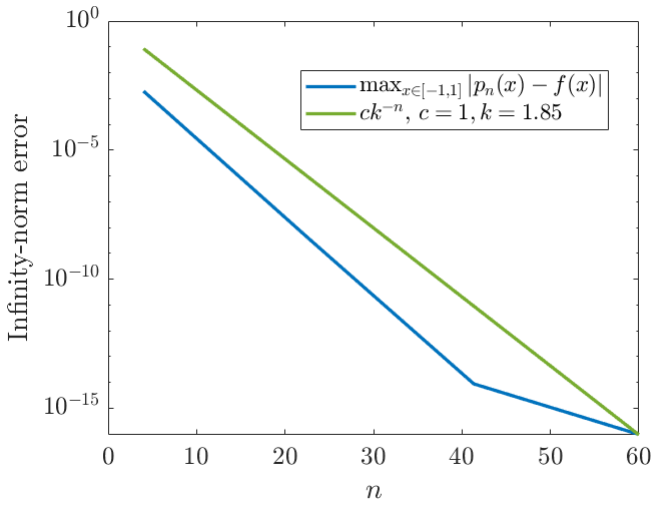

Figure 5: Infinity norm error of $f(x) = 1/(25x^2 + 1)$ and a bounding function

□

c. $f(x) = \cosh(\sin x)$.

Solution. Computing the infinity-norm error of 4001 evenly spaced points between -1 and 1 of each polynomial interpolant, a good approximation for k is 1.75 using $c = 1$.

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

f = @(x) cosh(sin(x));

interval = [-1 1];
x = linspace(interval(1), interval(2), 4001);
nodes = linspace(4, 60, 4);
len = length(nodes);
ines = zeros(len);

for k = 1 : len
  n = nodes(k);
  p = chebinterp(f, n);
  % Infinite norm error: max|p(x) - f(x)|
  ine = 0;
  for value = x
    ine = max(ine, abs(p(value) - f(value)));
  end
  ines(k) = ine;
end

g = @(x) 1./((1.85).^x);
semilogy(nodes, ines), hold on
semilogy(nodes, g(nodes));
```

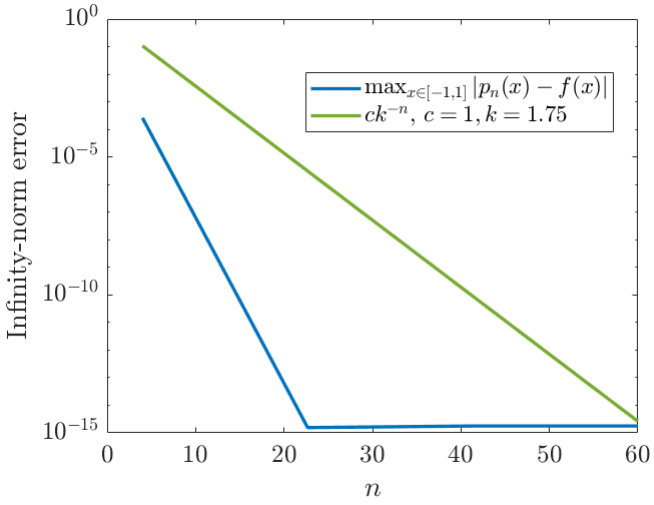

Figure 6: Infinite norm error of $f(x) = \cosh(\sin(x))$ and a bounding function

□

6. Write a function `function p = chebinterp(f, n)` that returns a function representing the polynomial interpolant of the function f using $n+1$ Chebyshev second-kind nodes (assuming $-1 \le x \le 1$). You should use

$$w_k = (-1)^k d_k, d_k = \begin{cases} 1/2 & \text{if } k = 0 \text{ or } k = n, \\ 1 & \text{otherwise} \end{cases}$$

to compute the barycentric weights directly, rather than using the method of finding ω and w in the barycentric formula. Test your function by revisiting the example with $f(x) = 1/(x^2 + 16)$ to use Chebyshev rather than equally spaced nodes.

Solution. The following function

```
function p = chebinterp(f, n)
  % Polynomial interpolation by Chebyshev points of the
      second kind.
  % Input:
  %   f: objective function (dom(f) = [-1, 1])
  %   n: n + 1 Chebyshev second-kind nodes
  % Output:
  %   p: polynomial interpolant (function)
   theta = linspace(0, pi, n + 1)';
   nodes = -cos(theta);
   p = polychebinterp(nodes, f(nodes));
end
```

uses the Chebyshev second-kind nodes to compute the polynomial interpolant with the barycentric formula computing the barycentric weights directly

```
function p = polychebinterp(t, y)
  % Polynomial interpolation by the barycentric formula
  % using Chebyshev points of the second kind.
  % Input:
  %   t: interpolation nodes (vector, length n + 1)
  %   y: interpolation values (vector, length n + 1)
  % Output:
  %   p: polynomial interpolant (function)

  t = t(:);  % column vector
  n = length(t) - 1;
  C = (t(end) - t(1)) / 4;  % scaling factor to ensure
      stability
  tc = t / C;

  % Barycentric weights using Chebyshev nodes
  w = ones(n + 1, 1);
  factor = -1;
  for k = 1 : n + 1
    if k == 1 || k == n + 1
      d = 1 / 2;
    else
      d = 1;
    end
    w(k) = factor * d;
    factor = factor * (-1);
  end

  p = @evaluate;

  function f = evaluate(x)
    % Compute interpolant, one value of x at a time.
    f = zeros(size(x));
    for j = 1:numel(x)
      terms = w ./ (x(j) - t );
      f(j) = sum(y.*terms) / sum(terms);
    end

    % Apply L'Hopital's Rule exactly.
    for j = find(isnan(f(:)))'  % divided by zero here
      [~,idx] = min(abs(x(j) - t));  % node closest to x(
          j)
      f(j) = y(idx);  % value at node
```

```
    end
  end

end
```

In order to test our function, we are going to plot the polynomials p obtained using $n+1$ nodes, for $n = 1, \ldots, 5$. The decision of choosing small values for n was because from values greater than 4 the polynomials are almost f and we are not going to notice any differences in the plot.

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

f = @(x) 1./(x.^2 + 16);
interval = [-1 1];
fplot(f, interval), hold on
for n = 1 : 5
  p = chebinterp(f, n);
  fplot(p, interval), hold on
end
hold off
```

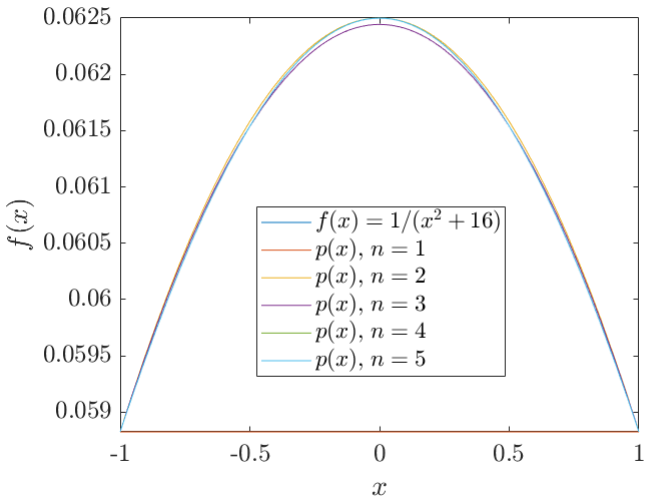

Figure 7: Polynomial interpolations of $1/(x^2+16)$ in $[-1, 1]$

□

7. Theorem 9.3.1 assumes that the function being approximated has infinitely many derivatives over $[-1,1]$. But now consider the family of functions $f_m(x) = |x|^m$.

a. How many continuous derivatives over $[-1,1]$ does f_m possess?

Solution. If m is even, we can write $m = 2k$ with k positive integer. Then

$$f_m(x) = |x|^{2k} = (|x|^2)^k = (x^2)^k = x^{2k} = x^m,$$

and because of being polynomial, f_m is infinitely differentiable and all its derivatives are continuous. When m is odd, we can write $m = 2k-1$ with k positive integer. Then

$$f_m(x) = |x|^{2k-1} = |x|\,|x|^{2k-2} = |x|\,x^{2k-2} = |x|\,x^{m-1}.$$

Theorem 1. *Given $f : [-1,1] \to \mathbb{R}$ with $f(x) = c_n|x|x^n$, where c is a constant positive number, has exactly n continuous derivatives.*

Prueba. Let's prove it by induction. When $n = 1$, the derivative in $(0,1]$ is $2cx$ and the derivative in $[-1,0)$ is $-2cx$. By definition, at $x = 0$ we have

$$\lim_{h\to 0} \frac{|h|h - 0}{h} = \lim_{h\to 0} |h| = 0.$$

so the derivative is $2c|x|$ which is continous and it doesn't have more derivatives because $|x|$ is not differentiable at $x = 0$. Suppose that $c_n|x|x^n$ has exactly n continuous derivatives. Analyzing $c_{n+1}|x|x^{n+1}$, in $(0,1]$ we have $(n+2)c_{n+1}x^{n+1}$, in $[-1,0)$ we have $-(n+2)c_{n+1}x^{n+1}$. By definition, at $x = 0$ we have

$$\lim_{h\to 0} \frac{c_{n+1}|h|h^{n+1}}{h} = \lim_{h\to 0} \frac{c_{n+1}|h|h}{h} = 0.$$

Then, the derivative is $(n+1)c_{n+1}|x|x^n = c_n|x|x^n$. This function is continuous in our interval because is a product of continous functions. Also, because of the inductive hypothesis, this function has n continuous derivative, so in total we have $n+1$ continuous derivatives. □

Using the theorem, we can conclude that f_m has $m-1$ continuous derivatives when m is odd. □

b. Compute the polynomial interpolant using n second-kind Chebyshev nodes in $[-1,1]$ for $n = 10, 20, 30, \ldots, 100$. At each value of n, compute the infinity-norm error. (that is, $\max |p(x) - f_m(x)|$ evaluated for at least 41000 values of x). Using a single log-log graph, plot the error as a function of n for all six values $m = 1, 3, 5, 7, 9, 11$.

Solution. We create a function called `abspow` defined as $\texttt{abspow}(x, m) = |x|^m$ to fix the exponent on each iteration and then use it as a one variable function of x in the interpolation.

```
function y = abspow(x, m)
  y = abs(x).^m;
end
```

Then, for every odd value of m between 1 and 11, we are going to plot a log-log graph of the infinity-norm errors.

```
clc;

set(0,'defaulttextInterpreter','latex')
set(0, 'defaultAxesTickLabelInterpreter','latex');
set(0, 'defaultLegendInterpreter','latex');
set(0, 'defaultAxesFontSize', 15)
set(0, 'defaultLineLineWidth', 2.0);

interval = [-1 1];
x = linspace(interval(1), interval(2), 41001);
nodes = 10 : 10 : 100;
len = length(nodes);

for m = 1 : 2 : 11
  % Fix f(x) = |x|^m
  f = @(x) abspow(x, m);
  ines = zeros(len);
  % For each n
  for k = 1 : len
    n = nodes(k);
    p = chebinterp(f, n);
    ine = 0;
    % Find max |p(x) - f(x)|
    for value = x
      ine = max(ine, abs(p(value) - f(value)));
    end
    ines(k) = ine;
  end
  loglog(nodes, ines), hold on
end

hold off
```

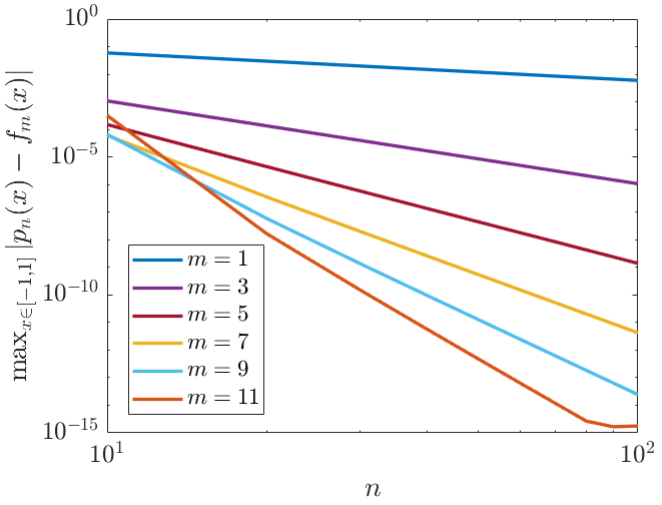

Figure 8: Infinity-norm error of $f(x) = |x|^m$ in $[-1, 1]$

□

c. Based on the result of parts (a) and (b), form a hypothesis about the asymptotic behaviour of the error for fixed m as $n \to \infty$.

Solution. From the graph in part (b), for a fixed value of m, we can form a hypothesis that

$$\ln \max_{x\in[-1,1]} |f(x) - p(x)| = -\alpha \ln n + \beta$$

with $\alpha > 0$ and $0 < \beta < 1$. The infinity-norm error of f is

$$\begin{aligned} \max_{x\in[-1,1]} |f(x) - p(x)| &= \exp(-\alpha \ln n + \beta) \\ &= \exp(-\alpha \ln n) \exp(\beta) \\ &= \exp(\beta)\, n^{-\alpha}. \end{aligned}$$

.

The theorem in the book requires an analytic function, however, even though our function only has $m - 1$ derivatives, experimentally the method converges and the infinity-norm error is $\mathcal{O}(n^{-\alpha})$. □